Министерство науки и высшего образования Российской Федерации
ФГБОУ ВО «Тульский государственный университет»
Технический колледж им. С.И. Мосина

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

по дисциплине «Базы данных»

специальности
09.02.01 Компьютерные системы и комплексы

Тула 2023

УТВЕРЖДЕНЫ
Цикловой комиссией информационных технологий
Протокол от «13» января 2023 г. № 6
Председатель цикловой комиссии И.В. Миляева

Лабораторная работа № 1

СОЗДАНИЕ БАЗЫ ДАННЫХ СРЕДСТВАМИ СУБД. СОЗДАНИЕ ТАБЛИЦ

1. ЦЕЛЬ РАБОТЫ

Изучение средств автоматизации конструирования реляционных таблиц в СУБД LibreOffice Base.

2. ОСНОВЫ ТЕОРИИ

Создание базы данных

База данных (БД) представляет собой совокупность средств для ввода, хранения, просмотра, выборки и управления информацией. К этим средствам относятся таблицы, формы, отчеты, запросы.

Поддерживаются два способа создания базы данных. Вы можете создать пустую базу данных, а затем добавить в нее таблицы, формы, отчеты и другие объекты. Такой способ является наиболее гибким, но требует отдельного определения каждого элемента базы данных. Кроме этого имеется возможность создать базу данных с помощью мастера со всеми необходимыми формами, таблицами и отчетами.

Создание новой базы данных осуществляется командой **Файл | Создать** . После ввода имени создаваемой базы и нажатия кнопки **Создать** откроется окно базы данных. Оно состоит из шести вкладок, которые пока пусты. Необходимо далее создать все компоненты, входящие в базу данных. Их перечень соответствует ярлыкам вкладок окна базы данных.

Основным компонентом базы данных являются таблицы. Они хранят всю информацию, помещаемую в БД. В таблицы будет водится информация, которая может дополняться изменяться и удаляться.

Создание таблицы осуществляется в окне БД. Создать таблицу можно несколькими способами: использовать мастер таблиц, создать таблицу в режиме конструктора таблиц, импортировать таблицу из внешнего файла. В любом случае каждой таблице присваивается определенное имя.

Создание таблицы в окне конструктора

Создание таблиц в окне конструктора предоставляет большие возможности и осуществляется выбором из списка вариантов значения **Конструктор.** В результате выполнения этих действий откроется окно конструктора. В верхней части окна диалога находится таблица, которая содержит следующие атрибуты: наименование поля, тип данных и описание. Наименование каждого из полей таблицы выбирается произвольно в соответствии с помещаемой в него типом информации.

Наименование поля должно быть уникально может содержать до 64

символов, исключая точку (.), восклицательный знак (!), прямых скобок ([]) и управляющих символов с кодами ASCII – 0-31.

Тип поля определяется типом данных, хранящихся в этом поле. В допустимы следующие типы: текстовый (до 255 символов), числовой, денежный (8 байт, до 4 знаков после запятой), счетчик (данные не редактируются), дата/время, логический, поле MEMO (до 64000 символов), поле объекта OLE (размер определяется объемом жесткого диска), мастер подстановок (создает поле, в котором предполагается выбор значений из раскрывающегося списка значений других таблиц).

Каждый из типов поля наделен собственными свойствами, которые отображаются в разделе «Свойства поля» окна конструктора.

Создать структуру таблицы можно следующим образом:

1. В окне конструктора в столбце **Имя поля** вводится имя поля данных

2. В столбец **Тип данных** из раскрывающегося списка вводится значение типа данных.

3. Столбец Описание представляет из себя пояснение, которое вы даете своим полям. Это пояснение появляется в строке состояния во время работы с БД.

4. Аналогичным образом вводится описание всех полей таблицы.

5. Завершив ввод структуры, ее надо сохранить, выполнив команду **Файл|Сохранить .**

Создание таблицы в режиме таблицы

Рассмотрим способ создания таблиц, который отличается своей простотой и наглядностью. Приведем последовательность действий, которую предстоит выполнить:

1) Перейдите на вкладку «Таблицы» окна базы данных и нажмите кнопку Создать. 2) В окне диалога «Новая таблица» выберите из списка вариантов значение **Режим таблицы** и нажмите кнопку **ОК.**

В результате выполнения этих действий откроется окно диалога «Таблица», содержащее созданную по умолчанию таблицу. Эта таблица содержит 20 столбцов и 30 строк, и этого вполне достаточно для начала. После сохранения этой таблицы, конечно, можно добавить столько строк и столбцов, сколько вам понадобится. 3. Наименования полей таблицы определены по умолчанию, но очень просто присвоить полям новые имена. Для этого нажмите дважды кнопкой мыши на область выбора первого поля (заголовок которого содержит Поле1). Имя поля выделяется и появляется Мигающий курсор. Введите имя первого поля и нажмите клавишу TAB. Аналогично введите остальные имена полей вашей таблицы в следующих столбцах.

4. Теперь заполните несколько строк вашей таблицы, вводя информацию в том виде, в каком она будет вводиться и в будущем. Старайтесь записывать все в одном стиле (например, если первую дату вы записали 10/14/96, то не пишите следующую в виде Ноябрь 3, 1996). Конечно, если установит

неправильный тип данных, вы можете позже его изменить, но и вам желательно стараться вводить все правильно с самого начала.

5. Сохраните таблицу, выполнив команду **Файл | Сохранить макет** или нажав кнопку **Сохранить** на панели инструментов, В открывшемся окне диалога «Сохранение» присвойте таблице имя и нажмите кнопку **ОК.**

6. На запрос о необходимости создания для таблицы первичного ключа нажмите кнопку **Да**, и создаст таблицу, удалив лишние строки и столбцы.

7. Теперь убедитесь, что выбрал для каждого поля правильные типы данных. Для этого перейдите в окно конструктора таблицы, выполнив команду Вид | Конструктор таблиц. Если вас что-то не устраивает в структуре таблицы, внесите необходимые изменения.

3. ОБЪЕКТЫ И СРЕДСТВА ИССЛЕДОВАНИЯ

Предметная область в соответствии с вариантом задания. ПК типа IBM PC, дисплейный класс, операционная система Windows XP и выше, СУБД OO Base.

4. ПОДГОТОВКА К РАБОТЕ

Ознакомиться с теоретическими положениями лабораторной работы.

5. ПРОГРАММА РАБОТЫ

По предложенному преподавателем варианту разработать необходимые таблицы БД и заполнить их информацией. Оформить отчет.

ВАРИАНТЫ ЗАДАНИЙ . Выбирается преподавателем из списка заданий (См. Приложение) и остается единым для всех лабораторных работ.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

6.1. Назовите основные компоненты базы данных ?

6.2. Основные способы создания таблиц?

6.3. Создание таблиц в режиме мастера?

6.4. Создание таблиц в режиме таблиц?

Лабораторная работа № 2

РАБОТА С ТАБЛИЦАМИ: ДОБАВЛЕНИЕ, РЕДАКТИРОВАНИЕ, УДАЛЕНИЕ, НАВИГАЦИЯ ПО ЗАПИСЯМ

1. ЦЕЛЬ РАБОТЫ

Изучение средств модификации реляционных таблиц в СУБД LibreOffice Base.

2. ОСНОВЫ ТЕОРИИ

Модификация структуры таблицы

Описанную структуру таблицы, не сделав при этом ни одной ошибки, вам необходимо сохранить.

Для исправления возможных допущенных ошибок предоставляет необходимые средства. К их числу относятся:

1) Изменение наименования поля и/или его типа . 2) Вставка пропущенного поля 3)Удаление ошибочно введенного поля. 4) Изменение порядка следования полей в таблице

Для модификации структуры таблицы, входящей в базу данных, установите в окне базы данных указатель на модифицируемую таблицу и нажмите кнопку **Конструктор.**

Изменение наименования поля или его типа осуществляется после установки указателя на данное поле или тип, который надо ввести. Неправильные символы удаляются клавишей Delete. После этого вводятся правильные символы.

Изменение порядка следования полей осуществляется нажатием на область выбора поля и после выделения строки и повторного нажатия левой кнопки мыши перетаскиванием строки (столбца) в нужное место.

Удаление полей таблицы осуществляется клавишей Delete после их выделения.

Добавление нового поля осуществляется командой **Вставка | Поле.**

Установка первичного ключа

Когда напоминает вам об отсутствии первичного ключа и предлагает его создать, нажмите Да. добавит поле код в первой строке описания структуры таблицы.

Если вы хотите установить первичный ключ самостоятельно, нажмите на поле, которое будет в дальнейшем использоваться в качестве первичного ключа. Затем нажмите правую кнопку мыши и выберите пункт всплывающего **Первичный** ключ. В области выбора поля, которое будет использоваться как ключ, появится маленькая пиктограмма с изображением первичного

ключа. Для установки первичного ключа также можно использовать кнопку Ключевое поле на панели инструментов.

3. ОБЪЕКТЫ И СРЕДСТВА ИССЛЕДОВАНИЯ

Предметная область в соответствии с вариантом задания. ПК типа IBM PC, дисплейный класс, операционная система Windows XP и выше, СУБД OO Base.

4. ПОДГОТОВКА К РАБОТЕ

Ознакомиться с теоретическими положениями лабораторной работы.

5. ПРОГРАММА РАБОТЫ

По предложенному преподавателем варианту заполнить таблицы информацией, добавить, отредактировать записи. При построении таблиц используйте индексацию записей. Оформить отчет.

ВАРИАНТЫ ЗАДАНИЙ . Выбирается преподавателем из списка заданий (См. Приложение) и остается единым для всех лабораторных работ.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

6.1. Расскажите о назначении и формах индексации записей БД.

6.2. Как осуществляется навигация в таблицах БД?

Лабораторная работа № 3
СОРТИРОВКА, ПОИСК И ФИЛЬТРАЦИЯ ДАННЫХ

1. ЦЕЛЬ РАБОТЫ

Изучение средств автоматизации сортировки, поиска данных в реляционных таблицах.

2. ОСНОВЫ ТЕОРИИ

Одним из основных требований, предъявляемых к СУБД, является возможность быстрого поиска требуемых записей среди большого объема информации. Индексы представляют собой наиболее эффективное средство, которое позволяет значительно ускорить поиск данных в таблицах по сравнению с таблицами, не содержащими индексов. В зависимости от количества полей, используемых в индексе, различают *простые* и *составные индексы.*

В допускается создание произвольного количества индексов. Индексы создаются при сохранении макета таблицы и автоматически обновляются при вводе и изменении записей. Можно в любое время добавить новые или удалить ненужные индексы в окне конструктора таблиц.

Требование уникальности индекса в не является обязательным. Для ускорения поиска требуемой информации могут быть использованы индексы, не являющиеся уникальными

Важной особенностью индексов является то, что можно использовать индексы для создания *первичных ключей.* Первичный ключ содержит информацию, которая однозначно идентифицирует запись. В этом случае индексы должны быть уникальными. Это означает, что для таблицы, содержащей только одно индексное поле, уникальными должны быть значения этого поля. Для составных индексов величины в каждом из индексных полей могут иметь повторяющиеся значения. Однако индексное выражение должно быть уникальным.

Прежде чем определить первичный ключ в таблице, просмотрите все поля создаваемой вами таблицы. Есть ли хоть одно поле, информация которого была бы уникальна для каждой записи? Даже если использовать полное имя, то есть фамилию, имя и отчество одновременно, оно также не будет неповторимым.

Часто наилучшее решение этой проблемы заключается в том, чтобы каждой записи в таблице поставить в соответствие идентификационный номер. Это и делает , когда вы предлагаете ему создать первичный ключ. Он создает поле Код с типом данных **Счетчик.** Это означает, что каждый раз при создании новой записи значение счетчика увеличивается на 1. Этот номер и является первичным ключом для каждой новой записи.

Создание индекса для одного поля

Для создания простого индекса используется свойство поля **Индексированное** поле, позволяющее ускорить выполнение поиска и сортировки записей по одному полю таблицы. Индексированное поле может содержать как уникальные, так и повторяющиеся значения.

Данное свойство поля **Индексированное поле** может принимать следующие значения:

Нет Значение по умолчанию. Индекс не создается.

Да (Допускаются совпадения) В индексе допускаются повторяющиеся значения

Да (Совпадения не допускаются) Повторяющиеся значения в индексе не допускаются

Для создания простого индекса необходимо выполнить следующие действия: 1) В окне конструктора таблицы выберите в верхней половине окна поле, для которого создается индекс. 2) В нижней половине окна для свойства **Индексированное поле** выберите одно из следующих значений: **Да (Допускаются совпадения)** или **Да (Совпадения не допускаются).**

Значение **Да (Совпадения не допускаются)** обеспечивает уникальность каждого значения данного поля.

Не допускается создание индексов для полей MEMO и полей объектов OLE.

Создание составного индекса

Индексы, содержащие несколько полей, следует определять в окне индексов.

1. В окне конструктора откройте таблицу, для которой вы создаете составной индекс. Для этого в окне базы данных установите указатель на данную таблицу и нажмите кнопку Конструктор.
2. Нажмите кнопку Индексы на панели инструментов. На экране откроется окно диалога «Индексы»
3, В открывшемся окне диалога введите имя индекса в поле столбца Индекс в первой пустой строке. В качестве имени индекса можно использовать имя одного из полей, включенных в индекс, или любое допустимое имя.
4. В столбце Имя поля той же строки нажмите кнопку раскрытия списка и выберите первое поле индекса.
5. В столбце Имя поля следующей строки выберите имя следующего поля индекса. (В этой строке поле столбца Индекс следует оставить пустым). Определите таким же образом остальные поля индекса. Индекс может включать до 10 полей.
6. Закончив выбор полей для индекса, нажмите кнопку закрытия окна, расположенную в строке заголовка окна диалога.

По умолчанию задается порядок сортировки По возрастанию. Для сортировки конкретного поля по убыванию выберите в столбце **Порядок сортировки** строки с выбранным полем значение **По убыванию.**

Установка первичного ключа

Когда напоминает вам об отсутствии первичного ключа и предлагает его создать, нажмите Да. добавит поле код в первой строке описания структуры таблицы.

Если вы хотите установить первичный ключ самостоятельно, нажмите на поле, которое будет в дальнейшем использоваться в качестве первичного ключа. Затем нажмите правую кнопку мыши и выберите пункт всплывающего **Первичный** ключ. В области выбора поля, которое будет использоваться как ключ, появится маленькая пиктограмма с изображением первичного ключа. Для установки первичного ключа также можно использовать кнопку Ключевое поле на панели инструментов.

3. ОБЪЕКТЫ И СРЕДСТВА ИССЛЕДОВАНИЯ

Предметная область в соответствии с вариантом задания. ПК типа IBM PC, дисплейный класс, операционная система Windows XP и выше, СУБД OO Base.

4. ПОДГОТОВКА К РАБОТЕ

Ознакомиться с теоретическими положениями лабораторной работы.

5. ПРОГРАММА РАБОТЫ

По произвольно выбранным полям осуществить сортировку и поиск записей. Оформить отчет.

ВАРИАНТЫ ЗАДАНИЙ . Выбирается преподавателем из списка заданий (См. Приложение) и остается единым для всех лабораторных работ.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

6.1. Назовите основные элементы таблиц БД?

6.2. Что такое первичный ключ и индекс?

6.3. Как осуществляется поиск информации в БД?

Лабораторная работа № 4
СПОСОБЫ ОБЪЕДИНЕНИЯ ТАБЛИЦ

1. ЦЕЛЬ РАБОТЫ

Изучение средств автоматизации проектирования схем БД.

2. ОСНОВЫ ТЕОРИИ

Определение отношений между таблицами

Вы можете устанавливать постоянные отношения между таблицами, которые будут поддерживаться при создании форм, отчетов и запросов.

Устанавливая связи между двумя таблицами, вы выбираете поле, которое содержит одну и ту же информацию. Чаще всего вы будете связывать первичный ключ одной таблицы с совпадающими полями другой таблицы.

Поля, с помощью которых устанавливается связь между двумя таблицами, могут иметь различные имена, но удобнее использовать совпадающие имена.

Из существующих типов связей тип «один-ко-многим» наиболее важен, поэтому самое пристальное внимание следует уделить ему. В отношении «один-ко-многим» *главной таблицей* является таблица, которая содержит первичный ключ и составляет часть «один» в отношении «один-ко-многим». *Внешний ключ* — это поле (или поля), содержащее такой же тип информации в таблице со стороны «много» в отношении «один-ко-многим», которую называют *подчиненной таблицей.*

Окно диалога «Схема данных»

Создание связей между таблицами в осуществляется в окне диалога «Схема данных». Для определения связей между таблицами необходимо выполнить следующие действия:

1. Откройте окно диалога «Схема данных», выполнив команду **Сервис | Схема данных** или нажав кнопку **Схема данных** на панели инструментов. На экране откроется окно диалога «Схема данных».

Перед определением связей между таблицами необходимо предварительно закрыть все открытые таблицы. Не допускается создание или удаление связей между открытыми таблицами.

2. Добавьте в это окно диалога последовательно две связываемые таблицы. Для этого выполните команду **Связи | Показать таблицу** или нажмите кнопку **Добавить таблицу** на панели инструментов. На экране откроется окно диалога «Добавление таблицы»

3. В списке таблиц выделите первую добавляемую таблицу и нажмите кнопку **Добавить.** Затем выберите вторую добавляемую таблицу и также нажмите кнопку **Добавить.** Затем нажмите кнопку **Закрыть** для закрытия

окна диалога «Добавление таблицы». В окне диалога «Схема данных» появились две связываемые таблицы.

4. Для связывания таблиц выберите поле в первой связываемой таблице и переместите его с помощью мыши на соответствующее поле второй таблицы. Для связывания сразу нескольких полей выберите эти поля при нажатой клавише *Ctrl* и переместите во вторую таблицу группу выделенных полей.

В большинстве случаев связывают ключевое поле (представленное в списке полей полужирным шрифтом) одной таблицы с соответствующим ему полем внешнего ключа (часто имеющим то же имя) во второй таблице. Связанные поля не обязательно должны иметь одинаковые имена, однако, они должны иметь одинаковые типы данных (из этого правила существует два исключения) и иметь содержимое одного типа. Кроме того, связываемые поля типа **Числовой** должны иметь одинаковые значения свойства Размер поля. Исключениями из этого правила являются поля счетчика с последовательной нумерацией, которые могут связываться с числовыми полями размера **Длинное целое**, а также поля счетчика с размером **Код репликации**, связываемые с полями типа **Числовой**, для которых также задан размер **Код репликации**.

5. На экране откроется окно диалога «Связи» . В данном окне диалога проверьте правильность имен связываемых полей, находящихся в столбцах. При необходимости выберите другие имена полей. Затем нажмите кнопку **Создать.** Вы вернетесь в окно диалога «Схема данных».

Тип создаваемой связи зависит от полей, которые были указаны при определении связи;

- Отношение «один-ко-многим» — создается в том случае, когда только одно из полей является ключевым или имеет уникальный индекс.

- Отношение «один- к-одному» — создается в том случае, когда оба связываемых поля являются ключевыми или имеют уникальные индексы.

- Связь с отношением «многие-ко-многим» — фактически представляет две связи с отношением «один-ко-многим» через третью таблицу, ключ которой состоит, по крайней мере, из двух полей, которые являются полями внешнего ключа в двух других таблицах.

В окне диалога «Схема данных» при переносе поля, не являющегося ключевым или не имеющего уникального индекса, на другое поле, которое также не является ключевым или не имеет уникального индекса, создается неопределенное отношение. В запросах, содержащих таблицы с неопределенным отношением, по умолчанию создает линию объединения между таблицами, но условия целостности данных при этом не поддерживаются и нет гарантии уникальности записей в любой из таблиц.

В окне диалога «Схема данных» можно также выполнять следующие действия:

1) Изменить структуру таблицы. 2) Изменить существующую связь. 3) Удалить связь. 4) Удалить таблицу из окна диалога «Схема данных». 5) Вывести на экран все существующие связи или связи только для конкретной

таблицы. 6) Определить связи для запросов, не задавая условия целостности данных.

Связывание двух полей одной таблицы

Иногда возникает необходимость в определении поля с подстановкой значений из той же таблицы. Для связывания поля таблицы с другим полем той же таблицы дважды добавьте эту таблицу в окно диалога «Схема данных» и создайте требуемую связь, соединив поля линией связи.

Создание между таблицами отношения «многие-ко-многим»

Рассмотрим создание между таблицами отношения «многие-ко-многим». В отношение «многие-ко-многим» представляет две связи с отношением «один-ко-многим» через третью таблицу, ключ которой состоит, по крайней мере, из двух полей, являющихся полями внешнего ключа в двух других таблицах.

Рассмотрим создание такой связи:

1) Создайте таблицы, между которыми требуется определить связь с отношением «многие-ко-многим». 2) Создайте третью (связующую) таблицу с полями, описание которых совпадает с описанием ключевых полей в каждой из двух связываемых таблиц. В этой таблице ключевые поля выполняют роль внешнего ключа. Другие поля в связующую таблицу можно добавлять без ограничений. 3) Определите в связующей таблице ключ, содержащий все ключевые поля двух связываемых таблиц. 4) Определите связи с отношением «один-ко-многим» между каждой из двух таблиц и связующей таблицей.

Изменение структуры таблицы в окне диалога «Схема данных»

При создании связи в окне диалога «Схема данных» может возникнуть необходимость в изменении структуры таблицы. При этом вы можете не покидать окна диалога, а внести нужные изменения в структуру таблицы непосредственно в окне диалога:

1. Находясь в окне диалога «Схема данных», установите указатель мыши на модифицируемую таблицу.
2. Нажмите правую кнопку мыши и выберите из контекстного меню команду **Конструктор таблиц.**
3. Внесите в структуру таблицы необходимые изменения.
4. Закончив внесение изменений, нажмите кнопку закрытия окна в строке заголовка окна диалога. В ответ на запрос о сохранении изменений выберите **Да** для сохранения изменений и возвращения в окно диалога «Схема данных».

Изменение существующей связи

Прежде чем приступить к изменению связей между таблицами, закройте все открытые таблицы. не допускает изменение связей между открытыми таблицами. Затем выполните следующую последовательность действий:

1. Находясь в окне базы данных, нажмите кнопку **Схема данных** на панели инструментов,

2. Если таблицы, связи между которыми требуется изменить, не отображаются в окне диалога «Схема данных», нажмите кнопку **Добавить таблицу** на панели инструментов, установите указатель на имя нужной таблицы и дважды нажмите кнопку мыши. После этого нажмите кнопку **Закрыть,**

3. Установите указатель на линию связи, которую требуется изменить, и дважды нажмите кнопку мыши.

4**.** В открывшемся окне диалога «Связи» внесите нужные изменения и нажмите кнопку **ОК.**

Удаление связи

Нажмите кнопку **Схема данных** на панели инструментов, установите указатель на линию связи, которую требуется удалить, и выделите ее, нажав кнопку мыши. Нажмите клавишу *Delete***.** Когда предложит вам подтвердить удаление связи, нажмите кнопку **Да.**

Удаление таблицы из макета схемы данных

1. Откройте окно диалога «Схема данных». 2. Выберите таблицу, которую требуется удалить из данного окна, и нажмите клавишу *Delete***.** Таблица будет удалена из макета схемы данных вместе с определенными для нее связями. Данная операция изменяет только макет в окне диалога «Схема данных». И таблица, и ее связи будут по-прежнему сохраняться в базе данных.

Определение условий целостности данных

Целостность данных является одним из самых важных требований, предъявляемых к базам данных. Для задания условий целостности данных служат установленные между таблицами отношения. Условиями целостности данных называют набор правил, используемых в MS для поддержания связей между записями в связанных таблицах. Эти правила делают невозможным случайное удаление или изменение связанных данных Условия целостности данных выполняются при следующих условиях:

1) Связанное поле главной таблицы является ключевым полем или имеет уникальный индекс. 2) Связанные поля имеют один тип данных. 3) Обе таблицы принадлежат одной базе данных .

Если таблицы являются присоединенными таблицами, то они должны быть таблицами MS . Невозможно определить условия целостности данных для присоединенных таблиц из баз данных других форматов.

При определении условия целостности данных действуют следующие ограничения:

- Невозможно ввести в поле внешнего ключа связанной таблицы значение, не содержащееся в ключевом поле главной таблицы. Однако возможен ввод в поле внешнего ключа пустых значений, показывающих, что записи не являются связанными.

- Не допускается удаление записи из главной таблицы, если существуют связанные с ней записи в подчиненной таблице.

- Невозможно изменить значение ключевого поля в главной таблице, если имеются записи, связанные с этой записью. Например, невозможно удалить код сотрудника в таблице Сотрудники, если в таблице Заказы имеются заказы, относящиеся к данному сотруднику.

Определение целостности данных предполагает выполнение следующих действий:

1) В окне диалога «Схема данных» два раза мышью на линии связи между двумя таблицами. Откроется окно диалога «Связь». 2) Установите флажок **Обеспечение целостности данных** и нажмите **ОК.**

Для того чтобы преодолеть ограничения на удаление или изменение связанных записей, сохраняя при этом целостность данных, следует включить режимы каскадного обновления и каскадного удаления. При установленном флажке **Каскадное обновление связанных полей** изменение значения в ключевом поле главной таблицы приводит к автоматическому обновлению соответствующих значений во всех связанных записях. При установленном флажке **Каскадное удаление связанных записей** удаление записи в главной таблице приводит к автоматическому удалению связанных записей в подчиненной таблице.

Использование каскадных операций

Обратимся к окну диалога «Связи». При установке опции **Обеспечение целостности данных** вам стали доступны опции **Каскадное обновление связанных полей** и **Каскадное удаление связанных нолей**. При выборе этих опций, выполняет изменения в связанных таблицах таким образом, чтобы сохранить целостность данных, даже если вы изменяете значения ключевых полей или удаляете запись в главной таблице.

Каскадные изменения

Может возникнуть необходимость внести изменения в ключевое поле записи, связанной по типу «один-ко-многим». Если вы не установили опцию **Каскадное обновление связанных полей, то** не позволит этого сделать. Вместо изменения выдаст вам предупреждение о том, что вы нарушите целостность данных, если попытаетесь осуществить это изменение. Это является нарушением потому что это поле является первичным ключом в связи «один-ко-многим», и изменяемое значение появляется, по крайней мере, в одной записи в поле связанной с ним таблицы. Если же вы установили **Каскадное обновление связанных полей,** проблем не возникнет. Вы можете ввести изменения. выполнит все необходимые изменения в связанных таблицах автоматически.

Каскадные удаления

При попытке удалить запись в ключевом поле записи, связанной по типу «один-ко-многим» при не установленной опции **Каскадное удаление связанных полей,** выдаст сообщение об ошибке.

Однако, если выбрана опция **Каскадное удаление связанных полей,** вы можете удалить запись из главной таблицы. Связанные с нею записи в подчиненных таблицах будут также автоматически удалены, соблюдая ,таким образом, правила целостности данных. Другими словами, если выбрана опция **Каскадное удаление связанных полей** то все связанные поля из подчиненной таблицы удаляются, как только удаляется запись из главной таблицы.

3. ОБЪЕКТЫ И СРЕДСТВА ИССЛЕДОВАНИЯ

Предметная область в соответствии с вариантом задания. ПК типа IBM PC, дисплейный класс, операционная система Windows XP и выше, СУБД OO Base.

4. ПОДГОТОВКА К РАБОТЕ

Ознакомиться с теоретическими положениями лабораторной работы.

5. ПРОГРАММА РАБОТЫ

Установите отношения между таблицами. Определите условия целостности данных. Оформить отчет.

ВАРИАНТЫ ЗАДАНИЙ . Выбирается преподавателем из списка заданий (См. Приложение) и остается единым для всех лабораторных работ.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

6.1 Как формируются отношения между реляционными таблицами?

6.2. Опишите технологию создания схемы данных?

6.5. Что такое каскадное изменение и удаление ?

Лабораторная работа № 5

СОЗДАНИЕ ФОРМ

1. ЦЕЛЬ РАБОТЫ

Изучение средств автоматизации конструирования форм просмотра и редактирования данных в СУБД.

2. ОСНОВЫ ТЕОРИИ

Форма позволяет объединить поля в группы по определенным признакам. Это облегчает восприятие информации.

Простейший способ создания формы

Для создания формы любым способом на начальном этапе необходимо выполнить следующие действия:

1. Открыть окно БД. 2. Перейти на вкладку Формы. 3. Нажать кнопку **Создать** . Выбрать нужный вариант. Ок.

Можно воспользоваться мастером форм и проделать следующие операции:

1. Открыть окно БД. 2. В окне перейти на вкладку «Таблицы». 3. Указать на таблицу, для которой создается экранная форма. 4. Выполнить команду **Вставка|Автоформа** или нажать кнопку **Новый объект** и из списка выбрать опцию **Автоформа.** На экране появится готовая форма.

Создание формы в конструкторе форм

В данном разделе обсудим создание формы с помощью конструктора форм. Процесс по созданию формы может включать в себя все или часть из приведенных ниже процедур:

- Размещение текста.
- Размещение полей.
- Создание управляющих кнопок.
- Размещение линий, прямоугольников и рисунков.
- Установка цвета объектов формы.
- Перемещение объектов формы.

Наиболее частым источником ошибок в базе данных является ввод пользователем неправильных данных. Тщательно разработав форму, которую пользователи будут использовать для ввода, редактирования или просмотра данных, можно предотвратить возникновение большого количества ошибок. При создании форм учитывайте следующее

- Если пользователи привыкли к использованию стандартных бланков, формы должны выглядеть так же, как эти бланки. Необходимость каждый

раз искать местонахождение информации никогда не способствовала безошибочному вводу данных.

- Для группировки элементов управления используйте линии и прямоугольники. При этом пользователь будет вводить близкие по смыслу данные (такие, как вся идентификационная информация по товару или полный домашний адрес) вместе.

- Не концентрируйте элементы управления в какой-либо части формы. Это затрудняет чтение информации. Пользователь должен наглядно видеть, с каким элементом управления он работает в данное время.

- Пояснительный текст формы должен быть максимально информативным и иметь минимальную длину.

- Используйте условия правильности ввода данных что поможет предотвратить ввод неверных данных.

- Используйте маски ввода для ввода стандартизированной информации. Например, если вам известно точное количество символов, необходимых для ввода номера телефона, используйте маску ввода.

- Для облегчения восприятия чисел используйте форматы ввода.

Настройка формы

Для создания формы войдите в окно конструктора форм. Для этого, находясь на вкладке «Формы» БД нажмите кнопку **Создать** и в открывшемся окне диалога «Новая форма» нажмите кнопку **ОК.** На экране откроется окно конструктора форм.

Первое, что нужно сделать — определить свойства самой формы как объекта. Каждая форма имеет свойства, определяющие расположение ее в основном окне , размер, заголовок, стиль и некоторые другие параметры.

1. Для определения или изменения стиля формы, находясь в конструкторе форм, выполните команду **Формат|Автоформат** или нажмите кнопку **Автоформат** на панели инструментов На экране откроется окно диалога «Автоформат» . Выберите из списка стили и посмотрите на их внешний вид в окне просмотра После того как вы нашли нужный стиль, нажмите кнопку ОК.

2. Для задания размеров формы используйте мышь. Установите указатель мыши в нижний правый угол формы. При этом курсор примет вид двунаправленной стрелки. Нажмите кнопку мыши и установите требуемый размер формы.

3. Для настройки остальных параметров формы откройте окно ее свойств, выполнив команду **Вид | Свойства**. На экране появится окно свойств со стандартными значениями свойств формы .

4. Перейдите на вкладку «Макет» окна свойств. На этой вкладке, редактируя свойства формы, вы можете задать различные параметры окна формы, например:

• Наличие кнопок оконного меню в верхнем правом углу с помощью свойства **Кнопка оконного меню**

• Наличие кнопок свертывания и развертывания окна с помощью свойства **Кнопки размеров** окна

• Отображение кнопки закрытия формы с помощью свойства **Кнопка закрытия**

• Расположение формы в центре окна приложения установкой значения свойства **Выравнивание по** центру равным **Да**

• Заголовок окна вводом текста заголовка в поле свойства Подпись

5. Для связывания формы с таблицей или запросом используется свойство **Источник записей** вкладки «Данные». Это свойство определяет в качестве источника данных для формы таблицу или запрос. Нажмите кнопку раскрытия списка и из списка таблиц и запросов выберите тот источник записей, для которого вы хотите создать форму.

6. Для использования формы только для просмотра, установите для свойства **Разрешить изменение** значение **Нет**. Данная установка не позволит пользователю редактировать содержимое элементов управления, связанных с таблицей или запросом.

7. Если установить для свойства **Разрешить добавление** и **Разрешить добавление** значения **Нет,** то становятся недоступными команды **Вставка|Запись, Записи|Ввод данных и Правка|Удалить запись.**

Размещение текстовой информации

Размещение текста в экранной форме осуществляется с помощью инструмента **Надпись,** который находится на панели элементов. Под текстом понимается любая текстовая информация: заголовки, поясняющая информация. Для размещения текста в форме выполните следующие действия:

1. Выберите инструмент **Надпись** на панели элементов. Если данная панель отсутствует на экране, для ее отображения выполните команду **Вид |Панели инструментов** и в списке панелей инструментов установите опцию **Панель элементов** или нажмите кнопку **Панель элементов** на панели инструментов.

2. Установите указатель мыши на место предполагаемого расположения текстового объекта и введите текст.

3. Закончив ввод текста, нажмите клавишу *Enter*.

4. Выделите созданный объект.

5. Используя панель инструментов «Форматирование» или окно свойств созданного объекта, задайте для него тип шрифта, размер, цвет шрифта, цвет рамки, тип и цвет фона и другие параметры оформления. Все свойства вы можете определить в окне свойств. Некоторые, наиболее часто используемые, можно задавать и с помощью панели «Форматирование»

Размещение полей ввода

Следующим шагом в создании формы является добавление в нее полей различных типов. Наиболее простым типом поля является поле ввода. Для размещения поля ввода в форме выполните следующие действия:

1. Выберите инструмент **Поле** на панели элементов.

2. Нажмите мышью место, в котором вы предполагаете разместить поле. В форме появится связанный объект, состоящий из поля ввода и его надписи. Выделите поле ввода и откройте для него окно свойств.

3. Чтобы связать созданное поле с полем таблицы или запроса выберите свойство **Данные** вкладки «Данные». В поле ввода свойства воспользуйтесь кнопкой раскрытия списка и выберите из списка всех полей открытой таблицы поле, которое хотите добавить в форму. Если вы хотите связать поле с выражением, нажмите кнопку **Построить.** Создайте необходимую формулу с помощью построителя выражений.

4. Используя панель инструментов форматирования или окно свойств поля ввода, задайте для него тип шрифта, размер, цвет шрифта, цвет рамки, тип, цвет фона и другие параметры.

5. Если вы создаете поле, информация из которого должна быть доступна только для чтения, необходимо установить значение свойства **Доступ** равным **Нет.**

6. Свойство **Всплывающая подсказка** вкладки «Другие» позволяет создать краткое пояснение к полю, которое будет появляться на экране, когда указатель установлен на поле и удерживается на нем некоторое время.

7. Для определения значения поля по умолчанию задайте свойство **Значение по умолчанию.**

8. Выделите надпись к полю ввода и откройте для него окно свойств.

9. Чтобы задать текст надписи, выберите свойство **Подпись** вкладки «Макет» и в поле ввода свойства введите текст надписи к полю.

10. Используя панель инструментов «Форматирование» или окно свойств, задайте для надписи тип шрифта, размер, цвет шрифта, цвет рамки, тип, цвет фона и другие параметры.

Скрытие поля

Используя окно свойств поля, вы можете сделать поле невидимым (и поле ввода и надпись к нему), так что его не будет видно в режиме просмотра формы. Аналогичная возможность существует в режиме просмотра таблиц, в котором для скрытия столбца таблицы используется команда **Формат | Скрыть столбцы.** В форме для скрытия поля и надписи к нему необходимо для свойства **Вывод на экран** вкладки «Макет» поля задать значение **Нет.**

Отображение сообщений в строке состояния

При создании таблиц упоминалось, что информация, помещенная в столбец «Описание», в режиме просмотра формы выводится в строке состояний, когда курсор находится в данном поле. Если этот столбец пуст, то сообщение не выводится.

Вы можете изменить сообщение в форме с помощью свойства **Текст строки состояния** вкладки «Другие». Для этого откройте окно свойств поля

и введите в поле ввода данного свойства текст, который будет появляться в строке состояния.

Изменение формата отображения дат и чисел

Для задания формата отображения дат и/или чисел в форме используется свойство **Формат поля** вкладки «Макет».

Размещение списка и раскрывающегося списка

При вводе информации в таблицу во многих случаях удобнее выбирать повторяющееся значение из списка, чем каждый раз вводить одно из двух-трех значений. Кроме того, выбор из списка позволяет быть уверенным, что введенное значение является допустимым.

Элементы панели элементов **Список** и **Поле со списком** предназначены для отображения на экране элементов списка. Выбор одного из этих двух типов объектов определяется требованиями пользователя и наличием свободного места в форме.

При небольшом числе элементов списка удобнее использовать **Список,** который всегда раскрыт на экране. Когда список большой, используйте элемент **Поле со списком**. Он занимает меньше места на экране, поскольку список раскрывается на экране только при нажатии кнопки раскрытия списка.

Список и **Поле со списком** состоят из строк данных. Строки содержат один или несколько столбцов, определяемых с помощью следующих средств:

- Список фиксированных значений.
- Список полей.
- Значения поля таблицы или запроса.

Тип источника данных определяется свойством **Тип источника** строк вкладки «Данные»,

Объекты типа списка и поля со списком имеют дополнительные свойства, которые отсутствовали у ранее рассмотренных объектов Надпись и Поле.

Для размещения списка, поля со списком, группы параметров и ряда других элементов управления вы можете использовать мастера. Для этого вам необходимо установить режим использования мастера и выбрать соответствующий инструмент на панели элементов. После указания места расположения элемента запускается соответствующий мастер.

Размещение флажка

Для индикации состояния, которое может иметь только одно из двух допустимых значений, используются *флажки.* Они могут использоваться по одному или группами. Установленный флажок будет соответствовать значению «постоянный», а снятый — значению «временный».

Рассмотрим последовательность ваших действий при создании флажка для редактирования поля , которое имеет тип **Логический.**

1. Откройте форму в режиме конструктора и выберите инструмент **Флажок** на панели элементов.
2. Нажмите мышью место предполагаемого размещения элемента в форме.
3. Откройте окно свойств размещенного в форме флажка.
4. Чтобы связать созданное поле с полем таблицы, выберите свойство **Данные** вкладки «Данные». В поле ввода значения свойства воспользуйтесь кнопкой раскрытия списка и выберите поле из списка полей таблицы.
5. Выделите надпись созданного флажка и в его окне свойств скорректируйте свойство **Подпись** вкладки «Макет». Просмотрите форму в режиме формы.

Для создания объектов логического типа используются элементы **Переключатель** и **Выключатель** панели элементов, соответственно.

Создание кнопок управления

Кнопки используются в формах для выполнения определенного действия или ряда действий. Например, можно создать в форме кнопку, открывающую другую форму, или создать набор кнопок для перемещения по записям таблицы, если вас не устраивают стандартные средства перемещения, предусмотренные в форме. Для того чтобы кнопка выполняла какое-либо действие, необходимо создать макрос или процедуру обработки события и связать их со свойством кнопки **Нажатие кнопки.**

В предусмотрено создание более 30 разных кнопок, что избавляет пользователя от необходимости самостоятельно разрабатывать макросы. Достаточно лишь воспользоваться мастером по созданию кнопки. Так для создания кнопки, открывающей другую форму нужно выполнить следующие действия:

1. Установите режим использования мастера на панели элементов и выберите инструмент **Кнопка** на панели элементов.
2. Установите указатель мыши на место в форме, в котором вы предполагаете расположить кнопку и нажмите кнопку мыши. Запускается мастер создания **кнопки конструктора форм.**
3. В первом окне диалога расположены два списка: Категории и **Действия**. При перемещении по списку Категории список Действия обновляется. Рассмотрите внимательно содержимое этих списков. Список **Категории** содержит наборов действий, а список **Действия** — действия, которые будут выполняться при нажатии на данную кнопку. Выберите нужное значение из списка **Действия** и нажмите кнопку **Далее.**
4. Во втором окне диалога мастера из списка форм базы данных выберите форму, которая будет открываться при нажатии на кнопку.
5. На следующем шаге определяется тип отображаемой информации на кнопке: текстовая или графическая.

Если вы выбрали текстовую информацию, то в поле ввода рядом с опцией **Текст** введите текст, отображаемый на кнопке. При размещении графической информации установите опцию **Рисунок** и нажмите кнопку **Обзор** для открытия окна диалога «Выбор рисунка», в котором выберите графическое изображение. После чего переходите к следующему шагу.

6. На заключительном шаге работы мастера задается имя создаваемого объекта Введите имя, затем нажмите кнопку **Готово**. Кнопка, при нажатии на которую открывается другая экранная форма, создана.

Добавление рисунка или другого объекта в форму

Способ вставки рисунка или объекта зависит от того, какой объект предполагается создать: присоединенный или свободный. Присоединенный объект хранится в таблице. При переходе к новой записи в форме или отчете отображается другой объект. Например, этот способ удобен для хранения фотографий сотрудников фирмы. Свободный объект является частью структуры формы. При переходе к новой записи объект не изменяется. Например, таким способом в форме или отчете сохраняют эмблему фирмы, созданную в приложении MS Paint.

Размещение графического изображения

В экранные формы можно вставлять различные графические изображения, позволяющие облегчать восприятие информации. Для этого используется инструмент **Рисунок** панели элементов. При размещении данного элемента в форме открывается окно диалога «Выбор рисунка» . Используя список **Тип файла,** укажите тип используемого изображения. Выбрав имя вставляемого рисунка, нажмите кнопку **ОК.**

Размещение объекта типа OLE

Примером использования данного типа объектов является расположение в форме поля Фотография таблицы Сотрудники. В данном поле таблицы хранятся фотографии всех сотрудников фирмы.

Для присоединенного объекта в форме выполните следующие действия:

1. Для добавления графического поля типа OLE в форму выберите инструмент **Присоединенная рамка объекта** на панели элементов.

2. Нажмите мышью место, где вы хотите добавить поле. Удерживая кнопку мыши в нажатом состоянии, переместите указатель по диагонали так, чтобы получилась рамка требуемого размера.

3. Откройте окно свойств созданного объекта.

4. Чтобы связать созданное поле с полем таблицы, выберите свойство **Данные. В** поле ввода свойства воспользуйтесь кнопкой раскрытия списка и из списка полей открытой таблицы Сотрудники выберите поле типа OLE Фотография.

5. Просмотрите форму в режиме формы. Если рисунок не помещается в рамке целиком, вернитесь в режим конструктора и увеличьте размер поля.

Использование линий и прямоугольников

Линии и прямоугольники в экранной форме применяются дня улучшения внешнего вида формы и восприятия информации, а также для объединения объектов в логические группы.

Для добавления в экранную форму линий используется инструмент **Линия** панели элементов. Чтобы нарисовать вертикальную или горизонтальную линию на экране выберите данный инструмент, нажмите мышью то место, где должна начинаться линия, и переместите указатель до получения линии нужной длины. Настройка параметров линии осуществляется с помощью ее свойств.

Для добавления в экранную форму прямоугольников используется инструмент **Прямоугольник** панели элементов. Чтобы нарисовать прямоугольник выберите данный инструмент, нажмите мышью место расположения одного из углов прямоугольника, и переместите указатель до получения прямоугольника нужного размера.

3. ОБЪЕКТЫ И СРЕДСТВА ИССЛЕДОВАНИЯ

Предметная область в соответствии с вариантом задания. ПК типа IBM PC, дисплейный класс, операционная система Windows XP и выше, СУБД OO Base.

4. ПОДГОТОВКА К РАБОТЕ

Ознакомиться с теоретическими положениями лабораторной работы.

5. ПРОГРАММА РАБОТЫ

Разработайте формы для часто встречающихся групп данных с использованием конструктора форм. Добавьте для наглядности в форму подходящее графическое изображение. Если ваши данные отображаются в виде графика, то постройте его в форме. Оформить отчет.

ВАРИАНТЫ ЗАДАНИЙ . Выбирается преподавателем из списка заданий (См. Приложение) и остается единым для всех лабораторных работ.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

6.1. Каково назначение форм в базах данных ?

6.2. Опишите технологию создания простейших форм в СУБД ?

6.3. Назовите элементы управления формами и способы их реализации?

Лабораторная работа № 6

ФОРМИРОВАНИЕ ОТЧЕТОВ

1. ЦЕЛЬ РАБОТЫ

Изучение средств автоматизации конструирования отчетов в СУБД.

2. ОСНОВЫ ТЕОРИИ

Отчет (report) — это объект базы данных, который используется для вывода на экран, в печать или файл структурированной информации. Reports позволяют извлечь из таблиц или запросов базы данных необходимую информацию и представить ее в виде удобном для восприятия. Report содержит заголовок, область данных, верхний и нижний колонтитулы, примечание и разбит на страницы.

Для создания **отчетов** можно использовать различные средства (рис. 1):

- Мастер отчетов
- Конструктор отчетов
- Инструмент Report
- Пустой report

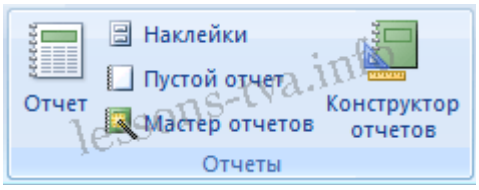


Рис. 1.

Отчеты целесообразно выполнять с помощью Мастера или других указанных инструментов, а дорабатывать их, т.е. вносить необходимые изменения можно в режиме макета или конструктора. В Microsoft 2007 предусмотрено два режима внесения изменений и дополнений в reports: режим макета и режим конструктора.

Режим макета — это более наглядный режим редактирования и форматирования (изменения) отчетов, чем режим конструктора. В тех случаях, когда в режиме макета невозможно выполнить изменения в отчете, то целесообразно применять режим конструктора.

Мастер отчетов. Для создания отчета при помощи Мастера отчетов необходимо выполнить следующие действия:

- В окне базы данных щелкнуть на вкладке Создание и затем щелкнуть на кнопке Мастер отчетов в группе Отчеты. Появится диалоговое окно Создание отчетов.
- В поле Таблицы и отчеты щелкнуть на стрелке и выбрать в качестве источника данных таблицу Студенты.
- Щелкнуть на кнопке ОК (в результате получим вид окна "Создание отчетов", представленный на рис. 2).

- Все "Доступные поля" переведем в "Выбранные поля", выделив их и щелкнув на кнопку >>.
- На следующем шаге (Добавить уровни группировки?) щелкаем далее.
- На шаге "Выберите порядок сортировки записей". В раскрывающемся списке выберем "Фамилия" для сортировки по возрастанию.
- На шаге "Выберите вид макета для отчета". Выбираем: Макет - блок, ориентация - книжная. Щелкнуть на кнопке Далее.
- На шаге " Выберите требуемый стиль". Выбираем - Изящная.
- Следующий шаг - "Задайте имя отчета". Вводим имя - Студенты мастер_отчетов. Дальнейшие действия: Просмотреть report; Изменить макет отчета. Выбираем Просмотреть, щелкаем на кнопке Готово. Report открывается в режиме Предварительного просмотра, который позволяет увидеть, как будет выглядеть report в распечатанном виде (Рис. 3).
- Перейдите в режим Конструктора и выполните редактирование и форматирование отчета. Для перехода из режима предварительного просмотра в режим конструктора необходимо в области переходов щелкнуть правой кнопкой мыши на имени отчета и в контекстном меню выбрать режим конструктора. На экране появится report в режиме Конструктора (Рис. 4).

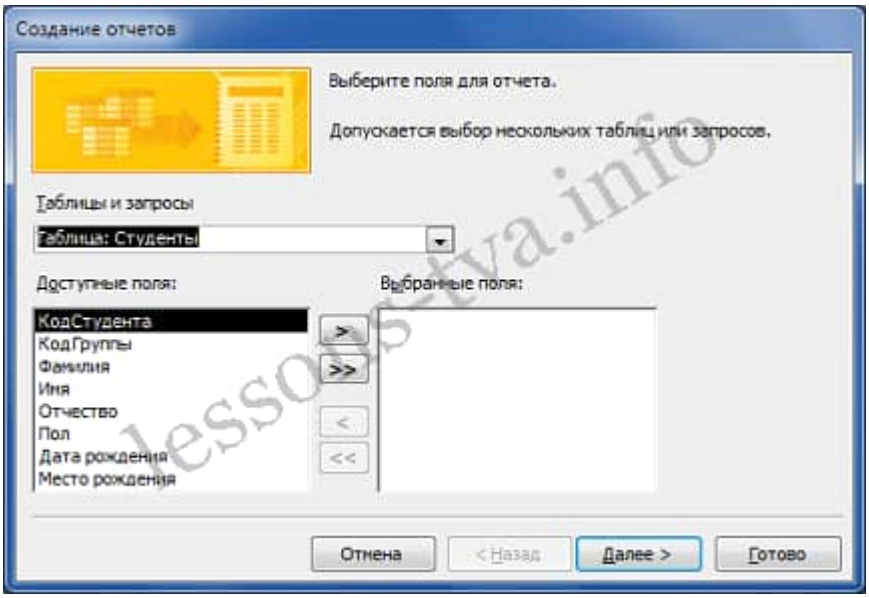


Рис. 2.

Инструмент Отчет. Для быстрого создания отчета, т.е. создания одним щелчком мыши можно воспользоваться инструментом Report. В этом случае report формируется на базе существующей таблицы или запроса. В созданном отчете будут отображаться все записи таблицы или запроса, на базе которых создается report. Но созданный report можно будет изменить в режиме макета или конструктора.

Для создания отчета необходимо выполнить следующее. В области переходов надо выделить таблицу (например, Студенты), на основе которой нужно создать report. Затем перейти на вкладку Создание и щелкнуть на пикто-

грамме Report. На экране будет отображен простой Отчет на основе текущей таблицы Студенты.

Средство Пустой отчет. Инструмент "Пустой report" позволяет создавать reports с нуля в режиме макета. Для этого надо щелкнуть Пустой report в группе Отчеты на вкладке Создание. В окне редактирования 2007 появится Отчет1 с пустой областью данных, а в правой части окна будет отображаться область "Список полей" существующих таблиц. Щелкнув на знак "+" таблицы (например, Студенты), откроется список необходимых полей.

Перетащите требуемые поля из этого списка в report, нажав и удерживая левую клавишу мыши. С помощью инструментов из группы "Элементы управления" на вкладке Формат, можно доработать report, добавив заголовок, номера страниц, дату и время. При необходимости его можно доработать в режиме конструктора. Сохраните report

3. ОБЪЕКТЫ И СРЕДСТВА ИССЛЕДОВАНИЯ

Предметная область в соответствии с вариантом задания. ПК типа IBM PC, дисплейный класс, операционная система Windows XP и выше, СУБД OO Base.

4. ПОДГОТОВКА К РАБОТЕ

Ознакомиться с теоретическими положениями лабораторной работы.

5. ПРОГРАММА РАБОТЫ

Разработайте отчеты для представления информации. Оформить отчет.

ВАРИАНТЫ ЗАДАНИЙ . Выбирается преподавателем из списка заданий (См. Приложение) и остается единым для всех лабораторных работ.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

6.1. Каково назначение отчетов в базах данных ?

6.2. Опишите технологию создания простейших отчетов в СУБД?

6.3. Назовите структуру отчетов и способы их реализации?

Лабораторная работа № 7

СОЗДАНИЕ МЕНЮ. ГЕНЕРАЦИЯ, ЗАПУСК

1. ЦЕЛЬ РАБОТЫ

Изучение средств автоматизации конструирования меню в СУБД.

2. ОСНОВЫ ТЕОРИИ

Главная кнопочная форма создается с целью навигации по базе данных. Эта форма может использоваться в качестве главного меню БД. Элементами главной кнопочной формы являются объекты форм и отчётов.

Запросы и таблицы не являются элементами главной кнопочной формы. Поэтому для создания кнопок Запросы или Таблицы на кнопочной форме можно использовать макросы. Сначала в окне базы данных создают макросы «Открыть Запрос» или «Открыть Таблицу» с уникальными именами, а затем в кнопочной форме создают кнопки для вызова этих макросов.

Для одной базы данных можно создать несколько кнопочных форм. Кнопки следует группировать на страницах кнопочной формы таким образом, чтобы пользователю было понятно, в каких кнопочных формах можно выполнять определенные команды (запросы, отчеты, ввода и редактирования данных). Необходимо отметить, что на подчиненных кнопочных формах должны быть помещены кнопки возврата в главную кнопочную форму.

Технология создания кнопочных форм следующая:

- создать страницу главной кнопочной формы (ГКФ);
- создать необходимое количество страниц подчиненных кнопочных форм (например, формы для ввода данных, для отчетов, для запросов и т.д.);
- создать элементы главной кнопочной формы;
- создать элементы для кнопочных форм отчетов и форм ввода или изменения данных;
- создать макросы для запросов или для таблиц с уникальными именами;
- создать элементы для кнопочных форм запросов или таблиц.

Структура кнопочных форм может быть представлена в следующем виде.

Рис. 1.

Для создания главной кнопочной формы и ее элементов необходимо открыть базу данных, (например, «Успеваемость_ студентов») и выполнить команду Сервис / Служебные программы / Диспетчер кнопочных форм. Если кнопоч-

ная форма ранее не создавалась, то откроется окно диалога «Диспетчер кнопочных форм».

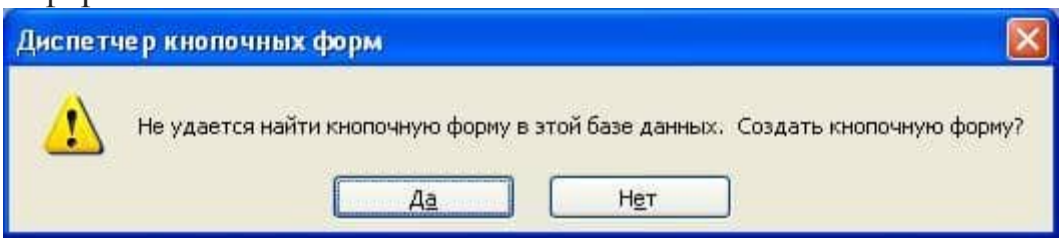


Рис. 2.

В окне диалога надо нажать кнопку «Да», тем самым подтвердить создание кнопочной формы. В результате будет создана страница Главной кнопочной формы.

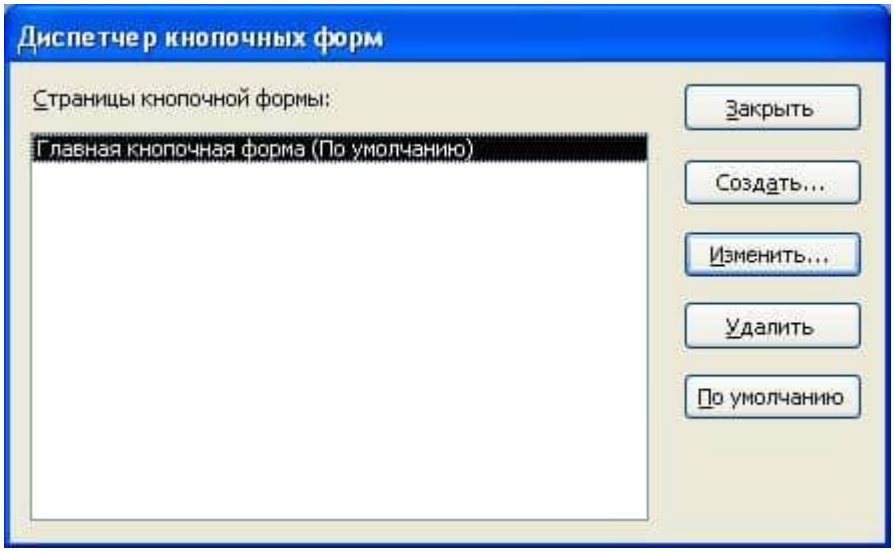


Рис. 3.

Далее можно создать еще три страницы кнопочной формы: Формы ввода данных, Отчеты и Запросы. Для этого следует щелкнуть на кнопке «Создать» и в появившемся окне ввести имя новой страницы «Формы ввода данных» и щелкнуть на кнопке «ОК».

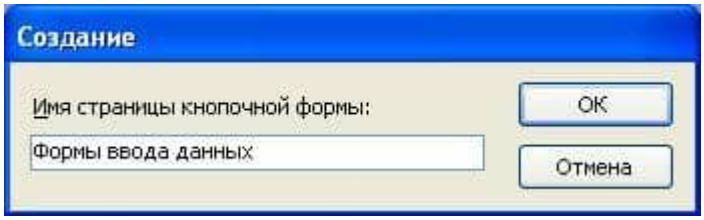


Рис. 4.

Будет создана страница кнопочной формы «Формы ввода данных». Аналогичным образом надо создать еще две страницы, в итоге получим четыре страницы кнопочных форм, которые отображаются в окне «Диспетчер кнопочных форм».

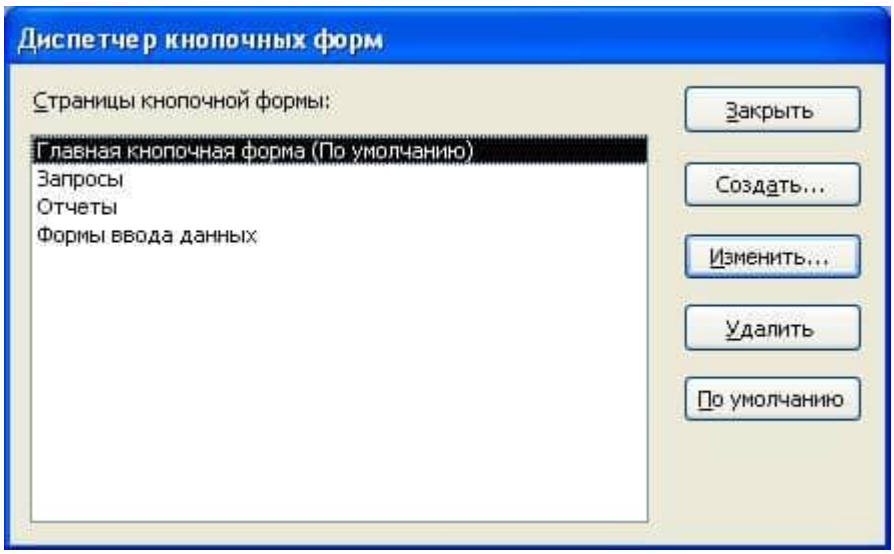


Рис. 5.

После этого создаем элементы ГКФ, для этого в «Окне диспетчер кнопочных форм» выделяем страницу «Главная кнопочная форма» и щелкаем «Изменить», откроется новое окно «Изменение страниц кнопочной формы».

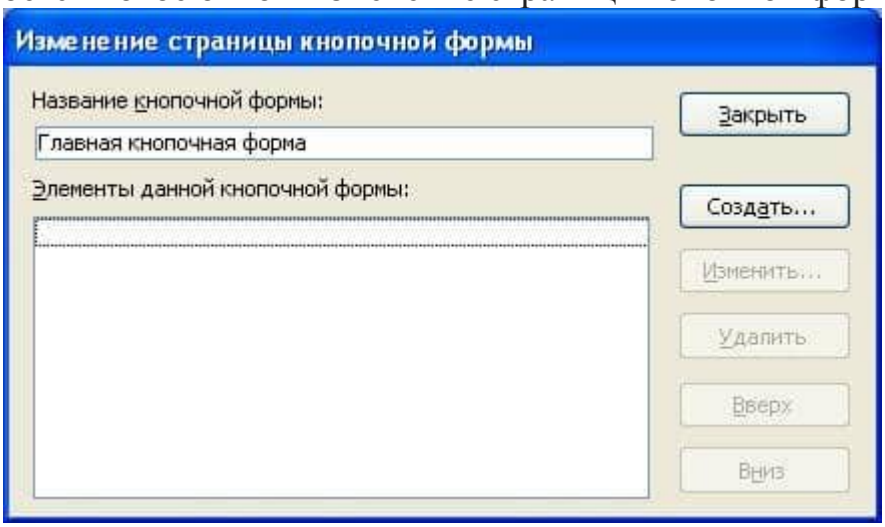


Рис. 6.

В этом окне щелкаем на кнопке «Создать», откроется новое окно «Изменение элемента кнопочной формы».

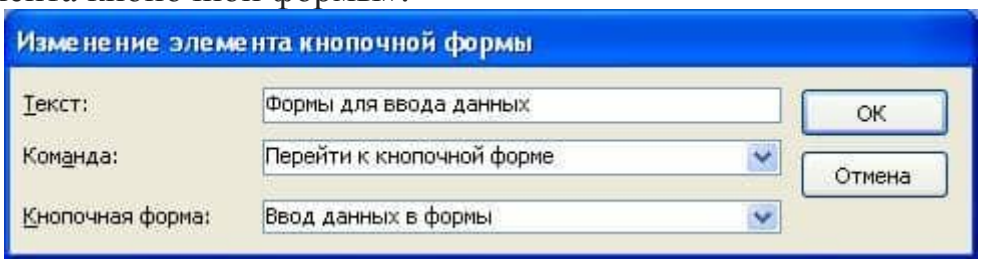


Рис. 7.

В окне выполняем следующее:

- вводим текст: Формы для ввода данных;
- выбираем из раскрывающегося списка команду: Перейти к кнопочной форме;
- выбираем из списка кнопочную форму: Ввод данных в формы, щелкаем на кнопке «ОК».

В окне «Изменение страницы кнопочной формы» отобразится элемент кнопочной формы «Формы для ввода данных».

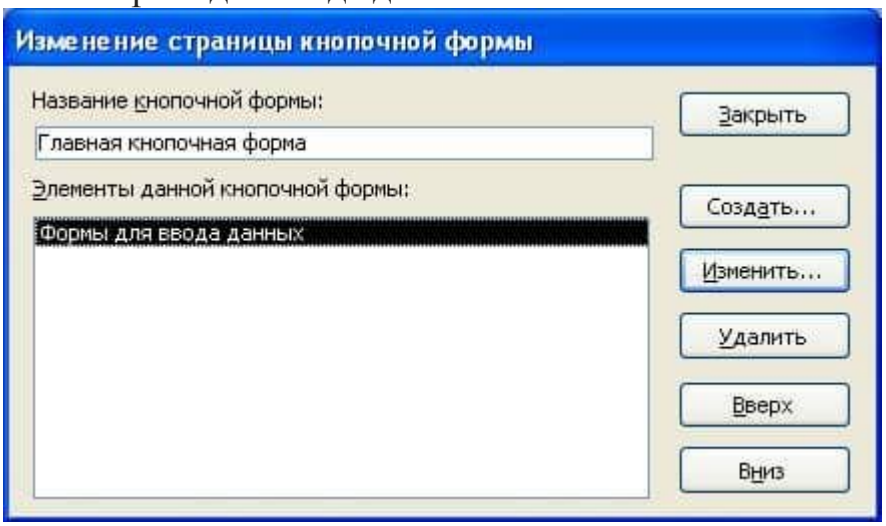


Рис. 8.

Аналогичным методом надо создать элементы: «Запросы» и «Отчеты», а затем элемент (кнопку) "Выход из БД".

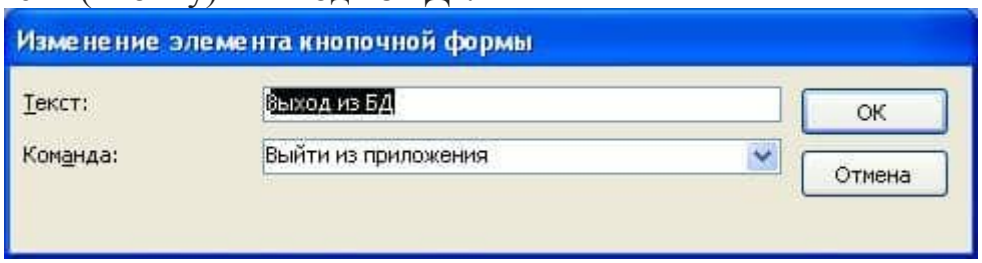


Рис. 9.

В результате в окне «Изменение страницы кнопочной формы» будут отображаться все элементы главной кнопочной формы.

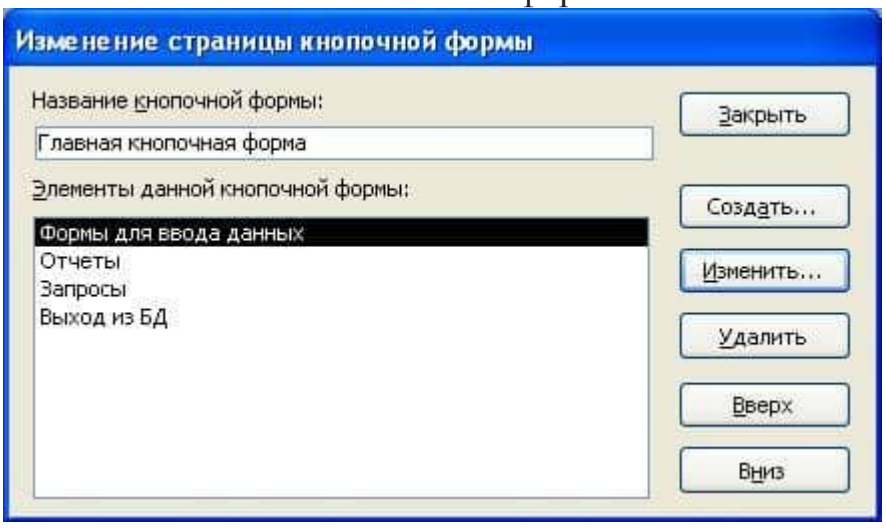


Рис. 10.

Кнопочная форма появится в списке в области окна базы данных на вкладке Формы на панели Объекты, а на вкладе Таблицы в списках появится таблица Switchboard Items. Дважды щелкнув на надписи "Кнопочная форма", откроется Главная кнопочная форма.

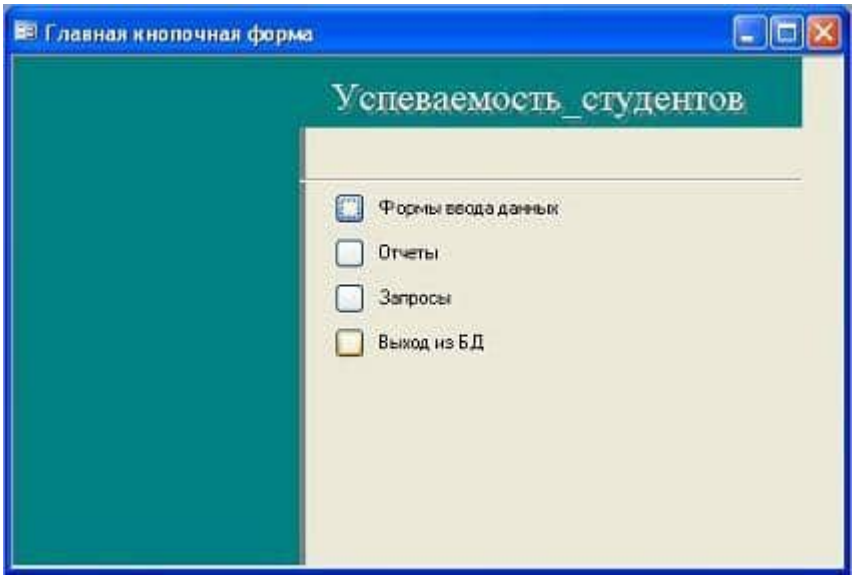


Рис. 11.

Для того чтобы эта форма отображалась при запуске базы данных, необходимо выполнить команду Сервис/Программы запуска, и в открывшемся окне выбрать "Кнопочная форма" из раскрывающегося списка, кроме того, надо снять флажки Окно базы данных и Строка состояния. Можно также ввести заголовок и значок приложения.

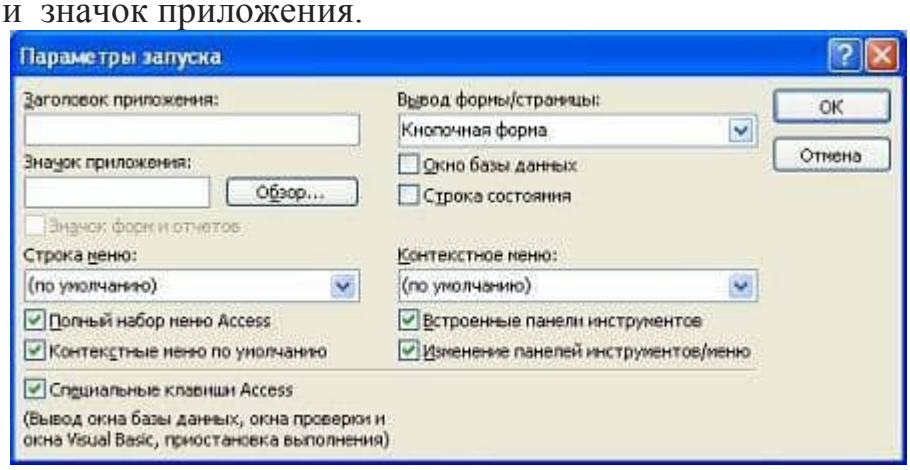


Рис. 12.

Но на этом создание кнопочных форм еще не законченно, так как на подчиненных кнопочных формах (Формы ввода данных, Отчеты, Запросы) нет элементов. Каким образом поместить элементы на подчиненные формы рассмотрим в следующем разделе.

3. ОБЪЕКТЫ И СРЕДСТВА ИССЛЕДОВАНИЯ

Предметная область в соответствии с вариантом задания. ПК типа IBM PC, дисплейный класс, операционная система Windows XP и выше, СУБД OO Base.

4. ПОДГОТОВКА К РАБОТЕ

Ознакомиться с теоретическими положениями лабораторной работы.

5. ПРОГРАММА РАБОТЫ

Разработайте кнопочную форму навигации по функциям БД. Оформить отчет.

ВАРИАНТЫ ЗАДАНИЙ . Выбирается преподавателем из списка заданий (См. Приложение) и остается единым для всех лабораторных работ.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

6.1. Каково назначение кнопочной формы в базах данных ?

6.2. Опишите технологию создания меню в СУБД?

6.3. Определите структуру меню и способы его реализации?

Лабораторная работа №8

СОЗДАНИЕ БАЗЫ ДАННЫХ С ПОМОЩЬЮ КОМАНД SQL. РЕДАКТИРОВАНИЕ, ВСТАВКА И УДАЛЕНИЕ ДАННЫХ СРЕДСТВАМИ ЯЗЫКА SQL

1. ЦЕЛЬ РАБОТЫ

Получить навыки использования SQL-запросов для решения различных задач.

2. ОСНОВЫ ТЕОРИИ

Запрос SQL — это запрос, создаваемый при помощи инструкций SQL (Инструкция (строка) SQL. Выражение, определяющее команду SQL, например **SELECT**, **UPDATE** или **DELETE**, и включающее предложения, например WHERE или ORDER BY. Инструкции/строки SQL обычно используются в запросах и в статистических функциях.). Язык SQL (Structured Query Language) используется при создании запросов, а также для обновления и управления реляционными базами данных, такими как базы данных Microsoft .

Когда пользователь создает запрос в режиме конструктора запроса, Microsoft автоматически создает эквивалентную инструкцию SQL. Фактически, для большинства свойств запроса, доступных в окне свойств в режиме конструктора, имеются эквивалентные предложения или параметры языка SQL, доступные в режиме SQL (Режим SQL. Окно, в котором выводится инструкция SQL текущего запроса или которое используется для создания запроса SQL (запроса на объединение, запроса к серверу или управляющего запроса). При необходимости, пользователь имеет возможность просматривать и редактировать инструкции SQL в режиме SQL. После внесения изменений в запрос в режиме SQL его вид в режиме конструктора может измениться.

Некоторые запросы SQL невозможно создать в бланке запроса. Для запросов к серверу (запрос SQL, используемый для передачи команд прямо на сервер базы данных ODBC. Запрос к серверу позволяет непосредственно работать с таблицами на сервере вместо обработки их данных с помощью ядра Microsoft Jet), управляющих запросов (запрос SQL, содержащий инструкции, которые позволяют создавать или изменять объекты в базе данных) и запросов на объединение (запрос, в котором оператор **UNION** используется для объединения результатов двух или нескольких запросов на выборку) необходимо создавать инструкции SQL непосредственно в окно запроса в режиме SQL. Для подчиненного запроса (инструкция SQL **SELECT**, расположенная внутри другого запроса на выборку или запроса на изменение) пользователь должен ввести инструкцию SQL в строку **Поле** или **Условие отбора** в бланке запроса.

1. Инструкция SELECT

По этой инструкции ядро базы данных Microsoft Jet возвращает данные из базы данных в виде набора записей.

SELECT [*предикат*] { * | *таблица*.* | [*таблица*.]*поле_1*
[AS *псевдоним_1*] [, [*таблица*.]*поле_2* [AS *псевдоним_2*] [, ...]]}
FROM *выражение* [, ...] [IN *внешняяБазаДанных*]
[WHERE...]
[GROUP BY...]
[HAVING...]
[ORDER BY...]
[WITH OWNER OPTION]

Ниже перечислены аргументы инструкции SELECT:

Элемент	**Описание**
предикат	Один из следующих предикатов отбора: ALL, DISTINCT, DISTINCTROW или TOP. Предикаты используются для ограничения числа возвращаемых записей. Если они отсутствуют, по умолчанию используется предикат ALL.
*	Указывает, что выбраны все поля заданной таблицы или таблиц.
таблица	Имя таблицы, из которой должны быть отобраны записи.
поле_1, *поле_2*	Имена полей, из которых должны быть отобраны данные. Если включить несколько полей, они будут извлекаться в указанном порядке.
псевдоним_1, *псевдоним_2*	Имена, которые станут заголовками столбцов вместо исходных названий столбцов в *таблице*.
выражение	Имена одной или нескольких таблиц, которые содержат отбираемые данные.
внешняяБазаДанных	Имя базы данных, которая содержит таблицы, указанные с помощью аргумента *выражение*, если они не находятся в текущей базе данных.

Элемент	**Описание**
ALL	Ядро базы данных Microsoft Jet отбирает все записи, соответствующие условиям, заданным в инструкции SQL.
DISTINCT	Исключает записи, которые содержат повторяющиеся значения в выбранных полях. Чтобы запись была включена в результат выполнения запроса, значения в каждом поле, включенном в инструкцию SELECT, должны быть уникальными. Например, в таблице «Сотрудники» есть однофамильцы. Если две записи содержат значение «Иванов» в поле «Фамилия», то следующая

инструкция SQL возвратит только одну из них:

```
SELECT DISTINCT
Фамилия
FROM Сотрудники;
```

Если опустить предикат DISTINCT, этот запрос возвратит обе записи для фамилии Иванов. Если предложение SELECT содержит более одного поля, то для включения записи в результат выполнения запроса необходимо, чтобы совокупность значений во всех этих полях была уникальной. Результат выполнения инструкции SQL, содержащей предикат DISTINCT, является необновляемым и не отражает последующие изменения, внесенные другими пользователями.

DISTINCTROW — Опускает данные, основанные на целиком повторяющихся записях, а не отдельных повторяющихся полях. Следующая инструкция SQL показывает, как можно использовать предикат DISTINCTROW для получения списка клиентов, разместивших хотя бы один заказ, без включения сведений о самих заказах:

```
SELECT DISTINCTROW Название
FROM Клиенты INNER JOIN Заказы
ON Клиенты.КодКлиента= Заказы.КодКлиента
ORDER BY Название;
```

Если опустить предикат DISTINCTROW, в результат выполнения запроса будет включено несколько строк о каждом клиенте, сделавшем несколько заказов. Предикат DISTINCTROW влияет на результат только в том случае, если в запрос включены не все поля из анализируемых таблиц. Предикат DISTINCTROW игнорируется, если запрос содержит только одну таблицу или все поля всех таблиц.

TOP *n* [PERCENT] — Возвращает определенное число записей, находящихся в начале или в конце диапазона, описанного с помощью предложения ORDER BY. Следующая инструкция SQL позволяет получить список 25 лучших студентов выпуска 1994 года:

```
SELECT TOP 25
Имя, Фамилия
FROM Студенты
WHERE ГодВыпуска = 1994
ORDER BY СреднийБалл DESC;
```

Если предложение ORDER BY будет опущено, запрос возвратит произвольный набор 25 записей из таблицы «Студенты», удовлетворяющих предложению WHERE. Предикат TOP не осуществляет выбор между равными значениями. Если в предыдущем примере средние баллы двадцать пятого и двадцать шестого студента будут равны, то запрос возвратит 26 записей.

Кроме того, можно использовать зарезервированное слово PERCENT для возврата определенного процента записей, находящихся в начале или в конце диапазона, описанного с помощью предложения ORDER BY. Предположим, что вместо 25 лучших студентов следует отобрать студентов, попавших в последние 10 процентов:
SELECT TOP 10 PERCENT
Имя, Фамилия
FROM Студенты
WHERE ГодВыпуска = 1994
ORDER BY СреднийБалл ASC;
Предикат ASC обеспечивает возврат последних значений. Значение, следующее после предиката TOP должно быть числовым значением типа **Integer** без знака. Предикат TOP не влияет на возможность обновления запроса.

Инструкции SELECT не изменяют данные в базе данных. Обычно слово SELECT является первым словом инструкции SQL. Большая часть инструкций SQL является инструкциями SELECT или SELECT...INTO.
Минимальный синтаксис инструкции SELECT *поля* FROM *таблица*
Для отбора всех полей таблицы можно использовать символ звездочки (*). Если несколько таблиц, включенных в предложение FROM, содержат одноименные поля, перед именем такого поля следует ввести имя таблицы и оператор **.** (точка).

Предложение FROM

… FROM *выражение* [IN *внешняяБазаДанных*]

Ниже перечислены аргументы инструкции SELECT, содержащей предложение FROM:

Элемент	**Описание**
выражение	Выражение, определяющее одну или несколько таблиц, откуда извлекаются данные. Это выражение может быть именем отдельной таблицы, именем сохраненного запроса или результатом операции INNER JOIN, LEFT JOIN или RIGHT JOIN.

Операция INNER JOIN объединяет записи из двух таблиц, если связующие поля этих таблиц содержат одинаковые значения.

…FROM *таблица_1* INNER JOIN *таблица_2* ON *таблица_1.поле_1* ***оператор*** *таблица_2.поле_2*

Оператор - Любой оператор сравнения: "=", "<", ">", "<=", ">=" или "<>".

Операция LEFT JOIN используется для создания левого внешнего объединения. Левое внешнее объединение включает все записи из первой (левой) таблицы, даже если нет совпадающих значений для записей из второй (правой) таблицы.
Операция RIGHT JOIN используется для создания правого внешнего объединения. Правое внешнее объединение включает все записи из второй (правой) таблицы, даже если нет совпадающих значений с записями из первой (левой) таблицы.

Предложение WHERE

Определяет, какие записи из таблиц, перечисленных в предложении FROM, следует включить в результат выполнения инструкции SELECT, UPDATE или DELETE. Ядро базы данных Microsoft Jet отбирает записи, соответствующие условиям, перечисленным в предложении WHERE. Предложение WHERE не является обязательным, однако, если оно присутствует, то должно следовать после предложения FROM. Например, можно отобрать всех сотрудников отдела продаж (WHERE Отдел = 'Продажи') или всех клиентов в возрасте от 18 до 30 лет (WHERE Возраст Between 18 And 30). Предложение WHERE может содержать до 40 выражений, связанных логическими операторами, такими как **And** и **Or**. При указании аргумента *условиеОтбора* литералы даты (символы дат) должны вводиться в американском формате, даже если используется неамериканская версия ядра базы данных Jet. Например, дата 10 мая 1996 года записывается в России как 10.05.96, а в США как 5/10/96.

Предложение GROUP BY

Объединяет записи с одинаковыми значениями в указанном списке полей в одну запись. Если инструкция SELECT содержит статистическую функцию SQL, например **Sum** или **Count**, то для каждой записи будет вычислено итоговое значение.

SELECT *списокПолей*
  FROM *таблица*
  WHERE *условиеОтбора*
  [GROUP BY *группируемыеПоля*]

Используйте предложение WHERE для исключения записей из группировки, а предложение HAVING для применения фильтра к записям после группировки. Если поле, включенное в предложение GROUP BY, не является полем типа Memo или объект OLE, оно может содержать ссылку на любое поле, перечисленное в предложении FROM, даже если это поле не включено в инструкцию SELECT, при условии, что инструкция SELECT содержит по крайней мере одну статистическую функцию SQL. Ядро базы данных Microsoft®

Jet не поддерживает группировку полей MEMO или объектов OLE. При использовании предложения GROUP BY все поля в списке полей инструкции SELECT должны быть либо включены в предложение GROUP BY, либо использоваться в качестве аргументов статистической функции SQL.

Предложение HAVING

Определяет, какие сгруппированные записи отображаются при использовании инструкции SELECT с предложением GROUP BY. После того как записи будут сгруппированы с помощью предложения GROUP BY, предложение HAVING отберет те из полученных записей, которые удовлетворяют условиям отбора, указанным в предложении HAVING.

SELECT *списокПолей*
  FROM *таблица*
  WHERE *условиеОтбора*
  [GROUP BY *группируемыеПоля*]
  [HAVING *условиеГруппировки*]

Предложение HAVING похоже на предложение WHERE, которое определяет, какие записи должны быть отобраны. После того как записи будут сгруппированы с помощью предложения GROUP BY, предложение HAVING указывает, какие из полученных записей должны быть отобраны. Предложение HAVING может содержать до 40 выражений, связанных логическими операторами, такими как **And** и **Or**

2. Инструкция SELECT...INTO

Создает запрос на создание таблицы.

SELECT *поле_1*[, *поле_2*[, ...]] INTO *новаяТаблица* [IN *внешняяБазаДанных*]
  FROM *источник*

Ниже перечислены аргументы инструкции SELECT...INTO:

Элемент	**Описание**
поле_1, *поле_2*	Имена полей, которые следует скопировать в новую таблицу.
новаяТаблица	Имя создаваемой таблицы. Это имя должно удовлетворять стандартным правилам именования. Если *новаяТаблица* совпадает с именем существующей таблицы, возникает перехватываемая ошибка.
внешняяБазаДанных	Путь к внешней базе данных. Более подробные сведения об этом аргументе можно найти в описании предложения IN.
источник	Имя существующей таблицы, из которой отбираются записи. Это может быть одна таблица, несколько таблиц или запрос.

Запрос на создание таблицы можно использовать для архивирования записей, создания резервных копий таблицы, копий для экспорта в другую базу данных, а также в качестве основы отчета, отображающего данные за конкретный период времени. Например, можно создать отчет «Ежемесячные продажи по областям», выполняя каждый месяц один и тот же запрос на создание таблицы.
Чтобы узнать, какие записи будут отобраны при выполнении запроса на создание таблицы, сначала просмотрите результаты инструкции SELECT, использующей те же условия отбора.

3. Инструкция INSERT INTO

Добавляет запись или записи в таблицу. Эта инструкция образует запрос на добавление записей.
Запрос на добавление нескольких записей:

INSERT INTO *назначение* [(*поле_1*[, *поле_2*[, ...]])] [IN *внешняяБазаДанных*]
  SELECT [*источник*.]*поле_1*[, *поле_2*[, ...]
  FROM *выражение*

Запрос на добавление одной записи:

INSERT INTO *назначение* [(*поле_1*[, *поле_2*[, ...]])]
  VALUES (*значение_1*[, *значение_2*[, ...])

Ниже перечислены аргументы инструкции INSERT INTO:

Элемент	**Описание**
назначение	Имя таблицы или запроса, в который добавляются записи.
поле_1, *поле_2*	Имена полей для добавления данных, если они следуют за аргументом *назначение*; имена полей, из которых берутся данные, если они следуют за аргументом *источник*.
внешняяБазаДанных	Путь к внешней базе данных.
источник	Имя таблицы или запроса, откуда копируются записи.
выражение	Имена таблицы или таблиц, откуда вставляются данные. Это выражение может быть именем отдельной таблицы или результатом операции INNER JOIN, LEFT JOIN или RIGHT JOIN, а также сохраненным запросом.
значение_1, *значение_2*	Значения, добавляемые в указанные поля новой записи. Каждое значение будет вставлено в поле, занимающее то же положение в списке: *значение_1* вставляется в *поле_1* в новой записи, *значение_2* - в *поле_2* и т. д. Каждое значение текстового поля следует заключать в кавычки (' '); для разделения значений используйте запятые.

Инструкцию INSERT INTO можно использовать для добавления одной записи в таблицу с помощью запроса на добавление одной записи, описанного выше. В этом случае инструкция содержит имя и значение каждого поля за-

писи. Нужно определить все поля записи, в которые будет помещено значение, и значения для этих полей. Если поля не определены, в недостающие столбцы будет вставлено значение по умолчанию или значение **Null.** Записи добавляются в конец таблицы.
Инструкцию INSERT INTO можно также использовать для добавления набора записей из другой таблицы или запроса с помощью предложения SELECT ... FROM, как показано выше в запросе на добавление нескольких записей. В этом случае предложение SELECT определяет поля, добавляемые в указанную таблицу *назначение*.
Вместо добавления существующих записей из другой таблицы, можно указать значения полей одной новой записи с помощью предложения VALUES. Если список полей опущен, предложение VALUES должно содержать значение для каждого поля таблицы; в противном случае инструкция INSERT не будет выполнена. Используйте дополнительную инструкцию INSERT INTO с предложением VALUES для каждой добавляемой новой записи.

4. Инструкция DELETE

Создает запрос на удаление записей, предназначенный для удаления записей из одной или нескольких таблиц, перечисленных в предложении FROM, которые удовлетворяют предложению WHERE.

DELETE [*таблица*.*]
  FROM *таблица*
  WHERE *условиеОтбора*

Ниже перечислены аргументы инструкции DELETE:

Элемент	**Описание**
таблица	Необязательное имя таблицы, из которой удаляются записи.
таблица	Имя таблицы, из которой удаляются записи.
условиеОтбора	Выражение, определяющее удаляемые записи.

Инструкция DELETE особенно удобна для удаления большого количества записей. При этом сохраняются структура таблицы и все остальные ее свойства, такие как атрибуты полей и индексы. Инструкцию DELETE можно использовать для удаления записей из таблиц, связанных отношением «один-ко-многим» с другими таблицами. Операции каскадного удаления приводят к удалению записей из таблиц, находящихся на стороне отношения «многие», когда в запросе удаляется соответствующая им запись на стороне «один». Запрос на удаление удаляет записи целиком, а не только содержимое указанных полей. Чтобы удалить данные в конкретном поле, следует создать запрос на обновление, который заменяет имеющиеся значения на значения **Null.**

5. Инструкция DROP

Удаляет существующую таблицу, процедуру или представление из базы данных или удаляет существующий индекс из таблицы.

DROP {TABLE *таблица* | INDEX *индекс* ON *таблица* | PROCEDURE *процедура* | VIEW *представление*}

Ниже перечислены аргументы инструкции DROP:

Элемент	**Описание**
таблица	Имя таблицы, которую следует удалить или из которой следует удалить индекс.
процедура	Имя удаляемой процедуры.
представление	Имя удаляемого представления.
индекс	Имя индекса, удаляемого из *таблицы*.

Прежде чем удалить таблицу или удалить из нее индекс, необходимо ее закрыть. Кроме того, для удаления индекса из таблицы можно использовать инструкцию ALTER TABLE.
Для создания таблицы можно использовать инструкцию CREATE TABLE, а для создания индекса - инструкцию CREATE INDEX или ALTER TABLE. Чтобы изменить таблицу, примените инструкцию ALTER TABLE.

6. Инструкция CREATE TABLE

Создает новую таблицу.

CREATE [TEMPORARY] TABLE *таблица* (*поле_1 тип* [(*размер*)] [NOT NULL] [WITH COMPRESSION | WITH COMP] [*индекс_1*] [, *поле_2 тип* [(*размер*)] [NOT NULL] [*индекс_2*] [, ...]] [, CONSTRAINT *составнойИндекс* [, ...]])

Ниже перечислены аргументы инструкции CREATE TABLE:

Элемент	**Описание**
таблица	Имя создаваемой таблицы.
поле_1, *поле_2*	Имена одного или нескольких полей, создаваемых в новой таблице. Таблица должна содержать хотя бы одно поле.
тип	Тип данных поля в новой таблице.
размер	Размер поля в знаках (только для текстовых и двоичных полей).
индекс_1, *индекс_2*	Предложение CONSTRAINT, предназначенное для создания простого индекса. Для получения более подробных сведений смотрите описание предложения CONSTRAINT.
составнойИндекс	Предложение CONSTRAINT, предназначенное для создания составного индекса. Для получения более подробных сведений смотрите описание предложения CONSTRAINT.

Инструкция CREATE TABLE используется для описания новой таблицы, ее полей и индексов. Если для поля добавлено ограничение NOT NULL, то при добавлении новых записей это поле должно содержать допустимые данные. Предложение CONSTRAINT устанавливает различные ограничения на поле и может быть использовано для определения ключа. Кроме того, для создания ключа или дополнительного индекса для существующей таблицы можно использовать инструкцию CREATE INDEX.
Допускается использование ограничения NOT NULL для одиночного поля, а также внутри именованного предложения CONSTRAINT, которое применяется к одиночному полю или к именованному предложению CONSTRAINT, предназначенному для создания составного индекса. Однако ограничение NOT NULL можно наложить на поле только один раз. При попытке применить это ограничение несколько раз возникает ошибка выполнения.
Создаваемая временная (TEMPORARY) таблица будет доступна только в том сеансе, где эта таблица была создана. По завершении данного сеанса она автоматически удаляется. Временные таблицы могут быть доступны для нескольких пользователей.
Использование атрибута WITH COMPRESSION допускается только для типов данных CHARACTER и MEMO (он же TEXT) и их синонимов. Атрибут WITH COMPRESSION был добавлен к столбцам CHARACTER вследствие перехода к формату представления знаков Юникод. Если столбец CHARACTER был определен с этим атрибутом, то при сохранении в нем данных осуществляется их автоматическое сжатие, а при извлечении данных - обратная операция.
Предусмотрены следующие типы данных: **Big Integer**, **Binary**, **Byte**, **Boolean**, **Char**, **Currency**, **Date**, **Decimal**, **Double**, **Float**, **GUID**, **Integer**, **Long**, **Long Binary** (**OLE Object**), **Memo**, **Numeric**, **Single**, **String**, **Text**, **Time**, **TimeStamp**, **VarBinary**, **Variant** (используемый по умолчанию), типы, определенные пользователем (создаваемые с помощью инструкции **Type**), и объектные типы данных, которые включают объектные типы данных приложения и типы объектов доступа к данным.

7. Инструкция UPDATE

Создает запрос на обновление, который изменяет значения полей указанной таблицы на основе заданного условия отбора.

UPDATE *таблица*
  SET *новоеЗначение*
  WHERE *условиеОтбора*;

Ниже перечислены аргументы инструкции UPDATE:

Элемент	Описание
таблица	Имя таблицы, данные в которой следует изменить.
новоеЗначение	Выражение, определяющее значение, которое должно быть вставлено в указанное поле обновленных записей.
условиеОтбора	Выражение, отбирающее записи, которые должны быть изменены. При выполнении этой инструкции будут изменены только записи, удовлетворяющие указанному условию.

Инструкцию UPDATE особенно удобно использовать для изменения сразу многих записей или в том случае, если записи, подлежащие изменению, находятся в разных таблицах.
Одновременно можно изменить значения нескольких полей. Следующая инструкция SQL увеличивает стоимость заказа на 10 процентов, а стоимость доставки на 3 процента:

```
UPDATE Заказы
SET СуммаЗаказа = СуммаЗаказа * 1.1,
СтоимостьДоставки = СтоимостьДоставки * 1.03
WHERE СтранаПолучателя = 'Литва';
```

8. Операция UNION

Создает запрос на объединение, который объединяет результаты нескольких независимых запросов или таблиц.

[TABLE] *запрос_1* UNION [ALL] [TABLE] *запрос_2* [UNION [ALL] [TABLE] *запрос_n* [...]]

Ниже перечислены аргументы операции UNION:

Элемент	Описание
запрос_1-n	Инструкция SELECT, имя сохраненного запроса или имя сохраненной таблицы, перед которым стоит зарезервированное слово TABLE.

В одной операции UNION можно объединить в любом наборе результаты нескольких запросов, таблиц и инструкций SELECT. В следующем примере объединяется существующая таблица «Новые счета» и инструкции SELECT:

```
TABLE [Новые счета] UNION ALL
SELECT *
FROM Клиенты
WHERE СуммаЗаказа > 1000;
```

По умолчанию повторяющиеся записи не возвращаются при использовании операции UNION, однако в нее можно добавить предикат ALL, чтобы гарантировать возврат всех записей. Кроме того, такие запросы выполняются быстрее. Все запросы, включенные в операцию UNION, должны отбирать одинаковое число полей; при этом типы данных и размеры полей не обязаны

совпадать. В каждом аргументе *запрос* допускается использование предложения GROUP BY или HAVING для группировки возвращаемых данных. В конец последнего аргумента *запрос* можно включить предложение ORDER BY, чтобы отсортировать возвращенные данные.

9. Статистические функции SQL

Статистические функции SQL используются для определения статистических данных на основе наборов числовых значений.

Функция **Avg**
Функция **Count**
Функции **First**, **Last**
Функции **Min**, **Max**
Функции **StDev**, **StDevP**
Функция **Sum**
Функции **Var**, **VarP**

5. ПРОГРАММА РАБОТЫ

1. Ознакомьтесь с теоретическими положениями.
2. Откройте БД.

№	ФИО студента	№ зачетки	Дисциплина	Вид ИА	Дата сдачи	Оценка
				Экзамен|зачет	Текущая дата	

3. Войдите в SQL-режим конструктора запросов и создайте таблицы в соответствии с указанной структурой, перенесите в них данные из ранее сформированной таблицы;

№	ФИО студента	№ зачетки
Ключ		

4. В SQL-режиме конструктора запросов создайте запросы на добавление/удаление новых записей в таблицы;
5. Из исходной таблицы удалите поле ФИО студента, сделайте поле «№зачетки» ключевым;
6. Выведите информацию о студентах, получивших за указанный период оценку «отлично» на экзамене.
7. Выведите список дисциплин (2-3), по которым студенты получают наихудшие оценки (в среднем);
8. Выведите ФИО студента, имеющего наилучшую успеваемость.

1. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое SQL-запросы?
2. Как создать SQL-запрос в ?
3. Какие SQL-инструкции реализованы в ?

Лабораторная работа № 9

ПРИМЕНЕНИЕ КОНСТРУКТОРА И МАСТЕРА ЗАПРОСОВ

1. ЦЕЛЬ РАБОТЫ

Изучение средств автоматизации формирования запросов в СУБД.

Отработка методов конструирования запросов, форм представления запросов и их реализация.

2. ОСНОВЫ ТЕОРИИ

Практическое использование базы данных основано на выборке из исходных таблиц определенного сорта информации, удовлетворяющей определенным критериям. Критерии могут определяться сочетанием ряда условий. Для решения задач, связанных с выборкой данных, предназначены мастера и конструкторы запросов.

Под запросом обычно понимается вопрос, сформулированный к БД. реализует метод формирования запроса по образцу. Запрос по образцу – это интерактивное средство для выбора данных из одной или нескольких таблиц. Формирование запроса осуществляется путем заполнения бланка запроса, который располагается в окне конструктора запросов. Такой метод формирования запроса прост для изучения и способствует эффективному использованию возможностей СУБД.

Для создания простейших запросов можно использовать мастер запросов.

Создание запросов

Неоспоримым преимуществом мастеров является возможность быстрого получения результатов при минимуме знаний о механизме его получения. Но, к сожалению, мастера запросов не позволяют указать условие выборки, критерий упорядочивания и более сложные, но и более интересные параметры выборки. Применение конструктора запросов в таких случаях позволяет создать сложные запросы в интерактивном режиме.

Окно конструктора запросов

Для вызова конструктора запросов перейдите в окне базы данных на вкладку «Запросы» и нажмите кнопку **Создать. В** окне диалога «Новый запрос» выберите опцию **Конструктор** и нажмите кнопку **ОК.** предложит вам выбрать таблицу или запрос, на основе которых будет осуществляться выборка. На этом этапе выбор таблицы или запроса не обязателен (вы можете добавить их и позже), но очевидно, что запрос должен выполняться, по крайней мере, на основе какой-либо таблицы или ранее созданного запроса. Выберите таблицу, нажмите кнопку **Добавить** и закройте окно диалога. На

экране появится окно конструктора запросов , а в основном меню — команда **Запрос.**

Для формирования запроса в окне конструктора запросов необходимо выбрать таблицы, из которых осуществляется выборка, и поля результата запроса, указать критерии для выборки, группировки и упорядочивания данных.

Меню **Запрос** содержит команды добавления таблиц в окно конструктора запросов и удаления их из нее, команды выбора типа запросов (Выборка, Создание таблицы, Перекрестный, Обновление, Добавление, Удаление), команду запуска запроса и управления запросом (Переключатель режимов, Групповые операции, Добавление Имени таблицы, Свойства).

В верхней части окна конструктора запросов находится схема данных запроса. Эта схема очень сильно напоминает схему данных базы данных. В отличие от нее, данная схема содержит список таблиц, включенных в запрос, и отображает связи между ними. В нижней части окна располагается бланк запроса. Каждая строка этого бланка выполняет определенную функцию:

- Поле. В этой строке помещаются те поля, которые вы используете для создания запроса, каждое в своей ячейке таблицы.

- Имя таблицы. Эта строка показывает вам, из какой таблицы (или запроса) выбрано данное поле.

- Сортировка. В этой строке вы указываете тип сортировки информации, возвращаемой в запросе, по возрастанию (от А до Я, от большего к меньшему, от более раннего к более позднему и т.д.) или по убыванию (от Я к А и т.д.).

- Вывод на экран. Если вы хотите, чтобы показывал информацию, найденную в поле, пометьте эту ячейку, чтобы установить флажок просмотра поля. Если же поле используется только для задания условия выбора данных, которые возвращает ваш запрос, оставьте его пустым.

- Условие отбора. В этой строке (и в строке, расположенной ниже ее) вы вводите ограничения поиска, задавая определенные условия, которые принято называть *критерием поиска.*

Создание простого запроса

Для создания простейших запросов достаточно данных одной таблицы. Рассмотрим создание такого запроса.

Вся необходимая информация находится в выбранной вами таблице. Поэтому для создания запроса выполните следующие действия:

1) В окне базы данных перейдите на вкладку «Запросы» и нажмите кнопку **Создать**. 2) Откроется окно диалога «Добавление таблицы», в котором выберите нужную таблицу и нажмите кнопку **Добавить**. Закройте окно диалога. 3) На экране открывается окно конструктора запросов, схема данных которого содержит всего одну таблицу, а бланк запроса пуст.

Добавление полей в бланк запроса

Для выбора полей, которые должны присутствовать в результирующей таблице, вам необходимо отобразить их в бланке запроса.

В существует два варианта выбора полей результирующей таблицы. Вы можете воспользоваться наиболее приемлемым с вашей точки зрения:

1. Для добавления в таблицу отдельных полей вы можете выбрать поле таблицы на схеме данных и дважды нажать кнопку мыши. Выбранное поле будет вставлено в следующий доступный столбец в строке Поле бланка запроса. В строке **Имя таблицы** сразу же появится имя таблицы, а позиция **Вывод на экран** будет помечена.

2. В широко используется механизм *перенести-и-оставить* (drag-and-drop). Для использования этого механизма при выборе полей перейдите в таблицу в схеме данных, из которой вам надо выбрать поля. Выделите поля, которые вы собираетесь отобразить в запросе, нажмите кнопку мыши и, не отпуская ее, перенесите выбранные поля в бланк запроса.

Для выделения группы полей используется мышь совместно с клавишами *Shift* и *Ctrl.* Для выбора отдельно расположенных полей вместо клавиши *Shift* используйте клавишу *Ctrl.*

В некоторых случаях необходимо выбрать все поля исходной таблицы. Для этого в существуют два способа, первый из которых состоит в выборе всех полей таблицы двойным нажатием мыши на строке заголовка и переносе выделенных полей в бланк запроса.

Вы можете использовать звездочку в списке полей таблицы для пометки всех полей в таблице. Для этого нажмите на звездочку в первой строке списка полей и, удерживая кнопку мыши в нажатом состоянии, перенесите ее в бланк запроса. Имя поля в бланке запроса будет содержать имя таблицы, за которым следует точка, а затем символ звездочки (например, клиенты.*). Это означает, что были выбраны все поля исходной таблицы. В отличие от первого способа, в режиме конструктора вы не увидите каждое поле в отдельном столбце, но после запуска запроса все они будут выбраны.

У метода переноса звездочки есть одно большое преимущество. Если вы добавите в таблицу новое поле после того, как вы создали и запустили запрос, включающий «*» ваш запрос автоматически включит в себя новое поле. При использовании первого способа запрос выбирает только те поля, которые были перенесены в бланк запроса.

Удаление полей из бланка запроса

Команда **Правка | Очистить бланк** удаляет все поля из бланка запроса.

Для удаления лишнего поля из запроса нажмите на область выбора столбца, а затем клавишу *Delete.* Поле исчезнет.

Прежде чем удалить поле, удостоверьтесь, что оно не используется в вашем запросе для определения условия выборки. В этом случае для запрета отображения поля в результирующей таблице снимите флажок **Вывод на экран.**

Изменение порядка полей

Порядок полей в бланке запроса определяет порядок появления их в результирующей таблице. Для того чтобы изменить расположение поля в этом списке, выполните следующие действия:

1) Установите указатель мыши на область выбора столбца, который располагается прямо над названием поля. Когда указатель изменит вид на стрелку, нажмите кнопку, чтобы выделить столбец. 2) Нажмите и удерживайте кнопку мыши в этом положении. На конце указателя появится прямоугольник. 3) Перемещайте столбец в требуемом направлении. Толстая вертикальная линия покажет его текущее положение. 4) Отпустите кнопку, когда толстая вертикальная линия окажется в требуемом месте. Поле будет перемещено в новое место.

Сортировка результатов выборки

Вы можете сортировать результаты выборки по одному или нескольким полям так же, как вы это делали при сортировке таблицы.

Порядок сортировки записей результирующей таблицы определяется порядком следования полей в бланке запроса и критерием упорядочивания отдельных полей. Чтобы отсортировать данные в отдельном поле, перейдите на строку **Сортировка** требуемого поля и из раскрывающегося списка выберите значение **По возрастанию** или **По убыванию.** Если вы решили отказаться от сортировки по полю, выберите значение **Отсутствует**. Для сортировки по нескольким полям последовательно переходите на сортируемые поля и установите для них критерий сортировки. В отличие от сортировки в режиме таблицы, вы увидите результат только после выполнения запроса.

Запуск запроса

Мы закончили создание простого запроса, и наступило время запустить его

на выполнение. Нажмите кнопку Запуск на панели инструментов или выполните команду **Запрос| Запуск.**

Сохранение запроса

Созданный запрос можно использовать в дальнейшем. Для этого вы Должны присвоить ему имя и сохранить его. Сохранение запроса осуществляется командой **Файл| Сохранить как/Экспорт,** которое откроет окно диалога ввода имени запроса. по умолчанию предложит имя запроса в поле ввода **Новое имя**, но лучше подобрать что-нибудь более значимое.

Использование критерия выборки записей для ограничения поиска

Предположим, что вы захотели при формировании запроса ограничиться только вашими знакомыми из Ленинграда. Для этого вы должны задать критерий выборки записей, который определяется следующим образом:

1. В списке полей таблицы найдите нужное поле и дважды нажмите на нем мышью, чтобы добавить в бланк запроса.

2. В строке **Условие отбора** для только что созданного поля введите его значение и снимите флажок вывода на экран для поля , так как это поле используется только для задания условия выборки записей.

3. Теперь запустите запрос, и результаты появятся в режиме таблицы,

4. Вы можете сохранить этот запрос под новым именем и использовать его в дальнейшем.

Изменение внешнего вида результирующей таблицы

Вы можете достаточно легко изменить внешний вид результирующей таблицы, используя те же средства, что и для обычных таблиц. Например, вы можете делать поля невидимыми, зафиксировать их, изменить шрифт, размеры столбцов и строк. Для выполнения этих функций перейдите в режим таблицы и выберите соответствующую команду из меню **Формат.**

3. ОБЪЕКТЫ И СРЕДСТВА ИССЛЕДОВАНИЯ

Предметная область в соответствии с вариантом задания. ПК типа IBM PC, дисплейный класс, операционная система Windows XP и выше, СУБД OO Base.

4. ПОДГОТОВКА К РАБОТЕ

Ознакомиться с теоретическими положениями лабораторной работы.

5. ПРОГРАММА РАБОТЫ

По предложенному преподавателем варианту разработать несколько вариантов часто используемых запросов по выборке данных из таблиц. Средствами конструктора запросов сформировать необходимые поля в бланке запросов. Предусмотреть сортировку выборок, а также различных критериев выборки записей из полей в соответствии с характером задания на лабораторные работы. Оформить отчет.

ВАРИАНТЫ ЗАДАНИЙ . Выбирается преподавателем из списка заданий (См. Приложение) и остается единым для всех лабораторных работ.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

6.1. Поясните метод формирования запросов к БД по образцу?

6.2. Опишите общую концепцию построения запросов в СУБД ?

6.3. Расскажите о технологии использования критериев выборки данных в вашей БД?

6.4. Опишите используемый вами способ формирования многотабличных запросов?

Лабораторная работа № 10

СОЗДАНИЕ В ЗАПРОСАХ ВЫЧИСЛЯЕМЫХ ПОЛЕЙ. ИСПОЛЬЗОВАНИЕ УСЛОВИЙ

1. ЦЕЛЬ РАБОТЫ

Изучение средств автоматизации формирования запросов в СУБД.

Отработка методов конструирования запросов, форм представления запросов и их реализация.

2. ОСНОВЫ ТЕОРИИ

Использование в запросах вычислимых полей

В результате выполнения запроса позволяет вам не только выбирать из таблицы содержащуюся в ней информацию, но также производить вычисления и отображать результат вычисления в результирующей таблице. Таким образом, вы можете получить данные, отсутствующие в исходной таблице.

При выполнении запроса вы можете вычислять значения по одному или нескольким полям исходной таблицы. Кроме того, вы можете использовать вычисляемые поля для объединения нескольких полей исходной таблицы в одно выходное поле. Например, вы можете в выходной таблице объединить фамилию и имя сотрудника.

Результаты вычислений, выводящиеся в поле, не запоминаются в исходной таблице. Вместо этого, вычисления выполняются каждый раз при запуске запроса, поэтому результаты всегда представляют текущее содержимое базы данных. Вы не можете редактировать результаты вычисления.

Для расчетов с использованием формул, определяемых пользователем, требуется создать новое вычисляемое поле прямо в бланке запроса. Вычисляемое поле создается с помощью выражения, которое вводится в пустую ячейку поля в бланке запроса или создается с помощью построителя выражений.

Выражение содержит формулы, которые связываются с помощью операторов. В качестве элементов формулы могут использоваться поля, константы и функции. Для изменения порядка вычислений и группировки данных в выражениях используются круглые скобки.

Обратите внимание, что наименование поля в выражении записывается в виде [Имя Таблицы] ! [Поле]. Имена таблицы и поля заключены в квадратные скобки, а между ними находится восклицательный знак. На этом примере видно неоспоримое преимущество использования построителя: вам не обязательно знать все принятые в соглашения. По выбранному вами значению построитель сам преобразует его в требуемый формат.

После завершения формирования выражения нажмите кнопку **ОК,** и выражение будет перенесено в строку **Поле** бланка запроса. Так как каждое поле результирующей таблицы должно иметь имя, автоматически задаст имя вычисляемого поля (например. Выражение 1), которое отделяется от выражения двоеточием.

Построение условий для выбора записей

На практике в большинстве случаев требуется получить не все записи исходной таблицы, а лишь ту часть, которая удовлетворяет определенным условиям. Простейший критерий для выбора записей предполагает точное совпадение значений поля. Определение такого критерия было рассмотрено ранее, поэтому перейдем к построению более сложных условий выборки записей.

Точное несовпадение значений одного из полей

Возможно, потребуется найти в таблице записи, значения которых не удовлетворяют определенному условию. Для установки таких условий используется оператор Not, который печатается перед сравниваемым значением. Выполните следующие действия:

1. В строке **Условие отбора** поля вставьте перед сравниваемым значением оператор Not или <>.

2. Проверьте установку флажка вывода на экран для контролируемого поля.

3. Для выполнения запроса нажмите кнопку Запуск. На экране появится результирующая таблица, которая содержит записи вне значения контролируемого поля.

Условие неточного совпадения

Требование точного задания чисел или последовательности символов в качестве критерия поиска является жестким ограничением. Время от времени возникает неприятная ситуация, при которой вы не помните точного написания условия поиска. Кроме того, вы можете просто не знать, какой регистр (верхний, нижний или их сочетание) был использован при вводе каждой записи. При работе с большими таблицами такая ситуация превращается в серьезную проблему.

В подобных случаях вы можете осуществить выбор записей по условию неточного совпадения значений. Данное условие позволяет найти требуемые записи, зная лишь приблизительное написание значения. В для этих целей используется оператор Like, который осуществляет сравнение значения поля с заданным образцом. Образец задается сразу же за оператором и может содержать точное значение (например, Like "Иванов") или использовать символы шаблонов для поиска диапазона значений (например, Like "Ив*"). Приведенная ниже таблица содержит символы шаблонов, которые используются с оператором Like, и количество цифр или символов, которым они соответствуют.

Символы в образце	**Соответствие в выражении**
?	Любой один знак
*	Нуль или более знаков
#	Любая одна цифра
[список_зиаков]	Любой один знак в **списке_знаков**
[!спнсок_знаков]	Любой один знак, не входящий в **список_знаков**

Группа из одного или более символов, заключенная в квадратные скобки, используется для установления совпадения с одним символом выражения и может содержать любые символы, за исключением перечисленных ниже особых символов.

Особыми символами являются открывающая квадратная скобка ([), вопросительный знак (?), знак числа (#) и знак звездочки (*). Для сравнения с ними необходимо заключить их в квадратные скобки. Закрывающую квадратную скобку нельзя применять в группе для установки соответствия с ней самой, но она может стоять вне группы как отдельный символ.

Помимо простого списка символов, заключенного в квадратные скобки, **список_знаков** позволяет задать диапазон символов. Вы можете указать знак (-) для разделения верхней границы диапазона от нижней. Например, использование шаблона [Г-Л] в образце позволяет выбрать все записи, которые в указанной позиции поля содержат одну из заглавных букв в диапазоне от Г до Л. Вы можете использовать даже несколько диапазонов внутри одной пары квадратных скобок без специальных разделителей. Например, [г-лГ-Л] позволяет выбрать как заглавные, так и прописные буквы.

Восклицательный знак в начале **списка_знаков** означает, что совпадение наступит, если в выражении будет найден любой символ, отсутствующий в **списке знаков.** Восклицательный знак, использованный вне квадратных скобок, соответствует самому себе.

Знак (-) для установления соответствия с самим собой можно использовать в начале (после восклицательного знака, если он есть) или в конце **списка_зваков.**

Рассмотрим простой пример использования оператора **Like.** Предположим, что в таблице Клиенты требуется найти запись о представителе фирмы, название которой начинается на Ric. Однако вы не уверены в правильности написания фирмы. Попытаемся найти требуемую запись. Для задания условия выберите поле с названием и в строке **Условие** отбора поля напечатайте Like 'Ric*' .

Рассмотрим более сложный пример использования оператора **Like.** Допустим, вам надо выбрать всех клиентов, наименования которых начинаются на А или С. Для этого в строке **Условие отбора** поля Название введите следующее условие: Like ' [АС] * '.

Выбор записей по диапазону значений

Часто необходимо выбрать из таблицы записи , данные которых попадают в диапазон значений. При этом границы диапазона заданы точно.

Для задания диапазона значений в окне конструктора запросов используются операторы > (больше), >= (не менее), < (меньше), <= (не более) **и Between,** которые вы можете использовать с текстовыми и цифровыми полями, а также полями дат. Для задания диапазона значений этого выполните следующие действия:

1) Откройте на экране окно конструктора запросов для нужной таблицы. 2) Выделите поля, которые хотите отобразить в запросе и перенесите их в бланк запроса. 3) Для задания условия отбора выберите поле и в строке **Условие отбора** поля напечатайте условие(например, >100000). 4) Нажмите кнопку **Запуск,** и на экране появится результирующая таблица, содержащая записи о значениях полей свыше 100 000.

В рассмотренном примере мы задавали только одну нижнюю границу диапазона. Очевидно, что можно использовать более сложные конструкции операторов

В заключение рассмотрим пример задания диапазона строковых данных. В этом случае кроме операторов сравнения вы можете использовать и оператор Like. Например, для получения списка клиентов, наименования которых начинаются с С по G, в строку Условие **отбора** поля Название введите условие отбора : Like '[C-G]* '

Объединение критериев нескольких полей

В предыдущих примерах мы вводили условие запроса только для одного из полей таблицы. Однако довольно часто возникают ситуации, когда вам необходимо использовать более сложный критерий выборки, в котором задаются условия для нескольких полей таблицы или же несколько условий для одного поля. Если запись выбирается только в случае выполнения всех условий, то условие такого выбора называется логическим И, а запрос — И-запросом. Если же запись выбирается при выполнении хотя бы одного из всех условий, то условие такого поиска называется логическим ИЛИ, а запрос — ИЛИ- запросом.

Для задания И- выражения вы должны просто задать условие в строке **Условие отбора** для каждого из полей, образующих критерий. В качестве примера предположим что из всех записей о заказах за определенный интервал времени требуется выбрать те, код доставки которых равен 1. Для решения этой задачи откройте запрос, в котором выбирались заказы в диапазоне дат, и выполните следующие действия:

1) Добавьте в бланк запроса поле Доставка. 2) Перейдите на строку **Условие отбора** поля и напечатайте 1 . 3) Выполните запрос, и вы увидите записи, содержащие сведения о заказах за указанный период дат, в которых использовалась доставка с кодом 1.

При задании ИЛИ- выражения каждое из условий выбора, образующих критерий должно располагаться на отдельной строке бланка запроса. Напри-

мер, для выбора списка клиентов из США и Швеции нужно просто расположить первое условие в строке Условие **отбора,** а второе - в строке **Или .**

При формировании ИЛИ- выражения вы можете расположить условия выбора для различных полей в разных строках бланка запроса.

При вводе условия вы можете использовать операторы **Or** и **And,** которые позволяют вам формировать в одной строке сложное условие выборки. Например, при поиске записей о клиентах из США и Швеции, вы можете поместить ИЛИ- выражение в одной строке. Результирующая таблица будет содержать те же записи, что и запрос. При вводе условия вы можете формировать любое допустимое в логическое условие, которое может содержать функции, операторы сравнения, Or, And, Not и скобки для изменения порядка выполнения выражения.

Многотабличные запросы

При выборе данных из таблиц наиболее часто используются многотабличные запросы, поскольку информация в реляционных базах данных содержится не в одной отдельной таблице, а в совокупности связанных таблиц. Связь между таблицами осуществляется на основании совпадающих полей.

Для формирования многотабличного запроса нужно добавить в окно конструктора запросов все таблицы, участвующие в выборке, и определить условия их объединения. Для добавления таблицы выберите команду **Запрос | Добавить таблицу** или нажмите кнопку **Добавить таблицу** на панели инструментов. В открывшемся окне диалога «Добавление таблицы» выберите нужную таблицу. Образ таблицы появится в схеме данных запроса. Если в базе данных установлены отношения между таблицами, участвующими в запросе, то эта связь будет отображаться в виде линии, соединяющей таблицы. В этом случае вам не придется устанавливать связь между таблицами в конструкторе запросов. Если же между таблицами не существует связи, то вы можете установить требуемую связь, используя механизм перенести -и-оставить. Для этого выберите поле в одной из таблиц, нажмите кнопку мыши и перенесите выбранное поле на связываемое поле в другой таблице.

В отличие от определения постоянных отношений между таблицами, при задании условия объединения таблиц вы можете использовать любые поля таблиц.

Объединение двух таблиц

Вначале рассмотрим простое объединение двух таблиц на следующем примере. Отпускаемый клиентам товар может доставляться разными средствами. Для определения типа доставки товаров необходимо использовать информацию из двух связанных таблиц Т1 и ТО. Пусть при создании базы данных между этими таблицами уже определены постоянные отношения. Для решения этой задачи выполните следующие действия:

1) Добавьте таблицу Т1 в новое окно конструктора запросов. 2) Нажмите кнопку **Добавить таблицу** на панели инструментов и добавьте в запрос

еще одну таблицу — ТО. Поскольку между этими таблицами установлено постоянное отношение, то после добавления таблицы в окне конструктора запросов автоматически отобразится связь между ними. 3) Перенесите в бланк запроса из таблицы ТО код товара, а из таблицы Т1 — код заказа и тип доставки . 4) Выполните запрос, и на экране появится результирующая таблица, содержащая требуемую информацию .

Условие отбора в многотабличных запросах

Теперь обратимся к более сложному запросу, в котором из всех записей в таблицах необходимо выбрать только те записи, которые отвечают определенному условию отбора.

Для задания условия отбора перейдите на строку **Условие отбора** поля, по которому производится выборка, снимите флажок **Вывод на экран** для этого поля**,** так как не имеет смысла выводить столбец, каждая строка которого будет содержать одно и то же значение. Перейдите на **Условие отбора** поля другой записи, а затем снимите для нее флажок **Вывод на экран.** Далее добавьте в бланк запроса дополнительные необходимые поля и выполните запрос. Вы увидите на экране результирующую таблицу, содержащую информацию **обо** всех интересующих вас значениях.

Итоговые запросы

При необходимости можно объединить в запросе любое количество требуемых таблиц. Однако довольно часто имеющихся в таблицах данных оказывается недостаточно. Например, вам нужно найти сумму, максимальную величину в поле или количество записей, содержащих определенную величину. В этом случае вам необходимо выполнить итоговые вычисления. Для этих целей используются математические возможности , который отлично приспособлен для выполнения таких вычислений.

Для определения суммы значений полей или нахождения среднего следует создать итоговый запрос. Для его создания, как и для обычного запроса, откройте новое окно конструктора запросов. Далее выберите используемые в запросе таблицы, а затем перенесите в бланк запроса нужные поля.

Запросы, выполняющие вычисления в группах записей, называются *итоговыми запросами.* В итоговом запросе выполняется не только Суммирование, но и другие виды вычислений. Например, вы можете найти среднее, минимальное и максимальное значения поля.

Для создания итогового запроса выберите **Вид| Групповые операции** или нажмите кнопку **Групповые операции** на панели инструментов. В бланке запроса появится новая строка с наименованием **Групповая операция**. В этой строке вы должны указать тип выполняемого вычисления (например, суммирование – **Sum).**

Табл. 4.1 содержит перечень всех допустимых видов итоговых операций, которые можно выбрать из раскрывающегося списка в строке Групповая операция. Две из них — StDev и Var -являются известными операциями

статистки. Операция Count требует некоторых дополнительных разъяснений. Она действует следующим образом: просматриваются все записи для указанного поля, и вычисляется количество записей, которые содержат в этом поле какое-либо значение. Другими словами, данная операция пропускает пустые значения.

Таблица. 4.1

Типы операций, доступные в строке **Групповая операция** бланка запроса

Значение	Выполняемая операция	Значение	**Выполняемая операция**
Sum	Сложение	**Min**	Минимальное значение
Avg	Среднее значение	**Count**	Количество записей, содержащих значения
Var	Дисперсия	**First, Last**	Значение в первой и последней записи
Max	Максимальное значение	**StDev**	Стандартное отклонение

Для удаления строки **Групиовая операция** достаточно нажать еще раз на кнопку **Гпупповая операция**.

Группировка полей запроса

Группировка позволяет получить вычисляемую информацию о подгруппах таблицы. Например, сгруппировав данные в таблице Заказано по полю КодТовара, можно получить сведения об итоговом количестве заказов каждого вида товара.

Как обычно, создание запроса начните с добавления таблицы в окно конструктора запросов. Далее выполните следующие действия:

1) После размещения полей в бланке запроса выберите команду **Вид | Групповые операции** для добавления в бланк запроса **строки Групповая операция.** 2) В строке **Групповая операция** группируемого столбца установите значение **Группировка,** а в другом столбце можно подсчитать количество записей, установив значение **Count.**

При формировании групповых итоговых запросов довольно часто данные упорядочиваются по итоговым полям, Например, в запросе для поля, в котором определяется количество записей, установлен признак сортировки по убыванию. Это означает, что в результирующей таблице первыми будут расположены наиболее часто встречающиеся записи.

Поле, используемое для группировки, не обязательно должно находиться в гой же таблице, что и итоговое поле. Оно может находиться в другой таблице.

Группировка по нескольким полям

Вы можете группировать данные не только по одному полю, но и по нескольким полям одновременно, а также создавать группы внутри групп. Для этого вы можете сгруппировать записи сначала по одному полю, затем — по другому. Порядок полей группировки, размещенных в бланке запроса, определяет порядок группировки выбираемых данных.

Включение в запрос выражений

предоставляет вам возможность выполнять итоговые операции над вычисляемыми полями выборки. Например, таблица Заказы содержит данные о клиентах, а таблица Заказано — о количестве товаров, проданных в каждой партии и цене товара. На основании этой информации вы можете вычислить итоговую стоимость заказа каждым покупателем по каждому товару.

Чтобы включить в итоговый запрос выражение, добавьте в бланк запроса вычисляемое поле и укажите тип итоговых имений, осуществляемых над этим полем (т.е. выражение для вычисления).

Выражения для вычисления могут содержать формулы, связанные арифметическими операторами. В качестве элементов формулы могут использоваться поля, константы и функции. Для изменения порядка вычислений в выражениях используются круглые скобки.

Изменение наименований итоговых полей

По умолчанию поля результирующей таблицы имеют те же наименования, что и поля исходной таблицы, а итоговым полям присваиваются наименования в соответствии с принятым в соглашением. Начальная часть имени обычно содержит имя итоговой операции, за которым следует имя поля, над которым выполняется итоговая операция. Во многих случаях предпочтительнее давать таким полям более осмысленные наименования. позволяет вам по своему усмотрению изменить наименования полей результирующей таблицы.

Для изменения наименования поля в строке **Поле** перед именем поля или выражения напечатайте новое имя поля и отделите его двоеточием.

Изменение наименований полей в бланке запроса действует только на поля в результирующей таблице. Имена полей в исходной таблице остаются без изменения.

Использование более сложных условий отбора записей в итоговых запросах

В итоговых запросах существует два типа критериев отбора записей. Первый тип аналогичен критерию, используемому в обычных запросах. Он исключает записи, не удовлетворяющие заданному критерию, перед выполнением итоговых вычислений. Второй тип является специфичным для итоговых запросов. Данный критерий применяется к результату итоговых вычислений. Критерии вводятся в строке **Условие отбора.**

3. ОБЪЕКТЫ И СРЕДСТВА ИССЛЕДОВАНИЯ

Предметная область в соответствии с вариантом задания. ПК типа IBM PC, дисплейный класс, операционная система Windows XP и выше, СУБД OO Base.

4. ПОДГОТОВКА К РАБОТЕ

Ознакомиться с теоретическими положениями лабораторной работы.

5. ПРОГРАММА РАБОТЫ

По предложенному преподавателем варианту разработать несколько вариантов часто используемых запросов по выборке данных из таблиц. Использовать в запросе одно или несколько вычислимых полей. Разработать один или несколько вариантов многотабличных запросов с использованием группировки полей. Разработать необходимые формы для просмотра и редактирования данных с использованием конструктора форм. Оформить отчет.

ВАРИАНТЫ ЗАДАНИЙ . Выбирается преподавателем из списка заданий (См. Приложение) и остается единым для всех лабораторных работ.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

6.1. Поясните метод формирования запросов к БД по образцу?

6.2. Опишите общую концепцию построения запросов в СУБД ?

6.3. Расскажите о технологии использования критериев выборки данных в вашей БД?

6.4. Опишите используемый вами способ формирования многотабличных запросов?

Лабораторная работа № 11

УПРАВЛЕНИЕ ДОСТУПОМ К ОБЪЕКТАМ БАЗЫ ДАННЫХ

1. ЦЕЛЬ РАБОТЫ

Получить навыки интеграции различных баз данных с приложениями.

ОСНОВЫ ТЕОРИИ

Компоненты ADO

Компоненты для работы с Microsoft® ActiveX® Data Objects (далее ADO) - это технология стандартного обращения к реляционным данным от Microsoft. Эта технология аналогична BDE по назначению и довольно близка по возможностям.

Обзор компонент

Для работы с ADO на вкладке компонентов ADO есть шесть компонентов: **TADOConnection, TADOCommand, TADODataSet, TADOTable, TADOQuery, TADOStoredProc.**

Рис. 6.1. Палитра компонент ADO

TADOConnection аналогичен компоненту BDE **TDatabase** и используется для указания базы данных и работы транзакциями.

TADOTable – таблица доступная через ADO.

TADOQuery – запрос к базе данных. Это может быть как запрос, в результате которого возвращаются данные и базы (например, SELECT), так и запрос не возвращающий данных (например, INSERT).

TADOStoredProc – вызов хранимой процедуры. В отличие от BDE и InterBase хранимые процедуры в ADO могут возвращать набор данных, поэтому компонент данного типа является потомком от **TDataSet** и может выступать источником данных в компонентах типа **TDataSource**.

TADOCommand и **TADODataSet** являются наиболее общими компонентами для работы с ADO, но и наиболее сложными в работе. Оба компонента позволяют выполнять команды на языке провайдера данных (так в ADO называется драйвер базы данных).

Разница между ними в том, что команда, исполняемая через **TADODataSet,** должна возвращать набор данных и этот компонент позволяет работать с ними средствами Delphi (например, привязать компонент типа

TDataSource). А компонент **TADOCommand** позволяет исполнять команды не возвращающие набор данных, но не имеет штатных средств Delphi для последующего использования возвращенного набора данных.

Очевидно, что все компоненты должны связываться с базой данных. Делается это двумя способами либо через компонент **TADOConnection** либо прямым указанием базы данных в остальных компонентах. К **TADOConnection** остальные компоненты привязываются с помощью свойства **Connection**, к базе данных напрямую через свойство **ConnectionString.**

База данных может быть указана двумя способами через файл линка к данным (файл в формате Microsoft Data Link, расширение UDL), либо прямым заданием параметров соединения.

Значение свойства всех **ConnectionString** этих компонент могут быть введены напрямую в текстовой форме, но куда проще вызвать редактор свойства нажав на кнопку «…» в конце поля ввода. Окно этого свойства выглядит так:

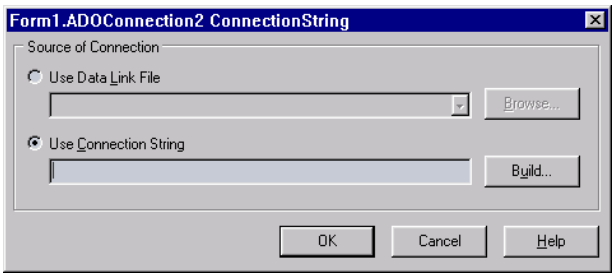


Рис. 6.2.

При выборе «Use data link file» и нажатии на кнопку «Browse…» появляется стандартный диалог выбора файла. Этот файл можно создать в любом окне explorer-а (в этом окне открытия файла, в самом explorer, на desktop и т.д.) вызвав контекстное меню и выбрав пункт «New/Microsoft Data Link». Потом вызовите локальное меню для созданного файла и выберите в нем пункт «Open». После этого появится property sheet описанный чуть ниже. Эти же вкладки содержит и property sheet, вызываемый через пункт «Property» локального меню UDL файла, но в нем еще есть вкладки относящиеся к самому файлу.

Использование файлов Microsoft Data Link упрощает поддержку приложений, так как возможно использовать средства Windows для настройки приложения.

При выборе в редакторе свойства «Use connection string» и нажатии на кнопку «Build…» появляется такой же property sheet, как и при выборе «Open» для Microsoft Data Link файла.

В этом окне выбирается тип базы данных, местоположение базы и параметры соединения.

На первой странице выбирается тип базы данных или Provider, в терминах ADO.

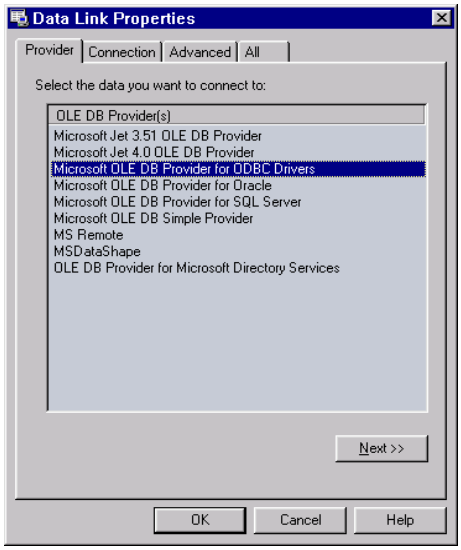


Рис. 6.3.

Базы доступны как через «Microsoft Jet OLE DB Provider», так и через «Microsoft OLE DB Provider for ODBC».

Следующая страница зависит от выбранного типа базы, однако для всех типов есть кнопка «Test connection» позволяющая проверить правильность и полноту параметров.

Для «Microsoft Jet OLE DB Provider» она выглядит так:

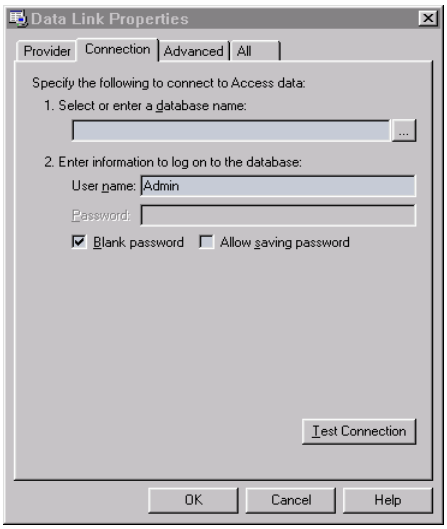


Рис. 6.4.

Checkbox «Blank password» подавляет диалог ввода идентификатора и пароля пользователя при установлении соединения, если поле пароля пустое.

Checkbox «Allow saving password» сохраняет пароль в строке параметров соединения. Если он не отмечен, то введенный пароль будет использоваться только при выполнении тестового соединения.

Для «Microsoft OLE DB Provider for ODBC» эта страница выглядит так:

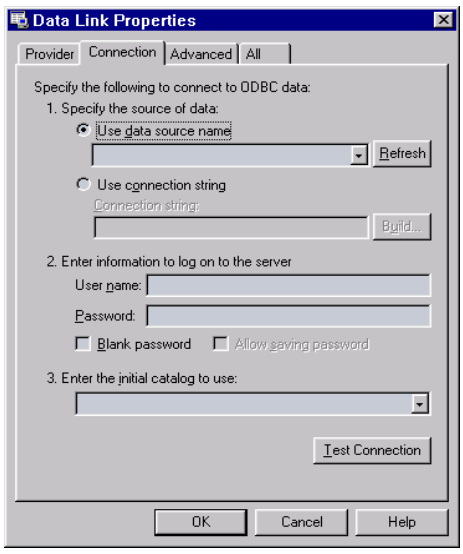


Рис. 6.5.

Радиокнопка «Use data source name» позволяет ввести предустановленный алиас ODBC, а через «Use connection string» вводится как алиасы так и тип ODBC драйвера и параметры соединения.

Параметры идентификации пользователя аналогичны выше описанным.

На странице «Advanced» расположены дополнительные параметры, с помощью которых устанавливается уровень доступа к файлу базы данных, таймаут сетевого соединения (то есть время через которое связь будет считаться потерянной, если сервер не отвечает) и уровень глубины проверки секретности соединения.

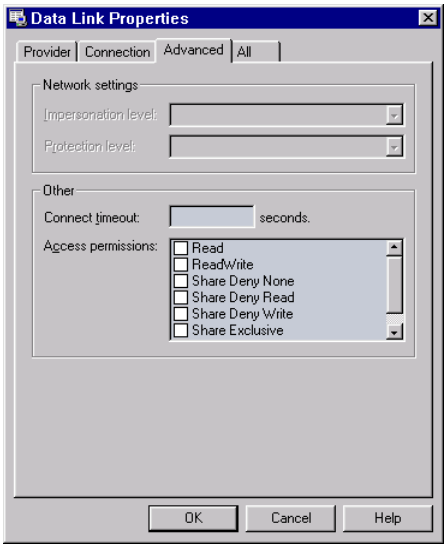


Рис. 6.6

На странице «All» можно отредактировать все параметры с предыдущих страниц и параметры зависящие от провайдера, но не вошедшие на страницу «Connection». Редактирование осуществляется в виде параметр –

значение, причем в текстовой форме, никаких диалогов нет. Помощи то же нет, эти параметры описаны только в документации на провайдер. Ее можно найти в MSDN Data Services/Microsoft Data Components (MDAC) SDK/Microsoft ActiveX Data Objects (ADO)/Microsoft ADO Programmer's Reference/Using Providers with ADO.

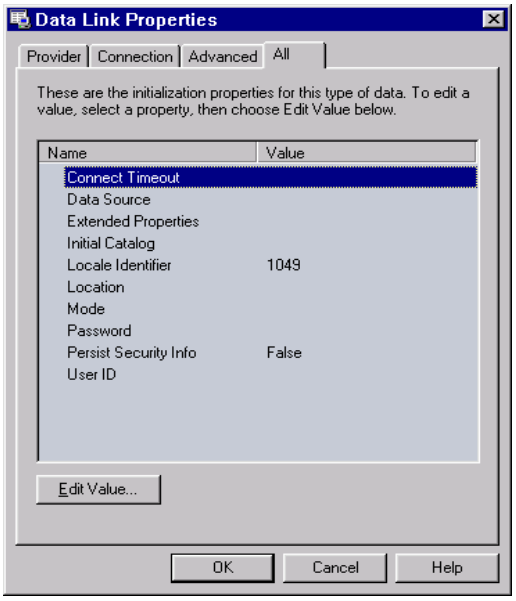


Рис. 6.7.

В компоненте **TADOConnection** есть свойства **Provider**, **DefaultDatabase** и **Mode** которые являются альтернативным методом задания частей строки параметров соединения – провайдера, базы данных (например, пути до базы) и режима совместного использования файлов базы данных. Эти значение этих свойств автоматически включаются в строку соединения, если были заданы до активизации компонента и автоматически выставляются после соединения.

Простейшее приложение

Создадим простейшее приложение, состоящее из одной таблицы. Создаем форму состоящую из трех компонент

TADOTable с палитры компонент ADO,

TDataSource с палитры компонент Data ,

TDBGrid с палитры компонент Data Controls.

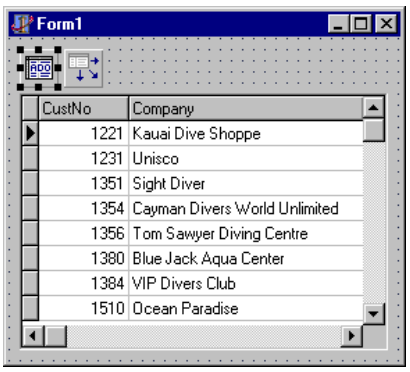


Рис 6.8

Связываем компоненты, устанавливая свойство **TDBGrid DataSource** на компонент **TDataSource**, свойство **DataSet** компонента **TDataSource** на компонент **TADOTable**.

Теперь нам необходимо указать базу данных. Делается это в свойстве **ConnectionString** компонента **TADOTable**. При нажатии на кнопку «…» появится редактор параметров соединения. Отметим радокнопку «Use data link file», нажмите на кнопку «Browse…» и выберите в появившемся окне после файл линка к базе данных «\Program Files\Common Files\System\ole db\Data Links\DBDEMOS.UDL». Этот линк указывает на базу в формате , входящую в поставку Delphi.

После этого в свойстве **TableName** компонента **TADOTable** выберем таблицу customer.

Активизируем компонента **TADOTable**, установив свойство **Active** в **True**.

Приложение можно запускать. Этот пример можно найти в директории Simple.

Обзор ADO

ADO основано на технологии COM. Все объекты и интерфейсы ADO являются интерфейсами и объектами COM.

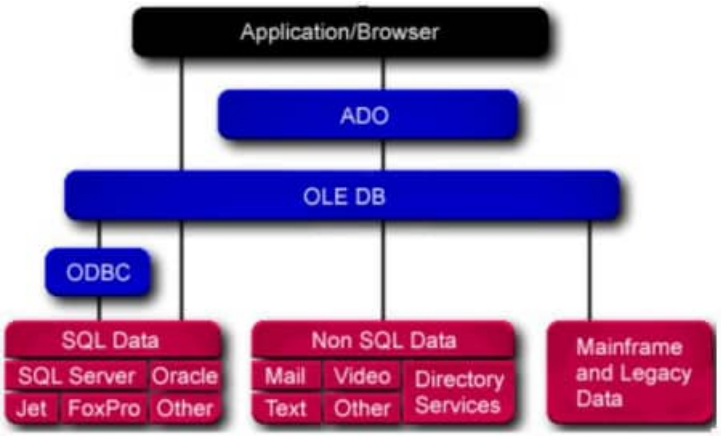


Рис 6.9. Архитектура ADO

Интерфейс **Connection**

Объекты этого типа выполняют следующие функции:

Связь с сервером.

Управление транзакциями.

Получение информации о произошедших ошибках (свойство **Errors**).

Получение информации о схеме данных (таблицы, поля и так далее).

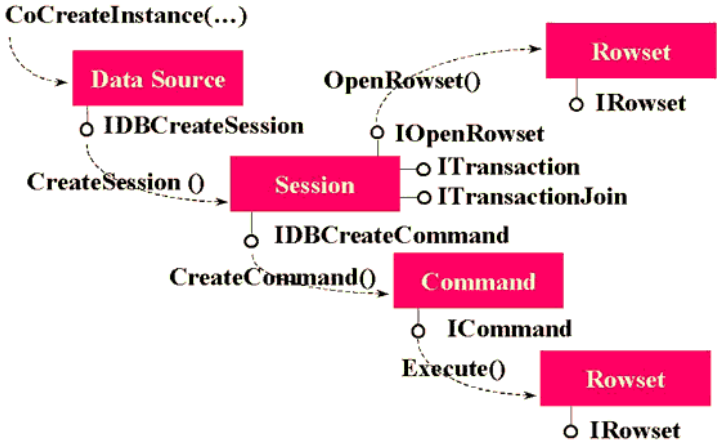


Рис 6.10. Схема взаимодействия в ADO основных COM интерфейсов

Интерфейсы **Recordset** и **Field**

Интерфейс **Recordset** (на нижнем уровне ADO это IRowset) является аналогом **TDataSet** в Delphi.

Поддерживает текущее положение и перемещение курсора, закладки (bookmarks), чтение, изменение и удаление записей и так далее. Значение полей и их типы доступны с помощью свойства **Fields.**

Интерфейс **Field** позволяет получать значение поля, его тип длину и так далее.

Интерфейсы **Command** и **Parameter**

Эти два типа позволяют работать с командами источника данных. Синтаксис команд для каждого из источников свой.

Интерфейс **Property**

Все объекты, кроме **Parameter**, имеют свойство **Properties**, которое позволяет получать и устанавливать параметры специфические для провайдера данных.

Библиотека довольно запутанная, многие функции дублированы в разных объектах. Например, **Recordset** можно создавать напрямую, методом **Open**, (причем предварительно создавать **Connection** не обязательно), можно получить как результат выполнения метода **Command.Execute**, либо после **Connection.Execute** задав команду без параметров.

Интерфейс **Command** инкапсулирован во все компоненты за исключением **TADOConnection**. Это сделано потому, что в ADO нет возможности получить данные не выполнив команду.

Интерфейс **Recordset** инкапсулирован в компоненты производные от **TCustomADODataSet.** Это компоненты **TADODataSet, TADOTable,**

TADOQuery, TADOStoredProc. Получать данные из них возможно штатными средствами Delphi.

Возможно получение данных и при выполнении компонента **TADOCommand**. Метод этого компонента **Execute** возвращает тип **_Recordset**. После чего его можно, например, связать с компонентом **TADODataSet** следующим образом

```
ADODataSet1.RecordSet := ADOCommand1.Execute;
```

Компоненты **TADOTable, TADOQuery** и **TADOStoredProc** являются частными случаями команды, соответственно для таблицы, SQL запроса и хранимой процедуры.

Тип **Connection** инкапсулируется в компонент **TADOConnection**.

Когда вы выполняете команду предварительно не открывая соединение, оно все равно создается. Получить к нему доступ возможно через свойство **Recordset**. Привязать компонент **TADOConnection** к уже открытому соединению возможно через свойство **ConnectionObject**.

Информацию о структуре базы данных можно получить с помощью метода **OpenSchema** компонента **TADOConnection**. Эта информация представлена как набор таблиц, как стандартизованных, так и специфических для провайдера. Таким способом можно узнать список таблиц, запросов, хранимых процедур и многое другое. Однако изменять структуру базы с помощью возвращаемых наборов данных невозможно.

Пример использования TADOConnection

В этом примере рассматривается работа с компонентом **TADOConnection**, SQL запросами с параметрами и трансакциями.

Создадим приложение из следующих компонентов

Connect типа **TADOConnection**
MasterSQL и **DetailSQL** типа **TADODataSet**
MasterDS и **DetailDS** типа **TDataSource**
MasterGrid и **DetailGrid** типа **TDBGrid**

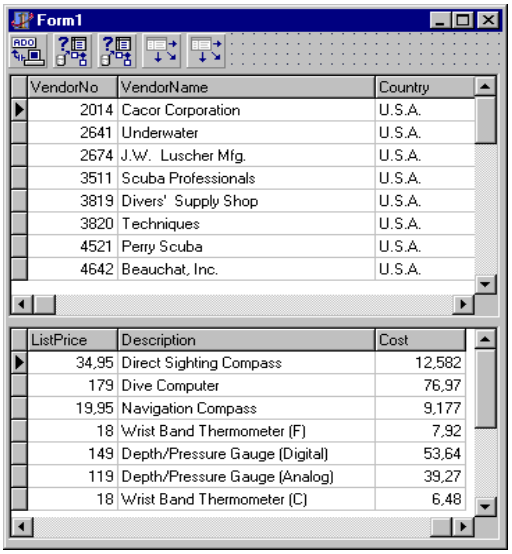


Рис 6.11. Master-detail форма на этапе дизайна

Связываем **MasterGrid, MasterDS, MasterSQL** и **DetailGrid, DetailDS, DetailSQL** аналогично предыдущему примеру, за исключением того, что вместо типа **TADOTable** используется тип **TADODataSet.**

Привязываем **Connect** к базе данных. Для этого в редакторе свойства **ConnectionString** выбираем ту же базу данных, что и в предыдущем примере.

Для ввода SQL запросов необходимо отредактировать свойство **CommandText** компонентах **MasterSQL** и **DetailSQL**. После нажатия на кнопку «…» появится редактор компонент, который выглядит следующим образом

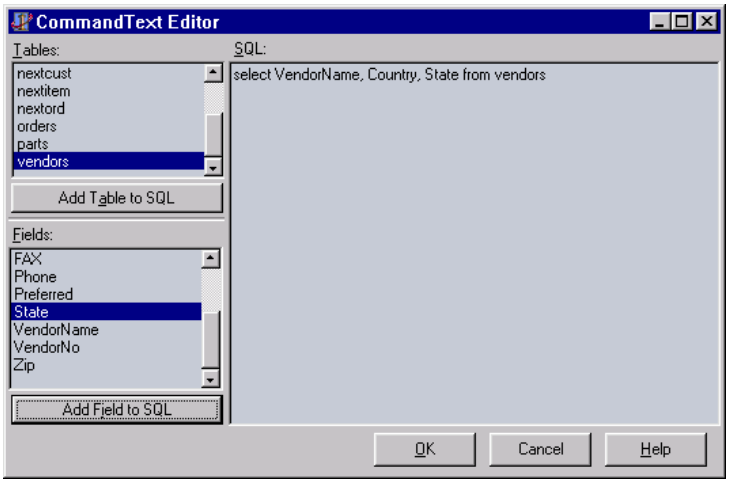


Рис. 6.12

Кнопка «Add Table to SQL» добавляет в текст SQL запроса таблицу, выбранную в списке «Tables», а «Add Field to SQL» поле таблицы, выбранное в списке «Fields».

Запрос для **MasterSQL**

```
select VendorNo, VendorName, Country, City, State, Preferred
from vendors
```

Запрос в **DetailSQL** должен выбирать только те детали, поставщик которых является текущим в **MasterSQL**. Для этого установим свойство **DataSource** компонента **DetailSQL** в значение **MasterDS.**

Запрос для **DetailSQL** следующий:

```
select PartNo, OnOrder, OnHand, ListPrice, Description, Cost
from parts
where VendorNo = :VendorNo
```

:VendorNo в части **where** – параметр запроса. Параметры при установленном **DataSource** берутся из него.

Активизируем **MasterSQL** и **DetailSQL** аналогично предыдущему примеру.

Приложение можно запускать. Этот пример можно найти в директории MasterDetail.

Пример использования параметров запроса

Теперь ограничим выборку поставщиков по значению поля **State.** Для этого добавим к форме следующие компоненты **StateEdit** типа **TEdit** с вкладки Standard, Query**Button** типа **TButton** с вкладки Standard

Изменим запрос в **MasterSQL** на

```
select VendorNo, VendorName, Country, City, State, Preferred
from vendors
where State = :StateID
```

:StateID – параметр, вместо которого при выполнении подставляется значение.

Добавим так же обработчик события **OnClick** в **QueryButton** следующего содержания

```
procedure TForm1.QueryButtonClick(Sender: TObject);
begin
  MasterSQL.Active := False;
  DetailSQL.Active := False;
  MasterSQL.Parameters.ParamByName('StateID').Value := StateEdit.Text;
  MasterSQL.Active := True;
  DetailSQL.Active := True;
end;
```

Программа готова. Этот пример можно найти в директории Param.

Синхронизация данных клиента и сервера

В ADO используются три метода синхронизации данных на клиенте и сервере.

Первый – с помощью метода **Resync**, который повторно считывает записи набора. Этот метод используется при выполнении метода **Refresh** Delphi.

Второй – повторный запрос методом **Requery**, который заново выполняет запрос на сервере. Выполнение этого метода то же самое, что и выполнение подряд закрытия и открытия набора данных.

Третий- уведомление сервером клиента в случае изменения данных.

Этих методы доступны во всех компонентах имеющих набор данных. Однако эти функции доступны не для всех баз данных.

Работа с транзакциями

В компонентах ADO работа с транзакциями осуществляется через компонент **TADOConnection.**

Тип транзакции устанавливается в свойстве **IsolationLevel** одной из следующих констант:

IlUnspecified	Сервер будет использовать лучший, по его мнению, тип изоляции.
IlChaos	Транзакции с более высоким уровнем изоляции не могут изменять данные измененные, но не подтвержденные в текущей транзакции.
IlReadUncommitted	Чтение данных измененных в не подтвержденных транзакций. То есть изменения видны сразу после того как другая транзакция передала их на сервер.
IlBrowse	То же самое что и **IlReadUncommitted**
IlReadCommitted	Чтение данных измененных подтвержденными транзакциями. То есть изменение данных будет видимо после выполнения Commit в другой транзакции.
IlCursorStability	То же самое что и **IlCursorStability**.
IlRepeatableRead	Изменения, сделанные другими транзакциями не видимы, но при выполнении перезапроса они транзакция может получать

	новый набор данных.
IlIsolated	Транзакция не видит изменений данных произведенных другими транзакциями.
IlSerializable	То же самое что и **IlIsolated**.

Обратите внимание на то, что не все типы провайдеров данных поддерживают все типы изоляции или работу с транзакциями.

Свойство **Attributes** устанавливает открывать ли новую транзакцию автоматически

xaCommitRetaining – при подтверждении транзакции

xaAbortRetaining – при отмене транзакции

Так же у компонента **TADOConnection** есть три метода для работы с транзакциями:

BeginTrans Начинает транзакцию

CommitTrans Подтверждает сделанные изменения

RollbackTrans Откатывает транзакцию.

Пример работы с транзакциями

За базовый возьмем пример использования **TADOConnection**. Добавим к форме две кнопки (**StCmButton** и **RollbackButton**) типа **TButton,** обработчики событий **OnClick** этих кнопок, процедуру **fix_controls** без параметров к форме, обработчик события **OnActivate** формы

Программу можно запускать. Этот пример находится в директории Trans.

Обратите внимание на то, что когда транзакция не запущена, все равно можно изменять данные и они записываются немедленно.

Доступ к данным

В отличие от BDE, ADO поддерживает больше настроек работы данных.

В ADO есть понятие набора данных (recordset) и тесно связанное с ним понятие курсора (cursor). Что такое курсор в документации на ADO не описано. Однако почему то месторасположение набора данных называется положением курсора. Я думаю, что это терминологическая путаница в Microsoft и курсор то же самое что набор данных.

Во всех компонентах имеющих набор данных (то есть в **TADODataSet, TADOTable, TADOQuery, TADOStoredProc**) есть свойства **CursorLocation**, **CursorType**, **LockType** и **MarshalOptions**, устанавливающие параметры обмена с сервером. Все эти свойства должны быть установлены до того, как набор данных открывается. Если вы установите их позже, то эффекта не будет.

CursorLocation – определяет, где выполняется работа с набором на клиенте (**clUseClient**) или на сервере (**clUseServer**). Если набор данных расположен на клиенте, то с сервера данные запрашиваются однократно (или до

```
type
TForm1 = class(TForm)
…
private
procedure fix_controls;
…
procedure TForm1.fix_controls;
begin
if Connection.InTransaction then
begin
StCmButton.Caption := 'Commit';
RollbackButton.Enabled := True;
end
else
begin
StCmButton.Caption := 'Begin';
RollbackButton.Enabled := False;
end;
MasterSQL.Requery;
DetailSQL.Requery;
end;

procedure TForm1.StCmButtonClick(Sender: TObject);
begin
if not Connection.InTransaction
then Connection.BeginTrans
else Connection.CommitTrans;
fix_controls;
end;

procedure TForm1.RollbackButtonClick(Sender: TObject);
begin
Connection.RollbackTrans;
fix_controls;
end;

procedure TForm1.FormActivate(Sender: TObject);
begin
fix_controls;
end;
```

выполнения повторного запроса), в дальнейшем вся выборка данных и позиционирование идет на клиенте. Однако модификация данных производится немедленно.

CursorType – устанавливает тип курсора. Значение одно из:

ctUnspecified – библиотека ADO сама определяет оптимальный тип блокировки.

ctStatic – статический курсор. Статическая копия набора записей, которую вы можете использовать, например, для генерации отчета. Добавления, изменения или удаление записей другими пользователями не видимы.

ctOpenForwardOnly – идентичен статическому курсору, за исключением того, что вы можете переходить только вперед. Это тип улучшает эффективность в ситуациях, когда вы делаете только один проход через набор данных.

ctDynamic – динамический курсор. Добавления, изменения и удаление другими пользователями видимы и возможны все типы передвижения по набору данных. Закладки (bookmarks) возможны только, если провайдер данных их поддерживает.

ctKeyset – курсор набора данных. Аналогичен динамическому курсору, за исключением того, что вы не увидите записи добавленные другими пользователями, а записи удаленные другими пользователями недоступны из вашего набора данных. Изменения данных другими пользователями видимы.

Надо заметить, что **TDBGrid** и другие компоненты, одновременно работающие с несколькими записями, могут работать только когда закладки поддерживаются. Поэтому для компонент с которыми вы будите связывать такие компоненты должны использоваться типы **ctKeyset** или **ctDynamic.**

LockType – определяет тип блокировки записей в наборе данных. Оно из:

ltUnspecified – библиотека ADO сама определяет какой тип будет использоваться.

ltReadOnly – только чтение, изменение данных невозможно.

ltPessimistic – пессимистическая блокировка. Запись блокируется сразу после начала редактирования и до сохранения записей.

ltOptimistic – оптимистическая блокировка. Запись блокируется только когда изменения сохраняются.

ltBatchOptimistic – тоже самое что и **ltOptimistic**, но используется отложенное сохранение изменений записей. Более подробно она рассматривается в следующем пункте.

MarshalOptions – это свойство определяет будут ли отправлены на сервер те поля, которые не были изменены. При значении **moMarshalAll** будут, а при **moMarshalModifiedOnly** не будут.

Работа с отложенными изменениями

Обратите внимание, что в компонентах ADO нет свойства **CachedUpdates**, но это не означает, что невозможно отложить передачу изменений данных на сервер. Эта возможность встроена с ADO и называется Batch Updates.

Для ее использования необходимо использовать клиентский курсор (то есть установить свойство **CursorLocation** в **clUseClient**) и **LockType** в **ltBatchOptimistic**

Так же есть метод сохраняющий изменения **UpdateBatch** и метод их отменяющий **CancelBatch.**

К каким записям из набора данных применяется действие зависит от единственного параметра этих функций

arCurrent – текущая запись
arFiltered – записи, которые попали в фильтрацию.
arAll – все записи набора.

Пример работы с отложенными изменениями

За основу возьмем пример работы с транзакциями.
Добавим компоненты
BatchCB типа **TCheckBox**
ApplyButton типа **TButton**
CancelButton типа **TButton**

Добавим обработчики событий **OnClick** во все эти три компонента.
Изменим обработчик события **OnActivate** формы.

События ADO

События ADO предназначены для той же цели, что и события VCL. Многие из них имеют аналогичные события VCL и компоненты вызывают из событий ADO события VCL. В компонентах доступны как события ADO, так и события BDE.

События соединения
OnConnectComplete – после установки соединения.
OnDisconnect – при разрыве соединения.

```
procedure TForm1.FormActivate(Sender: TObject);
begin
   fix_controls;
   ApplyButton.Visible  := BatchCB.State = cbChecked;
   CancelButton.Visible := BatchCB.State = cbChecked;
end;

procedure TForm1.BatchCBClick(Sender: TObject);
begin
   MasterSQL.Close;
   DetailSQL.Close;
   if BatchCB.State = cbChecked then
    begin
      MasterSQL.LockType := ltBatchOptimistic;
      DetailSQL.LockType := ltBatchOptimistic;
    end
   else
    begin
      MasterSQL.LockType := ltOptimistic;
      DetailSQL.LockType := ltOptimistic;
    end;
   MasterSQL.Open;
   DetailSQL.Open;
   ApplyButton.Visible  := BatchCB.State = cbChecked;
   CancelButton.Visible := BatchCB.State = cbChecked;
end;

procedure TForm1.ApplyButtonClick(Sender: TObject);
begin
   MasterSQL.UpdateBatch;
   DetailSQL.UpdateBatch;
end;

procedure TForm1.CancelButtonClick(Sender: TObject);
begin
   MasterSQL.CancelBatch;
   MasterSQL.CancelBatch;
end;
```

Эти события инкапсулированы в компоненте **TADOConnection**.

События транзакции.

OnBeginTransComplete – при выполнении **BeginTrans**.

OnCommitTransComplete – при выполнении **CommitTrans**.

OnRollbackTransComplete – при выполнении **RollbackTrans**.

Эти события инкапсулированы в компоненте **TADOConnection**.

События выполнения команд

OnWillExecute и **OnExecuteComplete** вызываются перед и после выполнением команды.

Эти события инкапсулированы в компоненте **TADOConnection**, а не в компоненте **TADOCommand**, как можно было бы предположить. Это связано с тем, что в ADO объекта команды как такого нет и по этой причине он не может получать сообщения.

В **TADOConnection** также инкапсулировано событие **OnInfoMessage,** которое вызывается при приходе с сервера дополнительной информации. Формат и назначение зависят от сервера, с которым вы работаете.

В ADO так же есть события связанные с набором данных, а не с соединением, как вышеописанные. Они инкапсулированы в компоненты имеющие набор данных – **TADODataSet**, **TADOTable**, **TADOQuery** и **TADOStoredProc**.

Эти события можно разбить на три группы.

События выборки данных

OnFetchProgress – многократно вызывается в процессе выборки набора данных.

OnFetchComplete – завершение выборки.

Уведомления об изменении положения текущей записи в наборе.

OnWillMove, **OnMoveComplete** – вызываются до и после изменения положения текущей записи. **OnWillMove** позволяет отменить действие.

OnEndOfRecordset – вызывается при достижении конца набора данных, позволяет добавить новую запись.

Уведомления об изменении набора данных

OnWillChangeField, **OnFieldChangeComplete** – до и после изменения текущей записи набора.

OnWillChangeRecord, **OnRecordChangeComplete** – вызываются до и после изменения, добавления, удаления строки набора и отмене этих действий.

OnWillChangeRecordset, **OnRecordsetChangeComplete** - вызываются до и после открытия, закрытия, повторного запроса и синхронизации набора данных.

Многие события допускают прерывание действия. Эта возможность бывает полезна при асинхронной работе с сервером. Например, для прерывания слишком длинного запроса.

Асинхронная работа с сервером

В ADO есть возможность не имеющая аналогов ни в BDE ни в InterBase. Это асинхронное выполнение операций с сервером. Могут асинхронно выполнятся установка соединения с сервером (Connection), выполнение команды (Execute) и выборка набора данных (Fetch)

Асинхронное соединение

Для включения этого режима необходимо установить свойство **ConnectOptions** компонента **TADOConnection** в **coAsyncConnect**.

При установлении соединения происходит

Вызывается обработчик события **OnWillConnect**

Управление передается в программу

После окончания соединения, как успешного, так и ошибочного, вызывается обработчик события **OnConnectComplete**

Асинхронное выполнение команды

Надо заметить, что все компоненты ADO, за исключением компонента **TADOConnection** при активизации или выполнении исполняют команду ADO. Эти компоненты **TADOCommand, TADODataSet, TADOTable, TADOQuery, TADOStoredProc**. Установите в свойстве **ExecuteOptions eoAsyncExecute**.

При исполнении происходит

Вызывается обработчик события **OnWillExecute**

Управление передается в программу

После окончания выполнения команды, как успешного, так и ошибочного, вызывается обработчик события **OnExecuteComplete**

Асинхронная выборка данных

Асинхронная выборка данных поддерживается в компонентах **TADODataSet, TADOTable, TADOQuery, TADOStoredProc**. Для ее включения установите в свойстве **ExecuteOptions eoAsyncFetch**.

После исполнения команды происходит передача данных. В процессе передачи многократно вызывается обработчик сообщения **OnFetchProgress**. В его параметрах есть как количество уже переданных записей, так и общее количество. Это очень удобно для создания индикаторов типа TProgressBar. После того, как выборка с сервера закончена, вызывается обработчик события **OnFetchComplete**.

2. ПРОГРАММА РАБОТЫ

1. Ознакомиться с теоретическими положениями.
2. Разработать на прикладном языке программирования произвольную таблицу с любой комбинацией атрибутов предыдущих лабораторных работ.
3. Значения атрибутов с помощью технологии ADO передать из реляционных баз данных, разработанных ранее, в построенную таблицу.
4. Предусмотреть в прикладной программе какую-либо математическую обработку данных в столбцах пользовательской таблице (например, среднее значение или сумма значений или т.п.)

Лабораторная работа № 12

ИСПОЛЬЗОВАНИЕ МЕХАНИЗМОВ ЗАЩИТЫ БАЗЫ ДАННЫХ

1. ЦЕЛЬ РАБОТЫ

Изучение методов и построение алгоритмов обеспечения защиты базы данных.

2. ОСНОВЫ ТЕОРИИ

Защита информации в базах данных, в отличие от защиты данных в файлах, имеет и свои особенности:

- необходимость учета функционирования системы управления базой данных при выборе механизмов защиты;
- разграничение доступа к информации реализуется не на уровне файлов, а на уровне частей баз данных.

При создании средств защиты информации в базах данных необходимо учитывать взаимодействие этих средств не только с ОС, но и с СУБД. При этом возможно встраивание механизмов защиты в СУБД или использование их в виде отдельных компонент. Для большинства СУБД придание им дополнительных функций возможно только на этапе разработки СУБД. В эксплуатируемые системы управления базами данных дополнительные компоненты могут быть внесены путем расширения или модификации языка управления. Таким путем можно осуществлять наращивание возможностей, например, в СУБД CA-Clipper 5.0.

В современных базах данных довольно успешно решаются задачи разграничения доступа, поддержания физической целостности и логической сохранности данных. Алгоритмы разграничения доступа к записям и даже к полям записей в соответствии с полномочиями пользователя хорошо отработаны, и преодолеть эту защиту злоумышленник может лишь с помощью фальсификации полномочий или внедрения вредительских программ. Разграничение доступа к файлам баз данных и к частям баз данных осуществляется СУБД путем установления полномочий пользователей и контроля этих полномочий при допуске к объектам доступа.

Полномочия пользователей устанавливаются администратором СУБД. Обычно стандартным идентификатором пользователя является пароль, передаваемый в зашифрованном виде. В распределенных КС процесс подтверждения подлинности пользователя дополняется специальной процедурой взаимной аутентификации удаленных процессов. Базы данных, содержащих конфиденциальную информацию, хранятся на внешних запоминающих устройствах в зашифрованном виде.

Физическая целостность баз данных достигается путем использования отказоустойчивых устройств, построенных, например, по технологии RAID. Логическая сохранность данных означает невозможность нарушения структуры моде-

ли данных. Современные СУБД обеспечивают такую логическую целостность и непротиворечивость на этапе описания модели данных.

В базах данных, работающих с конфиденциальной информацией, необходимо дополнительно использовать криптографические средства закрытия информации. Для этой цели используется шифрование как с помощью единого ключа, так и с помощью индивидуальных ключей пользователей. Применение шифрования с индивидуальными ключами повышает надежность механизма разграничения доступа, но существенно усложняет управление.

Возможны два режима работы с зашифрованными базами данных. Наиболее простым является такой порядок работы с закрытыми данными, при котором для выполнения запроса необходимый файл или часть файла расшифровывается на внешнем носителе, с открытой информацией производятся необходимые действия, после чего информация на ВЗУ снова зашифровывается. Достоинством такого режима является независимость функционирования средств шифрования и СУБД, которые работают последовательно друг за другом. В то же время сбой или отказ в системе может привести к тому, что на ВЗУ часть базы данных останется записанной в открытом виде.

Второй режим предполагает возможность выполнения СУБД запросов пользователей без расшифрования информации на ВЗУ. Поиск необходимых файлов, записей, полей, групп полей не требует расшифрования. Расшифрование производится в ОП непосредственно перед выполнением конкретных действий с данными. Такой режим возможен, если процедуры шифрования встроены в СУБД. При этом достигается высокий уровень защиты от несанкционированного доступа, но реализация режима связана с усложнением СУБД. Придание СУБД возможности поддержки такого режима работы осуществляется, как правило, на этапе разработки СУБД.

При построении защиты баз данных необходимо учитывать ряд специфических угроз безопасности информации, связанных с концентрацией в базах данных большого количества разнообразной информации, а также с возможностью использования сложных запросов обработки данных. К таким угрозам относятся:

- инференция;
- агрегирование;
- комбинация разрешенных запросов для получения закрытых данных.

Под **инференцией** понимается получение конфиденциальной информации из сведений с меньшей степенью конфиденциальности путем умозаключений. Если учитывать, что в базах данных хранится информация, полученная из различных источников в разное время, отличающаяся степенью обобщенности, то аналитик может получить конфиденциальные сведения путем сравнения, дополнения и фильтрации данных, к которым он допущен. Кроме того, он обрабатывает информацию, полученную из открытых баз данных, средств массовой информации, а также использует просчеты лиц, определяющих степень важности и конфиденциальности отдельных явлений, процессов, фактов, полученных результатов. Такой способ получения конфиденциальных сведений, например, по материалам средств массовой информации, используется давно, и показал свою эффективность.

Близким к инференции является другой способ добывания конфиденциальных сведений - **агрегирование**. Под агрегированием понимается способ получения более важных сведений по сравнению с важностью тех отдельно взятых данных, на основе которых и получаются эти сведения. Так, сведения о деятельности одного отделения или филиала корпорации обладают определенным весом. Данные же за всю корпорацию имеют куда большую значимость.

Если инференция и агрегирование являются способами добывания информации, которые применяются не только в отношении баз данных, то способ специального **комбинирования запросов** используется только при работе с базами данных. Использование сложных, а также последовательности простых логически связанных запросов позволяет получать данные, к которым доступ пользователю закрыт. Такая возможность имеется, прежде всего, в базах данных, позволяющих получать статистические данные. При этом отдельные записи, поля, (индивидуальные данные) являются закрытыми. В результате запроса, в котором могут использоваться логические операции AND, OR, NOT, пользователь может получить такие величины как количество записей, сумма, максимальное или минимальное значение. Используя сложные перекрестные запросы и имеющуюся в его распоряжении дополнительную информацию об особенностях интересующей записи (поля), злоумышленник путем последовательной фильтрации записей может получить доступ к нужной записи (полю).

Противодействие подобным угрозам осуществляется следующими методами:

- блокировка ответа при неправильном числе запросов;
- искажение ответа путем округления и другой преднамеренной коррекции данных;
- разделение баз данных;
- случайный выбор записи для обработки;
- контекстно-ориентированная защита;
- контроль поступающих запросов.

Метод **блокировки ответа** при неправильном числе запросов предполагает отказ в выполнении запроса, если в нем содержится больше определенного числа совпадающих записей из предыдущих запросов. Таким образом, данный метод обеспечивает выполнение принципа минимальной взаимосвязи вопросов. Этот метод сложен в реализации, так как необходимо запоминать и сравнивать все предыдущие запросы.

Метод **коррекции** заключается в незначительном изменении точного ответа на запрос пользователя. Для того, чтобы сохранить приемлемую точность статистической информации, применяется так называемый свопинг данных. Сущность его заключается во взаимном обмене значений полей записи, в результате чего все статистики i-го порядка, включающие i атрибутов, оказываются защищенными для всех i, меньших или равных некоторому числу. Если злоумышленник сможет выявить некоторые данные, то он не сможет определить, к какой конкретно записи они относятся.

Применяется также метод **разделения баз данных** на группы. В каждую группу может быть включено не более определенного числа записей. Запросы разрешены к любому множеству групп, но запрещаются к подмножеству записей из

одной группы. Применение этого метода ограничивает возможности выделения данных злоумышленником на уровне не ниже группы записей. Метод разделения баз данных не нашел широкого применения из-за сложности получения статистических данных, обновления и реструктуризации данных.
Эффективным методом противодействия исследованию баз данных является метод **случайного выбора записей** для статистической обработки. Такая организация выбора записей не позволяет злоумышленнику проследить множество запросов.
Сущность **контекстно-ориентированной защиты** заключается в назначении атрибутов доступа (чтение, вставка, удаление, обновление, управление и т. д.) элементам базы данных (записям, полям, группам полей) в зависимости от предыдущих запросов пользователя. Например, пусть пользователю доступны в отдельных запросах поля: "идентификационные номера" и "фамилии сотрудников", а также "идентификационные номера" и "размер заработной платы". Сопоставив ответы по этим запросам, пользователь может получить закрытую информацию о заработной плате конкретных работников. Для исключения такой возможности пользователю следует запретить доступ к полю "идентификатор сотрудника" во втором запросе, если он уже выполнил первый запрос.
Одним из наиболее эффективных методов защиты информации в базах данных является **контроль поступающих запросов** на наличие "подозрительных" запросов или комбинации запросов. Анализ подобных попыток позволяет выявить возможные каналы получения несанкционированного доступа к закрытым данным.

3. ОБЪЕКТЫ И СРЕДСТВА ИССЛЕДОВАНИЯ

Предметная область в соответствии с вариантом задания. ПК типа IBM PC, дисплейный класс, операционная система Windows XP и выше.

4. ПОДГОТОВКА К РАБОТЕ

Ознакомиться с теоретическими положениями лабораторной работы.

5. ПРОГРАММА РАБОТЫ

По предложенному преподавателем варианту разработать алгоритм, моделирующий механизм защиты информации в базе данных. Оформить отчет.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

6.1. Назовите основные подходы к защите баз данных ?
6.2. Основные методы защиты?

Лабораторная работа № 13
УПРАВЛЕНИЕ ПРИВИЛЕГИЯМИ ДОСТУПА К БАЗАМ ДАННЫХ

1. ЦЕЛЬ РАБОТЫ

Изучение методов управления привилегиями доступа к базам данных.

2. ОСНОВЫ ТЕОРИИ

Привилегии пользователей назначаются им администратором базы данных и определяют какие действия над данными и над объектами схемы являются разрешенными. При контроле привилегий используется имя пользователя базы данных, называемое иногда идентификатором авторизации (Autorization ID). Некоторые СУБД идентифицируют понятие "пользователь" с понятием "учетная запись".

Все объекты пользователя БД входят в его схему. На практике один пользователь, как правило, ассоциируется с одной схемой, хотя стандарт подразумевает, что одному пользователю может принадлежать несколько схем, содержащих взаимосвязанные объекты.

После успешного завершения процедуры идентификации открывается сеанс пользователя и устанавливается соединение с базой данных.

Существуют привилегии двух типов:

- системные привилегии (system privileges), контролирующие общий доступ к базе данных;
- объектные привилегии (object privileges), контролирующие доступ к конкретным объектам базы данных.

Синтаксис, используемый для работы с привилегиями, на практике значительно шире стандарта, но в значительной степени зависит от архитектуры конкретной БД.

Для управления привилегиями определены следующие правила:

- объект принадлежит пользователю, его создавшему (если синтаксисом не указано создание объекта другого пользователя, конечно, при соответствующих полномочиях);
- владелец объекта, согласно стандарту, может изменять привилегии своего объекта (в коммерческих СУБД, таких как Oracle, уровни полномочий представляют собой более сложную иерархию);
- объектная привилегия всегда соотносится с конкретным объектом, а системная - с объектами вообще.

Язык SQL поддерживает следующие привилегии:

- ALTER - позволяет выполнять оператор ALTER TABLE;

- SELECT - позволяет выполнять оператор запроса;
- INSERT - позволяет выполнять добавление строк в таблицу;
- UPDATE - позволяет изменять значения во всей таблице или только в некоторых столбцах;
- DELETE - позволяет удалять строки из таблицы;
- REFERENCES - позволяет устанавливать внешний ключ с использованием в качестве родительского ключа любых столбцов таблицы или только некоторых из них;
- INDEX - позволяет создавать индексы (не входит в стандарт SQL-92);
- DROP - позволяет удалять таблицу из схемы базы данных.

Предоставление и снятие привилегий

Предоставление привилегии выполняется SQL-оператором GRANT, который имеет в стандарте SQL-92 следующее формальное описание:

```
ON { [TABLE] table_name
     | DOMAIN domain_name
     | COLLATION collation_name
     | CHARACTER SET set_name
     | TRANSLATION translation_name }
TO { user_name .,: } | PUBLIC
[ WITH GRANT OPTION ]
```

где privilege определяется как

```
{ ALL PRIVELEGES }
| SELECT
| DELETE
| INSERT [(field .,:)]
| UPDATE [(field .,:)]
| REFERENCES [(field .,:)]
| USAGE
```

После фразы GRANT через запятую можно перечислить список всех назначаемых привилегий.

Фраза ON определяет объект, для которого устанавливается привилегия.

Фраза TO указывает пользователя или пользователей, для которых устанавливается привилегия.

Так, оператор GRANT SELECT ON tbl1 TO PUBLIC; предоставляет доступ к выполнению оператора SELECT для таблицы tbl1 не только всем существующим пользователям, но и тем, которые позднее будут добавлены в базу данных.

Оператор GRANT UPDATE ON tbl1 TO user1; предоставляет пользователю user1 привилегию UPDATE на всю таблицу, а оператор GRANT UPDATE (f1,f2) ON tbl1 TO user1 предоставляет привилегию UPDATE для изменения только столбцов f1 и f2.

Фраза WITH GRANT OPTION предоставляет получающему привилегию пользователю дополнительную привилегию GRANT OPTION,позволяющую выполнять передачу полученных привилегий.

Отмена привилегии выполняется SQL-оператором REVOKE, который имеет в стандарте SQL-92 следующее формальное описание:

```
REVOKE [ GRANT OPTION FOR ]
   { ALL PRIVILEGES } | privilege
   ON { [TABLE] table_name
      | DOMAIN domain_name
```

```
    | COLLATION collation_name
    | CHARACTER SET set_name
    | TRANSLATION translation_name }
  FROM { PUBLIC | user_name .,: }
  [ CASCADE | RESTRICT ]
```

После фразы REVOKE через запятую можно перечислить список всех отменяемых привилегий.

Фраза ON определяет объект, для которого отменяется привилегия.

Фраза FROM указывает пользователя или пользователей, для которых отменяется привилегия.

Фраза GRANT OPTION FOR определяет отмену не самих привилегий, а только права их передачи другим пользователям.

Если одна привилегия вместе с опцией WITH GRANT OPTION была последовательно передана от одного пользователя другому несколько раз, то образуется цепочка зависимых привилегий. Фразы CASCADE и RESTRICT определяют, что будет происходить с этими привилегиями при отмене одного из звеньев этой цепочки.

Если при отмене зависимой привилегии для объекта не остается ни одной существующей привилегии, то такой объект называется несостоявшимся. Например, подобное может произойти с представлением, созданным как запрос к таблице, привилегия на которую была утрачена.

Если при отмене привилегии появляется несостоявшийся объект, то фраза RESTRICT предотвратит выполнение оператора REVOKE, и никакие привилегии отменены не будут.

Если указана фраза CASCADE и при отмене привилегии появляется несостоявшийся объект, то все несостоявшиеся объекты (представления) удаляются, а при наличии несостоявшихся ограничений в таблицах они отменяются автоматически выполнением оператора ALTER TABLE несостоявшиеся ограничения в доменах отменяются автоматически выполнением оператора ALTER DOMAIN.

Роли

Ролью называется именованный набор привилегий. Объединение привилегий в роли значительно упрощает процесс назначения и снятия привилегий. Если СУБД поддерживает управление ролями, то в SQL-операторах GRANT и REVOKE вместо имени пользователя можно указывать имя роли.

Многоуровневый контроль доступа в БД Oracle

Среди современных коммерческих СУБД базу данных Oracle можно считать одной из самых продвинутых в области контроля доступа. Все привилегии делятся на системные и объектные.

Синтаксис оператора GRANT, выполняющего предоставление пользователям или ролям системных полномочий и ролей, может быть представлен следующей схемой:

предоставление пользователям или ролям системных полномочий и ролей,

Предоставление пользователям или ролям привилегий над обычными объектами БД Oracle также может быть показано на примере следующей схемы:

Предоставление пользователям или ролям привилегий над обычными объектами БД Oracle

system_priv - предоставляемое системное полномочие.
role - предоставляемая роль.
TO -определяет пользователей или роли, которым предоставляются системные или объектные полномочия.
PUBLIC - указывает, что системные или объектные полномочия, определяемые оператором, предоставляются всем пользователям.
WITH ADMIN OPTION - позволяет пользователю, получившему системные или объектные полномочия или роль, предоставлять их в дальнейшем другим пользователям или ролям. Такое разрешение включает и возможность изменения или удаления роли. (Синтаксически данная опция отличается от стандарта SQL-92.)
object_priv - определяет предоставляемую привилегию, которая может быть указана одним из следующих значений:

- ALTER
- DELETE
- EXECUTE (только для процедур, функций и пакетов)
- INDEX (только для таблиц)
- INSERT
- REFERENCES (только для таблиц)
- SELECT
- UPDATE.

column - определяет столбец таблицы или представления, на который распространяется предоставляемая привилегия.
ON - определяет объект (таблицу, вид, хранимую процедуру, снимок), на который предоставляется привилегия.

Например:

```
GRANT SELECT, UPDATE ON tbl1 TO PUBLIC
GRANT REFERENCES  (f1), UPDATE (f1, f2, f3)
    ON user1.tbl1 TO user2
```

Приведем список предоставляемых оператором GRANT системных полномочий, который характеризует систему контроля доступа БД Oracle. К системным полномочиям относятся следующие:
ALTER ANY CLUSTER - разрешает получившему эти полномочия изменение любого кластера в любой схеме;
ALTER ANY INDEX - разрешает изменение любого индекса в любой схеме;
ALTER ANY PROCEDURE - разрешает изменение любой хранимой функции, процедуры или пакета в любой схеме;
ALTER ANY ROLE - разрешает изменение в базе данных любой роли;
ALTER ANY SEQUENCE - разрешает изменение в базе данных любой последовательности;
ALTER ANY SNAPSHOT - разрешает изменение в базе данных любого снимка;
ALTER ANY TABLE - разрешает изменение в схеме любой таблицы или вида;
ALTER ANY TRIGGER - позволяет разрешать, запрещать или компилировать любой триггер базы данных в любой схеме; изменение в базе данных любой роли;
ALTER DATABASE- разрешает изменение базы данных;
ALTER PROFILE - разрешает изменение профилей;
ALTER RESOURCE COST - разрешает устанавливать цену ресурсов сеанса работы пользователя;
ALTER ROLLBACK SEGMENT - разрешает изменение сегментов отката;
ALTER SESSION - разрешает выполнение оператора ALTER SESSION;

ALTER SYSTEM - разрешает выполнение оператора ALTER SYSTEM;
ALTER TABLESPACE - разрешает изменение табличных пространств;
ALTER USER - разрешает изменение параметров для любого пользователя (пароль, количество доступного табличного пространства, назначенный профиль и т.п.);
ANALYZE ANY - разрешает анализировать таблицу, кластер или индекс в любой схеме;
AUDIT ANY - разрешает выполнять аудит любого объекта в любой схеме;
AUDIT SYSTEM - разрешает выполнение SQL-оператора AUDIT;
BACKUP ANY TABLE - позволяет выполнять экспорт объектов из схемы других пользователей;
BECOME USER - позволяет становиться другим пользователем (требуется при импорте БД);
COMMENT ANY TABLE - разрешает получившему эти полномочия комментарий для любой таблицы, вида или столбца в любой схеме;
CREATE ANY CLUSTER ;
CREATE ANY INDEX ;
CREATE ANY LIBRARY ;
CREATE ANY PROCEDURE ;
CREATE ANY SEQUENCE ;
CREATE ANY SNAPSHOT ;
CREATE ANY SYNONYM ;
CREATE ANY TABLE ;
CREATE ANY TRIGGER ;
CREATE ANY VIEW ;
CREATE CLUSTER - разрешает создавать кластер в своей схеме (системное полномочие, не содержащее в названии фразу ANY, распространяется только на собственную схему пользователя);
CREATE DATABASE LINK - разрешает создавать линк базы данных в своей схеме;
CREATE PROCEDURE ;
CREATE PROFILE ;
CREATE PUBLIC DATABASE LINK - разрешает создавать общедоступные линки базы данных;
CREATE PUBLIC SYNONYM ;
CREATE ROLE - разрешает создание ролей;
CREATE ROLLBACK SEGMENT ;
CREATE LIBRARY ;
CREATE SEQUENCE;
CREATE SESSION - разрешает соединение с базой данных;
CREATE SNAPSHOT;
CREATE SYNONYM;
CREATE TABLE;
CREATE TABLESPACE ;
CREATE TRIGGER;
CREATE USER ;
CREATE VIEW ;
DELETE ANY TABLE ;
DROP ANY CLUSTER ;
DROP ANY INDEX - разрешает удаление любого индекса;
DROP ANY LIBRARY ;
DROP ANY PROCEDURE ;
DROP ANY ROLE ;

DROP ANY SEQUENCE ;
DROP ANY SNAPSHOT ;
DROP ANY SYNONYM ;
DROP ANY TABLE ;
DROP ANY TRIGGER ;
DROP ANY VIEW ;
DROP LIBRARY ;
DROP PROFILE ;
DROP PUBLIC DATABASE LINK ;
DROP PUBLIC SYNONYM ;
DROP ROLLBACK SEGMENT ;
DROP TABLESPACE ;
DROP USER ;
EXECUTE ANY PROCEDURE - разрешает выполнение процедур и функций, а также ссылки на общедоступные переменные пакетов в любой схеме;
FORCE ANY TRANSACTION - позволяет выполнять фиксацию или откат любой сомнительной распределенной транзакции на локальной базе данных, а также определять сбой распределенной транзакции;
FORCE TRANSACTION - позволяет выполнять фиксацию или откат собственной сомнительной распределенной транзакции на локальной базе данных;
GRANT ANY PRIVILEGE ;
GRANT ANY ROLE ;
INSERT ANY TABLE ;
LOCK ANY TABLE ;
MANAGE TABLESPACE - разрешает переключение табличного пространства из автономного режима в оперативный или обратно, а также разрешает выполнять копирование табличного пространства;
RESTRICTED SESSION ;
SELECT ANY SEQUENCE ;
SELECT ANY TABLE ;
UNLIMITED TABLESPACE - разрешает неограниченное использование любого табличного пространства.Предоставление этого полномочия перекрывает любые ограничения на количество доступного табличного пространства, ранее установленные для пользователя;
UPDATE ANY TABLE - разрешает изменение строк в таблицах и видах любой схемы.

При создании базы данных Oracle некоторые роли создаются автоматически. Следующая таблица содержит названия автоматически создаваемых Oracle ролей и список предоставляемых ими системных полномочий. Эти роли создают иерархию предоставляемых полномочий.

Роль	Предоставляемые системные полномочия и роли
CONNECT	ALTER SESSION
	CREATE CLUSTER
	CREATE DATABASE LINK
	CREATE SEQUENCE
	CREATE SESSION
	CREATE SYNONYM
	CREATE TABLE
	CREATE VIEW

RESOURCE	CREATE CLUSTER
	CREATE PROCEDURE
	CREATE SEQUENCE
	CREATE TABLE
	CREATE TRIGGER
DBA	Все системные полномочия WITH ADMIN OPTION
	EXP_FULL_DATABASE (роль)
	IMP_FULL_DATABASE (роль)
EXP_FULL_DATABASE	SELECT ANY TABLE
	BACKUP ANY TABLE
	INSERT, UPDATE, DELETE
	ON sys.incexp, sys.incvid, sys.incfil

Для просмотра предоставленных привилегий администратор базы данных может использовать следующие системные представления словаря данных:

Системное представление	**Описание**
ALL_COL_PRIVS	Содержит список привилегий, предоставленных для столбцов таблицы другим пользователям или PUBLIC. ;
	Содержит столбцы:
	GRANTOR (кто предоставляет привилегию)
	GRANTEE (кому предоставляется привилегия)
	TABLE_SCHEMA (схема объекта)
	TABLE_NAME
	COLUMN_NAME
	PRIVILEGE
ALL_COL_PRIVS_MADE	Содержит список привилегий столбцов, которыми владеет пользователь или предоставляет на них привилегии.
	Содержит столбцы:
	GRANTEE
	OWNER
	GRANTOR
	TABLE_NAME
	COLUMN_NAME
	PRIVILEGE
ALL_TAB_PRIVS	Содержит список привилегий, предоставленных для таблицы другим пользователям или PUBLIC.
	Содержит столбцы:
	GRANTOR
	GRANTEE
	TABLE_NAMEPRIVILEGE
DBA_PROFILES	Содержит описания всех профилей базы данных и определяемых ими ограничений.
	Содержит столбцы:
	PROFILE
	RESOURCE_NAME

	LIMIT
DBA_ROLES	Содержит список имен всех существующих в базе данных ролей.
	Содержит столбцы:
	ROLE
	PASSWORD_REQUIRED
DBA_ROLE_PRIVS	Содержит список ролей, предоставляемых другим пользователям или ролям.
	Содержит столбцы:
	GRANTEE (кто получает полномочия)
	GRANTED_ROLE (предоставляемая роль)
DBA_SYS_PRIVS	Содержит список системных полномочий, предоставленных пользователям и ролям.
	Содержит столбцы:
	GRANTEE (кто получает полномочия)
	PRIVILEGE (название системного полномочия)
	ADMIN_OPTION
DBA_TAB_PRIVS	Содержит список всех предоставленных полномочий для объектов базы данных.
	Содержит столбцы:
	GRANTEE
	OWNER
	TABLE_NAME
	GRANTOR PRIVILEGE
DBA_USERS	Содержит информацию обо всех пользователях базы данных.
	Содержит столбцы:
	USERNAME
	USER_ID
	PASSWORD
	DEFAULT_TABLESPACE
	PROFILE
ROLE_SYS_PRIVS	Содержит информацию о предоставленных ролям системных полномочиях.
	Содержит столбцы:
	ROLE
	PRIVILEGE
ROLE_TAB_PRIVS	Содержит информацию о предоставленных ролям привилегиях для таблиц и столбцов.
	Содержит столбцы:
	ROLE
	OWNER
	TABLE_NAME
	COLUMN_NAME
	PRIVILEGE
SYSTEM_PRIVILEGE_MAP	Содержит список всех системных полномочий
TABLE_PRIVILEGE_MAP	Содержит информацию о кодах привилегий доступа к таблицам и столбцам.

USER_ROLE_PRIVS	Содержит список ролей, предоставленных пользователю.
	Содержит столбцы:
	USERNAME
	GRANTED_ROLE
USER_SYS_PRIVS	Содержит список всех системных полномочий, предоставленных пользователю.
	Содержит столбцы:
	USERNAME
	PRIVILEGE
USER_TAB_PRIVS	Содержит список привилегий для объектов, где пользователь является владельцем, получателем или лицом, предоставляющим привилегии.
	Содержит столбцы:
	GRANTEE (кому предоставляется привилегия)
	OWNER TABLE_NAME (имя объекта)
	GRANTOR (кто предоставляет привилегию)
USER_TAB_PRIVS_MADE	Содержит список всех предоставлений привилегий для объектов, принадлежащих пользователю.
	Содержит столбцы:
	GRANTEE
	GRANTOR
	TABLE_NAME
	PRIVILEGE
USER_TAB_PRIVS_RECD	Содержит список всех привилегий для объектов, где пользователь является получателем привилегии.
	Содержит столбцы:
	OWNER (владелец объекта)
	TABLE_NAME (имя объекта)
	GRANTOR (имя пользователя, предоставившего привилегию)
	PRIVILEGE

Эти системные представления позволяют администратору БД Oracle полностью контролировать назначение, передачу и взаимозависимость системных и объектных привилегий.

3. ОБЪЕКТЫ И СРЕДСТВА ИССЛЕДОВАНИЯ

Предметная область в соответствии с вариантом задания. ПК типа IBM PC, дисплейный класс, операционная система Windows XP и выше.

4. ПОДГОТОВКА К РАБОТЕ

Ознакомиться с теоретическими положениями лабораторной работы.

5. ПРОГРАММА РАБОТЫ

По предложенному преподавателем варианту разработать алгоритм, моделирующий механизм защиты информации в базе данных. Оформить отчет.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

6.1. Назовите основные подходы к защите баз данных?
6.2. Основные методы защиты?

Лабораторная работа № 14

АУДИТ ДАННЫХ С ПОМОЩЬЮ СРЕДСТВ СУБД

1. ЦЕЛЬ РАБОТЫ

Изучение методов аудита данных.

2. ОСНОВЫ ТЕОРИИ

В большинстве современных информационных систем протоколирование операций пользователей решается с помощью журнала событий. Однако бывают события, которые в таком журнале не отражаются, но способны нарушить целостность критически важной информации.

Большинство информационных систем строятся по схеме «клиент – сервер приложений – база данных», причем задача протоколирования выполняемых пользователями операций решается на уровне сервера приложений. Для этого, например, в прикладной системе может быть реализован служебный компонент для ведения журнала событий (операций), в котором регистрируются все значимые действия по просмотру и модификации информации. Преимущество такого подхода в том, что он позволяет регистрировать события в терминах предметной области, а не на уровне SQL-операций над таблицами базы данных. Казалось бы, подобный компонент, фиксирующий действия пользователей, решает все проблемы мониторинга операций и нет необходимости включения дополнительных средств на уровне СУБД, однако это не так. В крупных компаниях и организациях, имеющих большое количество взаимосвязанных систем, возникает острая необходимость контролировать критические изменения в данных не только по журналу событий, отражающих действия пользователя, но и на уровне операций, производимых в базе данных посредством SQL-команд или путем загрузки данных из разных источников.

Мониторинг на уровне СУБД

У любой прикладной системы обычно имеется администратор, сопровождающий ее с помощью SQL-интерфейса, и его действия, как правило, в журнале событий прикладной системы не отражаются. В результате на уровне приложения не остается никаких следов, позволяющих обнаружить возможные несанкционированные действия. Кроме этого, в самой прикладной системе могут быть уязвимости, позволяющие ее взломать, например, при помощи популярного среди хакеров метода SQL-инъекций, что может привести к непредсказуемым изменениям в базе, которые не будут зафиксированы в журнале событий. Наконец, часто возникает необходимость проведения прямых изменений в базе посредством SQL-команд, минуя интерфейс прикладной системы. Например, в задачу некоторых прикладных систем входит ежедневное выполнение расчетов определенных показателей (финансовых обязательств, биржевых котировок, и т. п.) на основе нормативных или оперативных данных, загружаемых из внешних источников. Бывают ситуации, когда перед расчетом выясняется, что поступившие в систему данные некорректны, а времени на их повторную загрузку уже нет, и в этом случае ошибки обычно исправляются вручную путем прямых изменений в базе. Очевидно, что такие действия необходимо контролировать, поскольку велика вероятность того, что исправление будет не-

верным. При этом важно отслеживать, что все изменения произведены до запуска расчета, а не в процессе его выполнения или окончания.

Все эти примеры свидетельствуют о том, что журнала событий прикладной системы недостаточно — необходим аудит изменений данных на уровне базы. Несмотря на инструментальную роль, отводимую базам данных в архитектуре современных информационных систем, в условиях, когда подавляющее большинство пользователей не употребляют SQL и вообще ничего не знают о схеме базы и структуре таблиц, с которыми они работают, применение средств мониторинга изменений на уровне базы оказывается полезным дополнительным инструментом для обеспечения безопасности и исключения ошибок при эксплуатации системы. Основным критерием для выбора механизма, обеспечивающего мониторинг, является минимальное влияние внедряемого решения на производительность системы.

Мониторинг с помощью триггеров

Наиболее известным способом мониторинга изменений является использование механизма триггеров, с помощью которых можно осуществлять перехват и протоколирование операций, производимых в базе, — модификацию данных (операции DML (Data Manipulation Language): insert, update, delete) и модификацию схемы (операции DDL (Data Definition Language): create table, alter table add column и т. п.). Этот метод также действует во встроенных механизмах аудита современных СУБД (например, при использовании команды AUDIT в Oracle), где система в автоматическом режиме создает и актуализирует набор триггеров, соответствующий текущим правилам контроля объектов базы, типов операций и т. п.

Основным недостатком применения триггеров и встроенного аудита для мониторинга изменений является падение производительности системы, поскольку в этом случае каждая операция в транзакции дополнительно сопровождается добавлением записи в таблицу аудита. Реально использовать встроенный аудит СУБД возможно только в том случае, когда удается заранее установить селективные условия аудита (например, настроить фильтр на протоколирование данных из небольшого подмножества таблиц базы, чтобы обеспечить обозримый объем выборки записей для анализа). Если же подключить к аудиту все операции над базой или их большую часть, то это сильно повлияет на производительность прикладной системы, и выполнять подобные операции на «боевой» системе, находящейся под постоянной или временами пиковой нагрузкой, обычно неприемлемо.

Мониторинг по журналу транзакций

С учетом названных ограничений получил развитие альтернативный подход к реализации мониторинга изменений, основанный на возможности извлечения и последующей обработки информации об операциях DML и DDL, имеющейся в журнале транзакций базы. Для основных СУБД (Oracle, Microsoft SQL Server и IBM DB2) созданы промышленные системы аудита, предлагаемые в виде отдельных продуктов: Oracle Audit Vault, Apex SQL Log и IBM Audit Management Expert. Эти системы обладают возможностями захвата изменений из журналов транзакций и позволяют аудитору удаленно подключаться к журналам транзакций целевых баз (включая архивы журналов), задавать разнообразные фильтры на выборку интересующих его записей, получать удобные для анализа отчеты в различных форматах и пр. Общим недостатком этих систем, несмотря на различия в их архитектуре, является то, что захват изменений осуществляется непосредственно на целевой базе — этот процесс достаточно «тяжелый», поскольку связан с парсингом и трансформацией данных.

По сравнению с подключением встроенного аудита СУБД способ захвата нагружает систему не так значительно, однако это не всегда можно делать на исходной базе, особенно когда прикладная система активно выполняет свою целевую функцию. Дополнительно следует отметить, что мониторинг, выполняемый по журналу транзакций, позволяет извлекать информацию о давно произведенных операциях в базе. Если изменения по каким-то объектам до этого не отслеживались, то аудитор тем не менее может запросить историю модификации любого из них за прошлый период по журналу транзакций. Очевидно, что эта функциональность полезна и востребована, но ее поддержка на сервере прикладной системы в критической степени сказывается на производительности системы, поскольку исполнение такого рода запросов вызывает необходимость выполнения ресурсоемкого сканирования большого количества (несколько тысяч) архивных файлов журнала транзакций.

Архитектура сервера контроля изменений

Перспективным можно считать вариант, когда серверы с базами данных информационной системы полностью избавляются от «непрофильной» работы, а выполнение задач по захвату, обработке и сохранению информации об изменениях осуществляется на специально выделенном сервере. На рис. 1 приведена предлагаемая архитектура системы такого класса.

Рис. 1. Архитектура сервера контроля изменений

Центральным звеном архитектуры является **сервер контроля изменений** (СКИ), на котором выполняется процесс захвата и обработки записей из архивных журналов, поступающих с серверов, содержащих исходные базы. Предполагается, что любая СУБД поддерживает механизм архивирования журналов транзакций, применяемый для решения задач резервного копирования и восстановления данных. Именно эти архивные журналы передаются на СКИ и обрабатываются, формируя структурированное представление о прочитанном из журнала изменении, которое в виде отдельной записи (или группы записей) может быть сохранено в базе СКИ.

СКИ может обслуживать несколько исходных баз, для каждой из которых порождается свой процесс захвата и обработки изменений. Основная особенность предложенной архитектуры заключается в том, что сами исходные базы на сервер СКИ не перемещаются, а метаданные о структуре базы, составляющие содержимое словаря данных (место размещения названий таблиц, полей и т. п.), могут быть получены из архивного журнала или каким-то иным способом переданы на СКИ. Не

имея этих метаданных (а они являются неотъемлемой частью любой базы данных и используются при чтении файлов журнала), анализировать записи журналов невозможно — они закодированы, и, например, вместо имен полей используются их номера.

К числу дополнительных преимуществ, вытекающих из данной архитектуры, относятся: экономическая целесообразность (для реализации подобной системы достаточно одного сервера); возможность получения интегральной информации (например, в виде отчетов) об изменениях, произведенных в нескольких источниках; организация централизованного долговременного хранения файлов архивных журналов и т. п.

СКИ для СУБД Oracle

На сегодняшний день Oracle, пожалуй, одна из немногих компаний, обеспечивающих разработчиков набором штатных инструментальных средств для решения задач мониторинга. Алгоритмы консолидации изменений, происходящих и транслируемых из исходных баз данных, основываются на механизме Oracle Streams версии 11g R2, предназначенном для распространения данных на базе очередей сообщений Oracle Advanced Queuing (AQ). Этот механизм достаточно универсален, с его помощью можно выполнять репликацию данных; резервное копирование; загрузку хранилищ данных из прикладных систем; управление событиями и т. п.

В СУБД Oracle используется журнал транзакций фиксированного размера, называемый также оперативным журналом повтора (online redo log). Рабочий объем журнала определяется при конфигурировании базы путем указания количества файлов журнала (не менее двух) и их размера. Выделение свободного места для записи генерируемого системой потока транзакций осуществляется по циклической схеме. Записи последовательно помещаются в один из файлов журнала (активный файл), по достижении конца которого производится переключение (logfile switching) на следующий по порядку файл либо на начало первого файла, если был заполнен последний из файлов журнала.

Для реализации возможности сохранения информации о транзакциях за продолжительный период времени, а также для передачи этой информации в другие системы необходимо включить режим ARCHIVELOG, активирующий механизм генерации архивных журналов (archived logs). При работе в таком режиме система осуществляет копирование заполненных файлов оперативного журнала в архивные журналы. Гарантируется, что до перезаписи файла система сделает копию его предыдущего содержимого.

В Oracle для любой базы может быть порождено несколько архивных журналов (не более 10) для решения различных прикладных задач. Каждый архивный журнал представляет собой папку в файловой системе, где сохраняются копии заполненных файлов оперативного журнала транзакций. Папки архивных журналов могут размещаться не только в локальной файловой системе, но и на удаленных устройствах. Обычно архивные журналы создаются для резервного копирования или для зеркального отображения данных на резервный сервер. Для передачи файлов журнала между основным и резервным серверами предназначен встроенный сервис Log Transport Service (процесс, осуществляющий транспортировку файлов журнала), который выполняет синхронизацию между серверами путем копирования журнальных файлов в определенную папку на резервном компьютере. Для подключения базы к Oracle Streams следует создать отдельный архивный журнал, который

будет передаваться на целевой сервер СКИ. Это рекомендуется сделать, чтобы избежать конфликтов со штатными процедурами архивирования, если таковые имеются.

При организации подключения базы к Oracle Streams возникает вопрос, связанный с выбором способа транспортировки файлов журнала на целевой сервер СКИ. Для этого можно использовать либо штатный сервис Log Transport Service, либо реализовать свое решение для передачи файлов.

Транспортировка архивных журналов

Опыт работы с Log Transport Service показал, что этот сервис имеет ряд принципиальных недостатков, ставящих под сомнение целесообразность его применения в качестве транспортного средства для передачи файлов журнала транзакций. Во-первых, метод аутентификации, используемый сервисом, требует совпадения паролей для учетной записи SYS на исходной и целевой базе данных. Данное требование необходимо при использовании сервиса транспортировки для организации зеркалирования, поскольку очевидно, что в этом случае вся учетная информация должна быть идентична. С другой стороны, при подключении к целевому серверу СКИ это ограничение нелогично и не отвечает требованиям безопасности.

Кроме того, Log Transport Service не обеспечивает гарантированной доставки файлов, в результате чего в журнале на целевом сервере могут возникать разрывы в последовательности отгруженных файлов. Это объясняется push-режимом, лежащим в основе архитектуры данного процесса. Если по каким-то причинам не удается передать файл (например, целевой компьютер выключен или с ним разорвано соединение), то процесс после нескольких неудачных попыток просто «забывает» про передаваемый файл и переключается на следующий. Такие разрывы приходится устранять вручную с помощью копий журнала, в которых следует отыскать все потерянные файлы и скопировать их на целевой сервер СКИ.

Можно рекомендовать альтернативный подход для организации транспортировки файлов журнала на целевой сервер СКИ, который основан на использовании pool-технологии. При подключении исходной базы к системе мониторинга в ней необходимо создать отдельный архивный журнал в папке файловой системы этого сервера. Эту папку следует сделать разделяемой и открыть к ней доступ по сети. На целевом сервере СКИ должен быть реализован и запущен процесс Log_File_Reader, который опрашивает удаленную папку и «забирает» из нее новые файлы. При подключении других внешних источников с архивными журналами каждый из этих журналов будет копироваться в свою папку на целевой сервер СКИ. Таким образом, любой архивный журнал исходной базы будет представлен своей копией, размещаемой в отдельном каталоге целевого сервера СКИ. После того как файл успешно передан, он удаляется из папки первоисточника. Разумеется, что для работоспособности описанной схемы у разделяемой папки-первоисточника должны быть установлены права на чтение/запись/удаление файлов для процесса Log_File_Reader. Изложенная схема по транспортировке файлов журнала обладает одним важным преимуществом — она устраняет возможность появления разрывов в последовательности загружаемых файлов.

Захват изменений

После того как решен вопрос с транспортировкой данных, требуется описать основные понятия и процессы Oracle Streams, которые позволяют выполнять обра-

ботку файлов журнала транзакций. Oracle Streams состоит из трех основных процессов:

- *Capture Process* — захват изменений в рабочих и архивных журналах транзакций, выбор из них записей об изменениях в исходной базе (Change Record, CR) и формирование логических записей изменений (Logical Change Record, LCR), помещаемых в очередь сообщений AQ;
- *Propagate Process* — передача сообщений из очереди исходной базы в очередь на целевой базе;
- *Apply Process* — применение изменений из очереди LCR к таблицам целевой базы либо их передача специальной обрабатывающей программе для выполнения необходимых преобразований.

В Oracle 11g R2 реализован комбинированный способ обработки журналов транзакций Combined Capture Apply, позволяющий через служебную очередь организовать взаимодействие между процессами Capture и Apply (рис. 2).

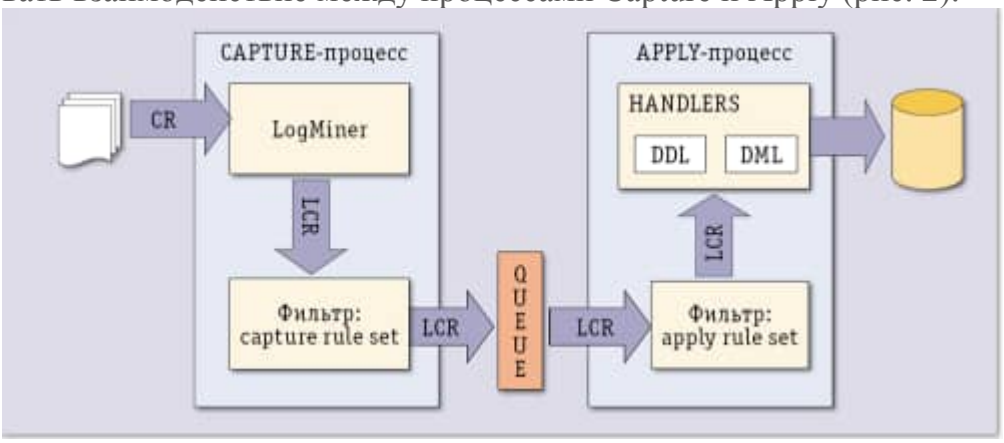


Рис. 2. Схема обработки журналов транзакций с помощью механизма Combined Capture Apply

Процесс захвата изменений осуществляет обработку поступающих log-файлов журнала при помощи вызова штатной утилиты Oracle LogMiner, которая считывает записи об изменениях (CR) и преобразует их в логические записи об изменениях — LCR. Для выполнения этой трансформации LogMiner использует словарь данных исходной базы, выгружаемый в журнал транзакций специальной процедурой (BUILD). На уровне процесса захвата изменений задаются правила, с помощью которых определяются схема и набор таблиц баз данных, подлежащих мониторингу. Соответственно, в очередь процесса Apply будут передаваться только изменения объектов, перечисленные в правилах. Набор правил может изменяться путем добавления в них новых объектов, подлежащих аудиту, или исключения тех, мониторинг которых утратил актуальность. Далее процесс Apply извлекает из очереди входящих сообщений LCR-записи и передает их обрабатывающей прикладной программе. Предварительно выбираемые записи могут быть отфильтрованы по аналогичным правилам, которые задаются для процесса захвата изменений. Обрабатывающая программа разбирает LCR, распознавая указанный там тип операции, и сохраняет ее в базе данных СКИ.

На наш взгляд, главным ограничением существующей версии Streams является отсутствие штатного доступа к словарю данных, который был получен из журнала транзакций. Из-за этого при построении пользовательских сервисов над базой СКИ

приходится обращаться к недокументированным системным таблицам для выбора объектов схемы (названий таблиц, полей и т. п.) из списка.

Управление мониторингом

Описанная технология позволяет реализовать на сервере СКИ сервис, предоставляющий конечным пользователям возможность самостоятельно управлять процессом аудита путем явного указания объектов, подлежащих мониторингу. На основании подключенных к аудиту баз данных пользователи сервиса смогут определять состав таблиц, для которых будут протоколироваться операции по изменению данных и схемы. С этой целью каждый аудитор создает свой профиль, содержащий набор правил, которые определяют интересующие его изменения. Система объединяет правила, указанные во всех профилях, формируя интегральный фильтр, который будет применяться для отбора данных из архивных журналов при захвате изменений.

При задании конкретного правила в своем профиле пользователь может дополнительно указать дату, начиная с которой он хочет получить соответствующие изменения, произведенные в базе прикладной системы. Указание такой даты «инструктирует» систему о необходимости перезагрузки файлов журнала задним числом, но не ранее момента подключения базы к мониторингу. Это возможно, если считать, что все архивные журналы-первоисточники сохранены в файловой системе СКИ.

Интересы аудиторов могут меняться в процессе эксплуатации системы мониторинга — в нее могут добавляться новые правила или удаляться существующие. Система должна отслеживать эти модификации, перестраивая интегральный фильтр, и если оказывается, что часть сохраненных в СКИ данных больше не востребована, они удаляются из базы. Кроме того, при определении любого правила можно указать количество дней до текущей даты, в течение которых будет храниться отобранная по данному правилу информация в базе СКИ.

Роль администратора при такой организации сводится лишь к подключению исходных баз к системе мониторинга, а все задачи управления номенклатурой загружаемой в СКИ информации полностью возлагаются на пользователей-аудиторов, поскольку именно они обладают знанием предметной области и могут точно определить, какие изменения необходимо контролировать.

***

Описанный подход к организации мониторинга изменений баз данных позволяет создать корпоративный сервис централизованного аудита всех критических изменений в прикладных системах компании или организации как на уровне данных, так и их структур. Такой аудит особенно важен в случае взаимодействия большого количества прикладных систем, в которые вносятся изменения, способные оказать влияние на логическую целостность и информационную безопасность. Успешное тестирование и апробация изложенных механизмов были проведены в рамках пилотного проекта в компании ОАО «АТС» — коммерческого оператора оптового рынка электроэнергии в России. Одной из побудительных причин для данного проекта стала возникшая в компании потребность в создании единого универсального механизма аудита на уровне СУБД действий пользователей, программ-загрузчиков, SQL-скриптов и т. п., которые затрагивают изменения в схеме базы и самих данных. Программный комплекс состоит из нескольких сотен взаимодействующих модулей, в которые постоянно вносятся изменения в соответствии с изменениями правил рынка. Подобный механизм аудита дает мощное средство обеспечения ло-

гической целостности и информационной безопасности всего программного обеспечения. Представленное в статье решение по организации аудита на основе журнала транзакций универсально и может быть реализовано на любой платформе СУБД во многих крупных компаниях и организациях.

3. ОБЪЕКТЫ И СРЕДСТВА ИССЛЕДОВАНИЯ

Предметная область в соответствии с вариантом задания. ПК типа IBM PC, дисплейный класс, операционная система Windows XP и выше.

4. ПОДГОТОВКА К РАБОТЕ

Ознакомиться с теоретическими положениями лабораторной работы.

5. ПРОГРАММА РАБОТЫ

Провести анализ методов аудита в СУБД. Оформить отчет.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

6.1. Назовите основные подходы к аудиту баз данных?
6.2. Основные методы аудита?

Лабораторная работа № 15

РЕЗЕРВНОЕ КОПИРОВАНИЕ И ВОССТАНОВЛЕНИЕ БАЗ ДАННЫХ

1. ЦЕЛЬ РАБОТЫ

Изучение методов резервного копирования и восстановления баз данных.

2. ОСНОВЫ ТЕОРИИ

Модели восстановления

Перед тем как браться за настройку резервного копирования, следует выбрать модель восстановления. Для оптимального выбора следует оценить требования к восстановлению и критичность потери данных, сопоставив их с накладными расходами на реализацию той или иной модели.

Как известно, база данных MS SQL состоит из двух частей: собственно, базы данных и лога транзакций к ней. База данных содержит пользовательские и служебные данные на текущий момент времени, лог транзакций включает в себя историю всех изменений базы данных за определенный период, располагая логом транзакций мы можем откатить состояние базы на любой произвольный момент времени.

Для использования в производственных средах предлагается две модели восстановления: простая и полная. Существует также модель с неполным протоколированием, но она рекомендуется только как дополнение к полной модели на период крупномасштабных массовых операций, когда нет необходимости восстановления базы на определенный момент времени.

Простая модель предусматривает резервное копирование только базы данных, соответственно восстановить состояние БД мы можем только на момент создания резервной копии, все изменения в промежуток времени между созданием последней резервной копии и сбоем будут потеряны. В тоже время простая схема имеет небольшие накладные расходы: вам необходимо хранить только копии базы данных, лог транзакций при этом автоматически усекается и не растет в размерах. Также процесс восстановления наиболее прост и не занимает много времени.

Полная модель позволяет восстановить базу на любой произвольный момент времени, но требует, кроме резервных копий базы, хранить копии лога транзакций за весь период, для которого может потребоваться восстановление. При активной работе с базой размер лога транзакций, а, следовательно, и размер архивов, могут достигать больших размеров. Процесс восстановления также гораздо более сложен и продолжителен по времени.

При выборе модели восстановления следует сравнить затраты на восстановление с затратами на хранение резервных копий, также следует принять во внимание наличие и квалификацию персонала, который будет выполнять восстановление. Восстановление при полной модели требует от персонала

определенной квалификации и знаний, тогда как при простой схеме достаточно будет следовать инструкции.
Для баз с небольшим объемом добавления информации может быть выгоднее использовать простую модель с большой частотой копий, которая позволит быстро восстановиться и продолжить работу, введя потерянные данные вручную. Полная модель в первую очередь должна использоваться там, где потеря данных недопустима, а их возможное восстановление сопряжено со значительными затратами.
Виды резервных копий
Полная копия базы данных - как следует из ее названия, представляет собой содержимое базы данных и часть активного лога транзакций за то время, которое формировалась резервная копия (т.е. сведения обо всех текущих и незавершенных транзакциях). Позволяет полностью восстановить базу данных на момент создания резервной копии.
Разностная копия базы данных - полная копия имеет один существенный недостаток, она содержит всю информацию базы данных. Если резервные копии нужно делать довольно часто, то сразу возникает вопрос неэкономного использования дискового пространства, так как большую часть хранилища будут занимать одинаковые данные. Для устранения этого недостатка можно использовать разностные копии базы данных, которые содержат только изменившуюся со времени последнего полного копирования информацию.
Обращаем внимание, разностная копия - это данные от момента последнего полного копирования, т.е. каждая последующая разностная копия содержит в себе данные предыдущей (но при этом они могут быть изменены) и размер копии будет постоянно расти. Для восстановления достаточно одной полной и одной разностной копии, обычно последней. Количество разностных копий следует выбирать исходя из прироста их размера, как только размер разностной копии сравнится с размером половины полной, имеет смысл сделать новую полную копию.
Резервная копия журнала транзакций - применяется только при полной модели восстановления и содержит копию журнала транзакций начиная с момента создания предыдущей копии.

Важно помнить следующий момент - копии журнала транзакций никак не связаны с копиями базы данных и не содержат информацию предыдущих копий, поэтому для восстановления базы вам необходимо иметь непрерывную цепочку копий того периода, в течении которого вы хотите иметь возможность откатывать состояние базы. При этом момент последнего успешного копирования должен быть внутри этого периода.
Посмотрим на рисунок выше, если будет утрачена первая копия файла журнала, то вы сможете восстановить состояние базы только на момент полного копирования, что будет аналогично простой модели восстановления, восстановить состояние базы на любой момент времени вы сможете только после следующего разностного (или полного) копирования, при условии, что це-

почка копий журналов начиная с предшествующего копированию базы и далее будет непрерывна (на рисунке - от третьего и далее).

Журнал транзакций

Для понимания процессов восстановления и назначения разных видов резервных копий следует более подробно рассмотреть устройство и работу журнала транзакций. Транзакция - это минимально возможная логическая операция, которая имеет смысл и может быть выполнена только полностью. Такой подход обеспечивает целостность и непротиворечивость данных при любых ситуациях, так как промежуточное состояние операции недопустимо. Для контроля над любыми изменениями в базе предназначен журнал транзакций.

При выполнении любой операции в журнал транзакций добавляется запись о начале транзакции, каждой записи присваивается уникальный номер (LSN) из неразрывной последовательности, при любом изменении данных в журнал вносится соответствующая запись, а после завершения операции в журнале появляется отметка о закрытии (фиксации) транзакции.

При каждом запуске система анализирует журнал транзакций и откатывает все незафиксированные транзакции, одновременно с этим происходит накат изменений, которые зафиксированы в журнале, но не были записаны на диск. Это дает возможность использовать кэширование и отложенную запись, не опасаясь за целостность данных даже при отсутствии систем резервного питания.

Та часть журнала, которая содержит активные транзакции и используется для восстановления данных называется активной частью журнала. Она начинается с номера, который называется минимальным номером восстановления (MinLSN).

В простейшем случае MinLSN - это номер записи первой незавершенной транзакции. Если посмотреть на рисунок выше, то открыв синюю транзакцию мы получим MinLSN равную 321, после ее фиксации в записи 324, номер MinLSN изменится на 323, что будет соответствовать номеру зеленой, еще не зафиксированной, транзакции.

На практике все немного сложнее, например, данные закрытой синей транзакции могут быть еще не сброшены на диск и перемещение MinLSN на 323 сделает восстановление этой операции невозможной. Для того, чтобы избежать таких ситуации было введено понятие контрольной точки. Контрольная точка создается автоматически при наступлении следующих условий:

При явном выполнении инструкции CHECKPOINT. Контрольная точка срабатывает в текущей базе данных соединения.

При выполнении в базе данных операции с минимальной регистрацией, например, при выполнении операции массового копирования для базы данных, на которую распространяется модель восстановления с неполным протоколированием.

При добавлении или удалении файлов баз данных с использованием инструкции ALTER DATABASE.

При остановке экземпляра SQL Server с помощью инструкции SHUTDOWN или при остановке службы SQL Server (MSSQLSERVER). И в том, и в другом случае будет создана контрольная точка каждой базы данных в экземпляре SQL Server.
Если экземпляр SQL Server периодически создает в каждой базе данных автоматические контрольные точки для сокращения времени восстановления базы данных.
При создании резервной копии базы данных.
При выполнении действия, требующего отключения базы данных. Примерами могут служить присвоение параметру AUTO_CLOSE значения ON и закрытие последнего соединения пользователя с базой данных или изменение параметра базы данных, требующее перезапуска базы данных.
В зависимости от того, какое событие произошло раньше, MinLSN будет присвоено значение либо номера записи контрольной точки, либо начала самой старой незавершенной транзакции.
Усечение журнала транзакций
Журнал транзакций, как и любой журнал, требует периодической очистки от устаревших записей, иначе он разрастется и займет все доступное место. Учитывая, что при активной работе с базой размер лога транзакций может значительно превышать размер базы, то этот вопрос актуален для многих администраторов.
Физически файл журнала транзакций является контейнером для виртуальных журналов, которые последовательно заполняются по мере роста лога. Логический журнал, содержащий запись MinLSN является началом активного журнала, предшествующие ему логические журналы являются неактивными и не требуются для автоматического восстановления базы.
Если выбрана простая модель восстановления, то при достижении логическими журналами размера равного 70% физического файла происходит автоматическая очистка неактивной части журнала, т.н. усечение. Однако это не приводит к уменьшению физического файла журнала, усекаются только логические журналы, которые после этой операции могут использоваться повторно.

Если количество транзакций велико и к моменту достижения 70% размера физического файла не окажется неактивных логических журналов, то размер физического файла будет увеличен.
Таким образом файл лога транзакций при простой модели восстановления будет расти согласно активности работы с базой до тех пор, пока не будет надежно вмещать всю активную часть журнала. После чего его рост прекратится.
При полной модели неактивную часть журнала нельзя усечь до тех пор, пока она полностью не попадет в резервную копию. Усечение журнала производится при условии, что выполнена резервная копия журнала транзакций, после чего была создана контрольная точка.

Неправильная настройка резервного копирования журнала транзакций при полной модели способно привести к неконтролируемому росту файла журнала, что часто составляет проблему для неопытных администраторов. Также часто попадаются советы по ручному усечению журнала транзакций. При полной модели восстановления делать этого не следует категорически, так как тем самым вы нарушите целостность цепочки копий журнала и сможете восстановить базу только на момент создания копий, что будет соответствовать простой модели.

В этом случае самое время вспомнить то, о чем мы говорили в начале статьи, если затраты на полную модель превышают затраты на восстановление следует отдать предпочтение простой модели.

Простая модель восстановления

Теперь, после получения необходимого минимума знаний, можно перейти к более подробному рассмотрению моделей восстановления. Начнем с простой. Допустим, на момент сбоя у нас имеется одна полная и две разностные копии:

Резервное копирование выполнялось раз в сутки и последняя копия была создана ночью с 21-го на 22-е. Сбой происходит вечером 22-го до создания очередной копии. В этом случае нам потребуется последовательно восстановить полную и последнюю разностные копии, при этом данные за последний рабочий день будут утеряны. Если по каким-либо причинам копия от 21-го также окажется повреждена, то мы можем восстановить предыдущую копию, потеряв еще день работы, в тоже время повреждение копии за 20-е число никак не помешает успешно восстановить данные на вечер 21-го, при наличии соответствующей копии.

Полная модель восстановления

Рассмотрим аналогичную ситуацию, но с применением полной модели восстановления. Резервные копии у нас также делаются ежесуточно, по принципу полная + разностные, а также несколько раз в сутки копируется лог транзакций.

Процесс восстановления в этом случае будет более сложен. Прежде всего потребуется создать вручную резервную копию заключительного фрагмента журнала (показана красным), т.е. часть журнала с момента прошлого создания копии и до аварии.

Если этого не сделать, то восстановить базу можно будет только до состояния на момент создания последней копии журнала транзакций.

При этом повреждение файла копии журнала за предыдущий день не помешает нам восстановить актуальное состояние базы, но ограничит нас моментом создания последней копии, т.е. текущими сутками.

Затем последовательно восстанавливаем полную и разностную копию и цепочку копий журнала, созданную после последнего резервного копирования, последней восстанавливаем копию заключительного фрагмента журнала, что даст нам возможность восстановить базу прямо на момент аварии или произвольный, предшествовавший ему.

Если последняя разностная копия будет повреждена, то в случае с простой моделью это приведет к потере еще одного рабочего дня, полная модель позволяет восстановить предпоследнюю копию, после чего нужно будет восстановить всю цепочку копий лога транзакций от момента предпоследней копии и до сбоя. Глубина восстановления зависит только от глубины непрерывной цепочки логов.

С другой стороны, если одна из копий лога транзакций будет повреждена, скажем, предпоследняя, то восстановить данные мы сможем только на момент последней резервной копии + период в неповрежденной цепочке копий журналов. Например, если журналы делались в 12, 14 и 16 часов и поврежден журнал, созданный в 14 часов, то располагая суточной копией мы сможем восстановить базу до момента окончания непрерывной цепочки, т.е. до 12 часов.

3. ОБЪЕКТЫ И СРЕДСТВА ИССЛЕДОВАНИЯ

Предметная область в соответствии с вариантом задания. ПК типа IBM PC, дисплейный класс, операционная система Windows XP и выше.

4. ПОДГОТОВКА К РАБОТЕ

Ознакомиться с теоретическими положениями лабораторной работы.

5. ПРОГРАММА РАБОТЫ

Изучить модели копирования и восстановления баз данных. Оформить отчет.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

6.1. Назовите основные подходы к копированию и восстановлению баз данных?

6.2. Основные методы восстановления баз данных?

ПРИЛОЖЕНИЕ

ВАРИАНТЫ ПРИМЕРНЫХ ПРЕДМЕТНЫХ ОБЛАСТЕЙ ДЛЯ ВЫДАЧИ ЗАДАНИЙ ЛАБОРАТОРНЫХ РАБОТ ПО БД

ЗАДАНИЕ 1:
ИНФОРМАЦИОННАЯ СИСТЕМА "ДЕТСКАЯ ПОЛИКЛИНИКА"

Система хранит информацию о врачах, пациентах, заболеваниях, детских учреждениях. Информационная система должна предоставлять данные по отдельным запросам:

- информацию о враче (фамилия, специализация, стаж, оклад, совместительство);
- данные о больном (фамилия, возраст, адрес, детское учреждение, место работы родителей, хронические заболевания, прививки, последнее обращение к врачу);
- данные о детском учреждении (наименование, адрес, количество детей, наличие карантина, выявленные инфекционные заболевания, дата последнего профилактического обследования).

Кроме того периодически должны выдаваться следующие ведомости:

- список детских учреждений, в которых зафиксированы инфекционные заболевания;
- статистический отчет по заболеваниям;
- отчет по заболеваемости в детских учебных заведениях.

ЗАДАНИЕ 2:
ИНФОРМАЦИОННАЯ СИСТЕМА "УПРАВЛЕНИЕ ДОРОЖНЫМ ДВИЖЕНИЕМ"

Система хранит информацию об автомобилях, зарегистрированных в области, владельцах автомашин, о нарушениях правил дорожного движения. Системой представляется следующая информация по отдельным запросам:

- информация об автомобиле (регистрационный номер, марка, номер двигателя, дата последнего техосмотра, владелец);
- информация о владельце (номер и марка автомобиля, количество предупреждений в талоне, дата медицинского обследования, фамилия, адрес);
- информация о водителях некоторого автохозяйства.

Периодически и по запросу должны выдаваться следующие ведомости:

- список нарушителей за указанный период;
- список водителей автохозяйств, направленных на медицинскую комиссию;

- статистика нарушений по типам, маркам автомобилей, по автохозяйствам, владельцам.

ЗАДАНИЕ 3:
ИНФОРМАЦИОННАЯ СИСТЕМА "ТЕЛЕАТЕЛЬЕ"

Хранит информацию о типах телевизоров, комплектующих деталях, обслуживающих бригадах, о качестве ремонта. В ателье реализована бригадная форма работы: задание выдается на бригаду, каждая бригада отдельно обеспечивается деталями.

Отдельные запросы, реализуемые системой, могут касаться:

- конкретного телевизора (номер приемной квитанции, тип телевизора, адрес владельца, дата приемки, срок исполнения заказа);
- комплектующей детали (название, маркировка, возможная замена, количество деталей на складе);
- работа отдельной бригады (фамилия бригадира, количество членов бригады, количество выполненных заказов, выполнение плана в стоимостном выражении, рекламация и возврат).

Система должна также обеспечить выдачу следующих выходных документов:

- отчет о выполнении планового задания; состояние склада телеателье;
список заказов с истекшим сроком исполнения.

ЗАДАНИЕ 4:
ИНФОРМАЦИОННАЯ СИСТЕМА "ЗООПАРК"

Хранит информацию о животных, занимаемых ими помещениях, о рационе животных, об обслуживаемом персонале. Система должна представлять информацию по следующим запросам:

- информацию о конкретном животном (кличка, возраст, вид, размещение, состояние здоровья);
- данные о работниках зоопарка (фамилия, возраст, специальность, отделение, стаж, адрес);

Кроме того периодически должны выдаваться следующие документы:

- рекомендуемый рацион для всех животных;
- список заболевших животных;
- список животных с адресами по видам;
- состояние животных, обслуживаемых отдельными работниками;
- ведомость заполнения помещений зоопарка.

ЗАДАНИЕ 5:
ИНФОРМАЦИОННАЯ СИСТЕМА "АБИТУРИЕНТ"

Создается информационная система АБИТУРИЕНТ для автоматизации работы приемной комиссии вуза. Один из фрагментов этой предметной области требует обработки анкетных данных абитуриентов. Для этого фрагмента разработать программу :

- меню и средства диалога; ввода и изменения данных; подготовки печатных форм.

Анкета включает следующие данные об абитуриенте :

- регистрационный номер; фамилия, имя, отчество; дата рождения;
- окончание среднего учебного заведения; дата окончания;
- наличие красного диплома или золотой медали;
- адрес (город, улица, N дома, квартиры, телефон);
- выбранная специальность.

Исходными документами для заполнения анкеты является аттестат или диплом о среднем образовании, заявление абитуриента. В вузе определен список специальностей, который может изменяться ежегодно. По каждой специальности вуза определен список сдаваемых предметов, например:

- математика (П), - математика (У), - родной язык (П), и т.д.

ИС должна обеспечивать выполнение следующих функций :

- ввод и коррекцию анкетных данных;
- просмотр анкетных данных по специальностям в алфавитном порядке;
- ввод, коррекцию и просмотр специальностей и сдаваемых предметов;
- вывод на печать анкетных данных абитуриентов, имеющих красный диплом или медаль;
- вывод на печать всех инициалов абитуриентов по специальностям в алфавитном порядке с указанием сдаваемых предметов;
- вывод на печать анкетных данных по специальностям.

ЗАДАНИЕ 6:
ИНФОРМАЦИОННАЯ СИСТЕМА "НАЧИСЛЕНИЯ ЗАРПЛАТЫ

Создается информационная подсистема НАЧИСЛЕНИЯ ЗАРПЛАТЫ для автоматизации начисления заработной платы в бухгалтерии.

Зарплата начисляется работникам вуза, имеющим установленные оклады (сдельных работ нет). На каждого работника хранятся следующие данные: личный номер; ф.и.о.; должность; оклад; семейное положение и число детей; данные о невыходе на работу по болезни (даты заболевания и выздоровления) и т.д.

В период болезни работнику начисляется 50 % зарплаты; 100 % начисляется лишь членам профсоюза. Работникам могут начисляться премии и другие надбавки. С общей суммы зарплаты снимается подоходный налог 8 % и налог за бездетность 6 %.

ИС должна обеспечивать:

- ввод, изменение анкетных данных работников, сведения о болезнях, надбавках; ежемесячный перерасчет зарплаты с выдачей ведомости на экран и на печать.

Разработать программы:

меню и выдачи печатных форм; ввода и изменения данных.

ЗАДАНИЕ 7:
КОНТРОЛЬ ИСПОЛНЕНИЯ ПОРУЧЕНИЙ

Создается информационная подсистема КОНТРОЛЬ ИСПОЛНЕНИЯ ПОРУЧЕНИЙ для некоторой организации. В качестве исходной информации используются данные:

- порядковый номер поручения, название поручения, содержание поручения, дата выдачи поручения, срок исполнения, дата фактического исполнения, исполнитель, кто выдал поручение.

Поручения могут выдавать руководитель организации и руководители подразделений. Ввод всех данных в персональную ЭВМ выполняет один оператор.

ИС должна обеспечивать:

- ввод и коррекцию данных о поручениях, просмотр поручений по некоторой дате, ежедневную печать поручений с текущей датой исполнения (для руководителя организации).

Разработать программы:

меню и выдачи печатных форм, ввода, изменения и просмотра данных.

ЗАДАНИЕ 8
ИНФОРМАЦИОННАЯ СИСТЕМА "СНАБЖЕНИЕ"

Информационная система создается для оптовой базы. Основным назначением оптовой базы является снабжение сети магазинов различными товарами. Отдел снабжения в каждый момент времени должен иметь точные данные о названии товаров, их количестве на складе, о названии магазинов, о названии и количестве каждого вида товара в каждом магазине, о заявках магазинов на текущий год.

Отдел снабжения должен иметь возможность проделывать следующие операции:

- включить новый товар в список товаров на складе; удалить ненужный товар из складского списка; включить новый магазин в список магазинов; удалить ненужный магазин из списка; выполнить поступление некоторого товара на склад; просмотреть информацию о товарах на складе; просмотреть информацию о товарах по магазинам; провести инвентаризацию склада и каждого магазина; выдать магазину товар со склада и отпечатать накладную; ввести заявку магазина на текущий год.

ЗАДАНИЕ 9:
ИНФОРМАЦИОННАЯ СИСТЕМА "ДЕКАНАТ"

Создается информационная система ДЕКАНАТ для автоматизации работы деканата факультета вуза. Пусть максимальное число кафедр на факультете равно 10, кафедры готовят студентов по 12 специальностям.

По каждой специальности имеется учебный план, который содержит список всех предметов, изучаемых на этой специальности, с указанием общего количества лекционных, практических, лабораторных часов, раскладку предметов и курсовых работ по семестрам с указанием количества часов, форму сдачи предмета (зачет/экзамен).

По окончании вступительных экзаменов приёмная комиссия вуза передает в деканат личные дела и списки зачисленных абитуриентов. На каждого студента 1-го курса заводится учебная карточка, в которую заносятся его точные данные, а также список предметов, подлежащих сдаче согласно учебному плану специальности. По мере сдачи предметов и перехода с курса на курс учебная карточка заполняется соответствующими оценками.

По окончании вуза копия учебной карточки выдаётся студенту как приложение к диплому. Учебная карточка выдаётся также при переводе в другой вуз.

Система должна обеспечивать ввод и обработку учебных планов специальностей, учебных карточек студентов, выдачу списков студентов по различным выборкам, должна исключать дублирование данных о предметах. Пользователями ИС являются декан и секретарь деканата. Систему можно расширить, включив обработку данных о преподавателях, их нагрузке. Система ДЕКАНАТ может включаться в ИС всего вуза. Каждое подразделение, кафедра, учебный отдел, научная часть, бухгалтерия, от дел кадров и пр. могут иметь собственные информационные под системы.

ЗАДАНИЕ 10:
ИНФОРМАЦИОННАЯ СИСТЕМА "КАДРЫ"

Разработать ИС КАДРЫ для автоматизации работы отдела кадров предприятия с числом сотрудников до 1000 человек. Система должна функционировать в двух режимах: первичной загрузки данных и текущей обработки информации.

В режиме загрузки базы данных система должна предоставлять ввод данных из личных карточек работающих с контролем вводимой информации. В режиме текущей обработки система должна реализовывать функции:

- обработку данных по движению кадров: прием, увольнение, перемещение;
- получение статистической отчетной информации по уволен ным и работающим в различных разрезах;

- получение справочной информации по данным, содержащимся в личной картотеке;

- ведение табельного учета по отсутствующим на местах.

ЗАДАНИЕ 11:
ИНФОРМАЦИОННАЯ СИСТЕМА "ЗАРПЛАТА"

Разработать ИС "Зарплата" для автоматизации учета труда и заработной платы.

Система должна предоставлять следующие функциональные возможности:

а) начисление аванса за первую половину месяца; основной заработной платы рабочим-повременщикам и распределение премии с учетом коэффициента трудового участия КТУ; основной заработной платы по установленному окладу или тарифу; различного рода премий, доплат, надбавок к основной заработной плате;

б) расчет сдельного заработка рабочим-сдельщикам, работающим по бригадному подряду и индивидуально; пенсии работающим пенсионерам;

в) исчисление налогов;

г) удержание по исполнительным листам; за товары, купленные в кредит; различных ссуд Госбанка; профсоюзных взносов;

д) перечисление сумм организациям Госстраха; в сберегательный банк;

е) формирование расчетно-платежной документации.

ЗАДАНИЕ 12
ИНФОРМАЦИОННАЯ СИСТЕМА "КОММЕРЦИЯ"

В вашем городе создается компьютерный центр коммерческой информации. Его функцией является сбор сведений о предприятиях, фирмах, кооперативах и пр., о производимых ими товарах и услугах, систематизация этих данных по различным параметрам, издание ежеквартальных бюллетеней о сведениях, зарегистрированных за прошедший квартал, выдача интересующей информации по заказу отдельных лиц и организаций.

Информация собирается из периодической печати (рекламные объявления в газетах и журналах),а также может предоставляться самой регистрируемой организацией. Еженедельная порция вводимых данных составляет до 10 новых организаций и до 100 организаций, в которых обновляется информация. Хранимая в базе данных информация об организации должна включать следующие сведения: точное название организации; страна, город и точный адрес, телефон, телекс, телефакс; основные виды деятельности или отрасли производства; вид или наименование производимых товаров или услуг.

Оперативная информация: что приобретается, продается, кто требуется на работу и пр. Данные в базе данных не должны дублироваться.

ИС должна обеспечивать выборку информации по различным критериям, например: "выдать список фирм, занимающихся производством бытовой электроники", далее "в этом списке выбрать список фирм, производящих электронные часы" и т.д.

ЗАДАНИЕ 13
ИНФОРМАЦИОННАЯ СИСТЕМА "КОМИССИОННЫЙ МАГАЗИН"

Информационная система создается для автоматизации работы комиссионного магазина. Основным назначением системы является учет товарооборота магазина.

Система должна предоставлять по запросу выдачу данных о названии товаров, их количестве, о товарах, подлежащих переоценке в связи с истечением срока и обеспечивать выполнение следующие операции: включить новый товар в список товаров; удалить реализованный товар из списка; просмотреть информацию о товарах по магазину; провести инвентаризацию магазина; выдать ведомость товаров сданных в магазин на определенную дату.

ЗАДАНИЕ 14
ИНФОРМАЦИОННАЯ СИСТЕМА "АВТОСЕРВИС"

Создается система для хранения и выдачи информации об автомобилях, находящихся на обслуживании, владельцах автомобилей, неисправностях и о наличии деталей на складе.

Информация об автомобиле должна содержать данные: номер квитанции, марку автомобиля, неисправность, детали, необходимые для ремонта. Информация о владельцах должна содержать следующие данные: Ф.И.О., номер квитанции, марку автомобиля.

Информация о запчастях должна содержать данные: код детали, название, стоимость, расценка ее замены.

Периодически по запросу должны выдаваться следующие сведения:

- перечень Ф.И.О. клиентов с указанием марки автомобиля, вида неисправности, номера накладной и перечня используемых деталей. Запрос о наличии определенной детали на складе. Результаты инвентаризации склада.

ЗАДАНИЕ 15
ИНФОРМАЦИОННАЯ СИСТЕМА "УНИВЕРМАГ"

Информационная система хранит информацию о товарах, поставщиках товаров, отделах универмага и сотрудниках, работающих в универмаге.

Система должна представлять информацию по следующим запросам:

- информация о поставщиках товаров,
- информация о товарах.

Система должна обеспечивать выдачу следующих выходных документов:

- список фирм, поставляющих товар; список товаров, находящихся на реализации; список цен на товары; список стран-производителей товаров; адреса фирм-поставщиков.

ЗАДАНИЕ 16
ИНФОРМАЦИОННАЯ СИСТЕМА "АВТОХОЗЯЙСТВО"

Система хранит информацию об автомобилях, зарегистрированных в данном автохозяйстве, шоферах автомашин, о нарушениях правил дорожного движения. Системой представляется следующая информация по отдельным запросам:

- информация об автомобиле (регистрационный номер, марка, номер двигателя, дата последнего техосмотра, владелец); информация о шофере автохозяйства (номер и марка автомобиля, количество предупреждений в талоне, дата медицинского обследования, фамилия, адрес);

Периодически и по запросу должны выдаваться следующие ведомости:

- список нарушителей за указанный период; список водителей автохозяйства, направленных на медицинскую комиссию; статистика нарушений по типам, маркам автомобилей.

ЗАДАНИЕ 17
ИНФОРМАЦИОННАЯ СИСТЕМА "ДЕТСКАЯ БОЛЬНИЦА"

Система хранит информацию о врачах, пациентах, заболеваниях, детских учреждениях. Информационная система должна предоставлять данные по отдельным запросам:

информацию о враче (фамилия, специализация, стаж, оклад, совместительство); данные о больном (фамилия, возраст, адрес, детское учреждение, место работы родителей, хронические заболевания, прививки, последнее обращение к врачу); информацию о лекарствах для приобретения их в аптеке.

Кроме того периодически должны выдаваться следующие ведомости:

- список больных детских учреждений, в которых зафиксированы инфекционные заболевания; перечень дефицитных лекарств, отсутствующих в больнице.

ЗАДАНИЕ 18
ИНФОРМАЦИОННАЯ СИСТЕМА «АЭРОПОРТ»

Создается система для хранения и выдачи информации о самолетах, о расписании самолетов, о сотрудниках аэропорта.

Периодически по запросу должны выдаваться следующие сведения:

- информацию о самолетах (тип самолета, номер самолета, номер обслуживающей самолет бригады);
- информацию о расписании (номер рейса, пункт отправления, пункт назначения, время отправления, время полета, стоимость билета),
- Информацию о летном составе (Ф.И.О., должность, стаж работы, адрес, оклад).

ЗАДАНИЕ 19
ИНФОРМАЦИОННАЯ СИСТЕМА "СКЛАД"

Информационная система создается для объекта склад.

Периодически по запросу должны выдаваться данные о названии товаров, их количестве на складе.

Оператор ЭВМ должен иметь возможность проделывать следующие операции обработки данных:

- включение нового товара в перечень товаров на складе;
- удаление ненужный товар из списка;
- просмотреть информацию о наличии товаров;

а также выдачу документов: инвентаризации склада; накладную на получение товаров со склада; ведомость пересортировки товаров.

ЗАДАНИЕ 20
ИНФОРМАЦИОННАЯ СИСТЕМА "СПОРТ"

Создается информационная система СПОРТ для автоматизации обработки данных о проводимых соревнованиях и чемпионатах.

Перечень информационных требований к системе:

- выдача данных о результатах чемпионата по определенному виду спорта с указанием страны, занятого места и количества набранных баллов,
- выдача информации о спортсменах-победителях чемпионата,
- перечень чемпионатов, проводимых в стране с указанием места, времени, вида спорта, стран участниц

ЗАДАНИЕ 21
ИНФОРМАЦИОННАЯ СИСТЕМА "АВТОБУСНОЕ ДВИЖЕНИЕ"

Информационная система создается для автоматизации процесса обработки и хранения данных о проверке состояния машин и возможности ответов на запросы, представленные следующим набором требований:

- ведомость контроля за состоянием машин, закрепленных за бригадой;
- расписание рейсов автобусов с указанием направления движения и временем отправления,
- расписание отправления автобусов из АТП (для диспетчера);
- ведомость выполнения расписания движения.

Периодически и по запросу должны выдаваться следующие ведомости:

- данных о водителях бригады, с указанием стажа работы, количества нарушений, виды поощрения;
- данные об авариях и невыхода транспорта с указанием причины.

ЗАДАНИЕ 22
ИНФОРМАЦИОННАЯ СИСТЕМА "ЖЕЛЕЗНОДОРОЖНЫЙ ТРАНСПОРТ"

Информационная система создается для автоматизации процесса обработки и хранения оперативных данных и справочной информации о работе железнодорожного транспорта. Она должна предусматривать ответы на некоторые запросы и должна отвечать следующим требованиям:

- оперативный ввод информации о заявках на перевозку грузов и пассажиров;
- печать графика движения железнодорожных составов пассажирских и грузовых) исходя из пропускной способности транспортного узла.
- печать графика работы железнодорожных бригад (машинистов), учитывая болезни, отпуска, выполнения обязанностей и т.д.)
- расчет графика профилактики , ремонта подвижного состава, транспортных путей;
- выдавать ведомость о перевозках пассажиров по дням, месяцам;
- выдавать ведомость состава обслуживающего персонала локомотивных бригад.

Периодически и по запросу предусмотреть выдачу оперативных данных:

- о нарушении графика движения, с указанием причин и виновных,
- о наличии неиспользуемого транспортного парка.

ЗАДАНИЕ 23
ИНФОРМАЦИОННАЯ СИСТЕМА "ВОЕНКОМАТ"

Создать информационную систему для автоматизации процесса обработки и хранения данных о результатах прохождения призывной комиссии и

возможности ответов на некоторые вопросы, представленные следующими требованиями:

- распечатка учетно-послужных карточек призывников, содержащих анкетные данные о призывнике,
- распечатка медицинских ведомостей призывников, с указанием состава врачей приемной комиссии и результатах пригодности призывника к военной службе;
- распечатка ведомостей призывной медицинской комиссии с указанием ФИО призывника, кода учетно-призывной карточки и рода войск;
- запрос данных о составе медицинской комиссии с указанием специальностей врачей, номера больницы и других анкетных данных;
- запрос данных о распределении призывников по родам войск с указание количества призывников в разрезе по роду войск.

ЗАДАНИЕ 24
ИНФОРМАЦИОННАЯ СИСТЕМА "КНИГОХРАНИЛИЩЕ"

Создать информационную систему для автоматизации процесса обработки и хранения данных об имеющихся в данном книгохранилище книгах и возможности ответов на некоторые запросы, представленные следующим перечнем информационных требований:

- результаты инвентаризации книг по отделам с указанием степени изношенности;
- сведения о новых поступлениях в книгохранилище;
- сведения о списанных книгах;
- запрос данных о наличии произведений конкретного автора;
- запрос данных об имеющихся в книгохранилище книгах по конкретной тематике (тематический каталог).

ЗАДАНИЕ 25
ИНФОРМАЦИОННАЯ СИСТЕМА "ОТЕЛЬ"

Создать информационную систему для автоматизации процесса обработки и данных о работе отеля.

Система должна обрабатывать и хранить данные о служащих отеля, данные о количестве свободных номеров и стоимости проживания в номере в зависимости от категории, а также вести оперативный учет о проживающих, отъезжающих жильцах.

Информационные требования и запросы к системе:

- распечатка данных о проживающих с указанием номера и срока проживания;
- список забронированных номеров с указание срока действия брони;
- данные об иностранцах, проживающих в отеле;

- запрос данных о служащих отеля с указанием стажа работы в отеле,
- запрос об экскурсоводах, их анкетные данные и профиль экскурсий.

ЗАДАНИЕ 26
ИНФОРМАЦИОННАЯ СИСТЕМА "АПТЕКОУПРАВЛЕНИЕ"

Создается система для хранения и выдачи информации для автоматизации процесса обработки данных о работе аптекоуправления.

Система должна обеспечивать выдачу следующих выходных документов:

- Ведомость наличия в аптеках города особо дефицитных лекарств с указанием адреса аптеки, вида упаковки (таблетки, раствор, для инъекций) и их стоимости.
- Сведения о поставщиках импортных лекарств, с указанием наименования лекарства, номера лицензии на изготовление лекарства, объема поставок, срока поставки и вида расчета (рубли, конвертируемая валюта).
- Запрос на поставку лекарств в конкретную аптеку с указанием наименования, количества и срока поставки.

Минобрнауки России
ФГБОУ ВО «Тульский государственный университет»
Технический колледж имени С.И. Мосина

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ И САМОСТОЯТЕЛЬНЫХ РАБОТ

по дисциплине

Основы алгоритмизации ипрограммирования

специальность 09.02.01 Компьютерные системы и комплексы

2023 г.

УТВЕРЖДЕНЫ
Цикловой комиссией информационных технологий
Протокол от «13» января 2023 г. № 6
Председатель цикловой комиссии И.В. Миляева

Содержание

Практическое занятие № 1

Разработка линейных алгоритмов и алгоритмов ветвлениия

Цель работы: научиться составлять базовые структоры алгоритмов – линейный ветвящиеся алгоритмы.

Краткие теоретические сведения.

Линейным называется алгоритм, в котором действия выполняются последовательно – одно за другим.

Разветвляющийся называется алгоритм, в котором действие выполняется по одной из возможных ветвей решения задачи, в зависимости от выполнения условий. В отличие от линейных алгоритмов*, в которых команды выполняются последовательно одна за другой*, в разветвляющиеся алгоритмы входит условие, в зависимости от выполнения или невыполнения которого выполняется та или иная последовательность команд (действий) – ветка (рисунок 1).

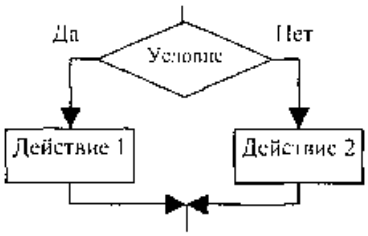


Рисунок 1

В качестве условия в разветвляющемся алгоритме может быть использовано любое понятное исполнителю утверждение, которое может соблюдаться (быть истинно) или не соблюдаться (быть ложно). Такое утверждение может быть выражено как словами, так и формулой. Таким образом, алгоритм ветвления состоит из условия и двух последовательностей команд.

Ветвящийся процесс, включающий в себя две ветви, называется простым, более двух ветвей — сложным. Сложный ветвящийся процесс можно представить с помощью простых ветвящихся процессов.

Примером может являться разветвляющийся алгоритм, изображенный на рисунке 2. Аргументами этого алгоритма являются числа *A* и *B*, а результатом — число *X*. Если условие *A >= B* истинно, то выполняется операция умножения чисел *(X = A* * B), в противном случае выполняется операция сложения *(X = A + B)*. В результате печатается то значение *X*, которое получается в результате выполнения одного из действий.

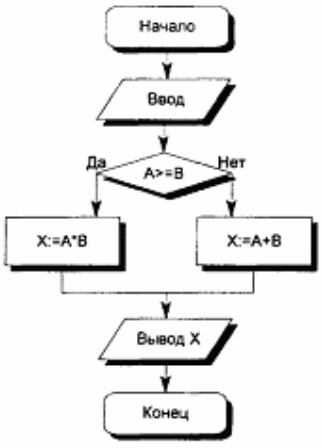


Рисунок 2

Пример выполнения задания

Задание 1: вычислить значение функции

$$y = \frac{\sin^3 c * \cos^2 a}{5 \sin^d b} + \frac{2}{15} \text{ при}$$

$a = 9{,}5$; $b = 1{,}365$; $c = 6{,}5$; $d = 5$.

Блок=схема алгоритма представлена на рисунке

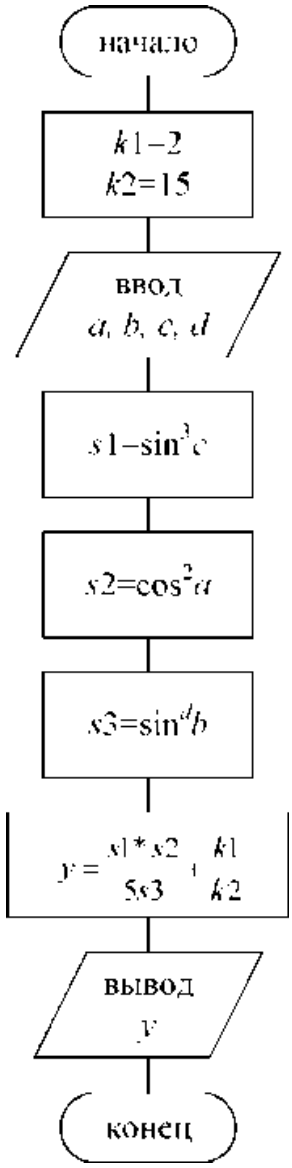


Рисунок 3

Задание 2: по заданным координатам *x* и *y* определить, где находится точка (рисунок 3): внутри заштрихованной области; вне заштрихованной области; на границе этой области.

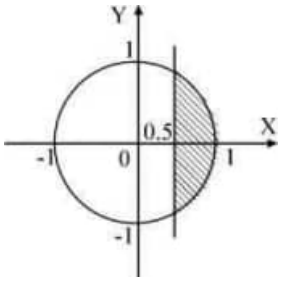


Рисунок 2 — Постановка задачи

Метод решения задачи:

1. Для решения задачи будем использовать уравнение окружности $x^2 + y^2 = R^2$. Так как R=1, то уравнение принимает вид $x^2+y^2 = 1$.
2. Определяем условие, при котором точка будет находиться внутри заштрихованной области: (x>0.5) и (x^2+y^2<1).
3. Определяем условие, при котором точка будет находиться вне заштрихованной области: (x<0.5) или (x^2+y^2>1).

Составим блок-схему решения задачи (рисунок 4).

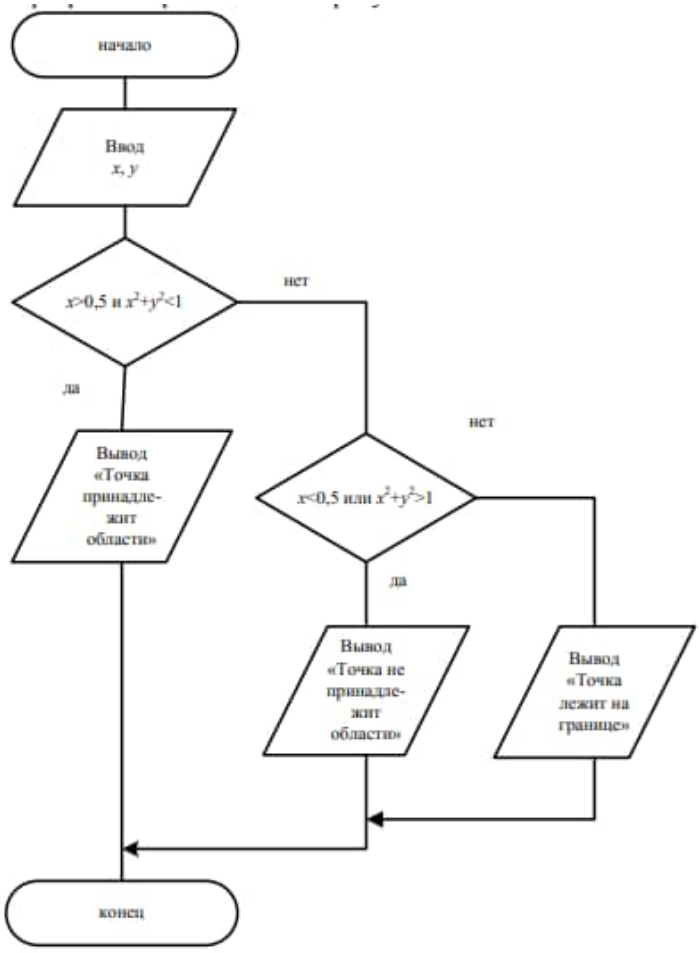


Рисунок5

Варианты заданий

Задание 1 Вычислить значение функции

1 $y = \arccos\ ^{2} A - B \lg(A^{2} + B^{2})$ при A = 5 B = 2,35

2 $Y = \lg^{3}(A/B + \mathrm{tg}A)$ при $A = 4$; $B = 13{,}6$•

3 Y = (sin3 c • cos2 a)/ arcsin d3+ 5sin d b при A = 9.5; B = 1,365; C = 6,6; D = 3

4 Y = 2A • arcsin(A2+B3) при A = 8; B = 5,6 •

5 $Y = 2B \cdot \ln(A - B^{s}) - \sin(D\text{-}C2)$ при $A = 5{,}6$; $B = 2{,}8$; $D = 3$ $C = 3{,}6$;

6 y = sin 2a + sin 5a - sin3a /(cos a +1 - $2\sin^{2}2a$) при a = 4 71 •

7 $Y = 1 - \sin^{2} 2a + \cos 2a$ при a = 1,57 •

8 $Y = 4\cos^{a} \cos a + \cos 4a$ при a = 2,09•

9 $Y = \arccos(A - B) + tg(1\text{-}B) \cdot \lg^{3}(A^{2}/B^{2})$ при $A = 5$, $B = 2{,}35$ •

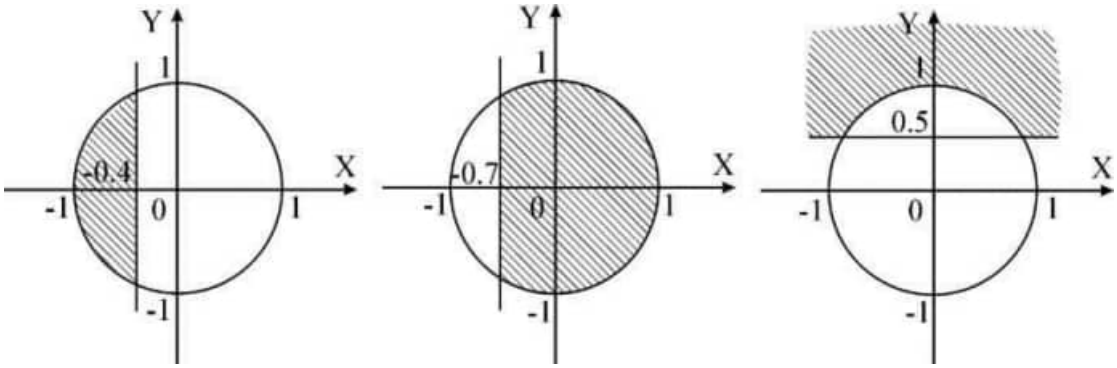
10 $Y = \lg(A/B + \mathrm{tg}A) - cos(A+B)$ при $A = 4$; $B = 13{,}6$ •

Задание 2: По заданным координатам точки определить, где находится точка:

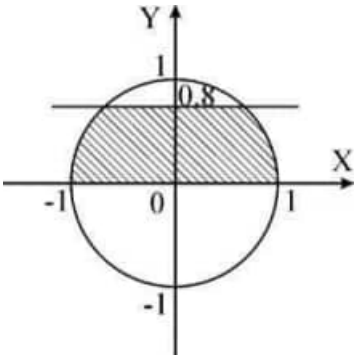
1) внутри заштрихованной области;

2) вне заштрихованной области;

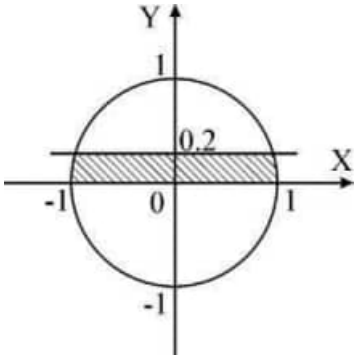
3) на границе этой области.



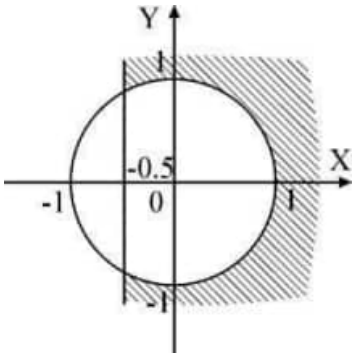
1. 2. 3



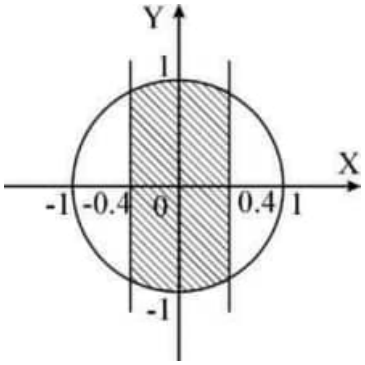
4.



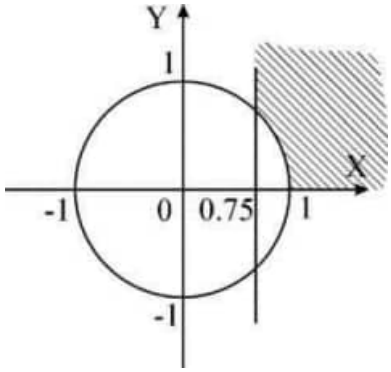
5.



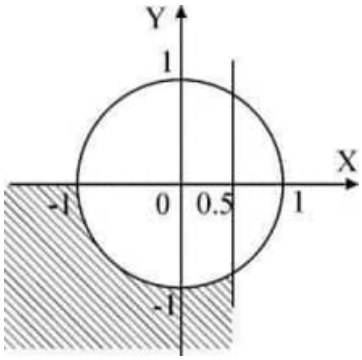
6.



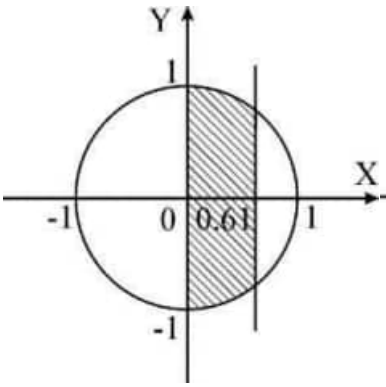
7.



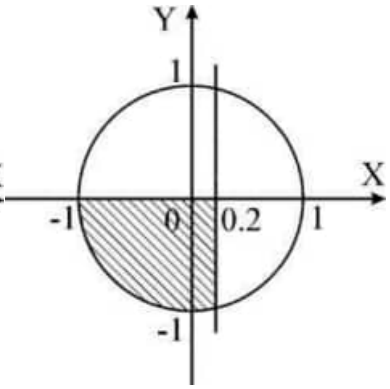
8.



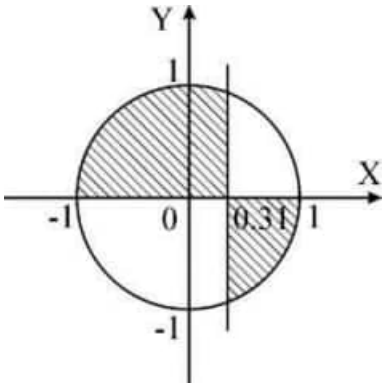
9.



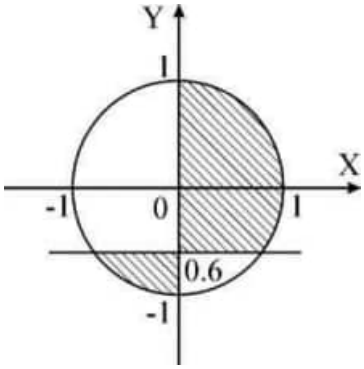
10.



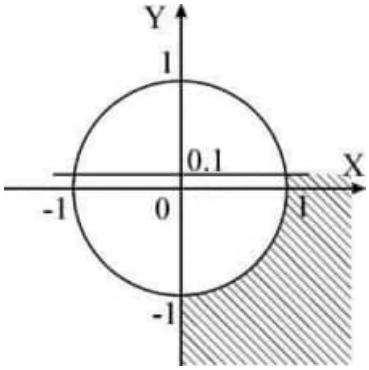
11.



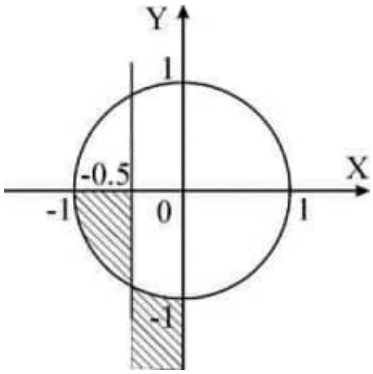
12.



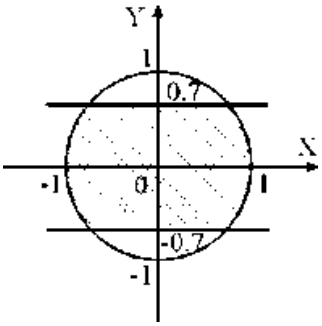
13.



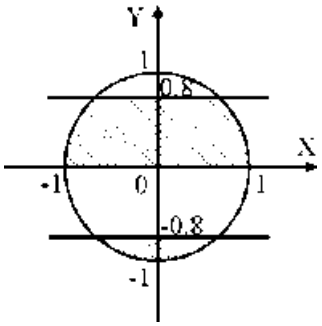
14.



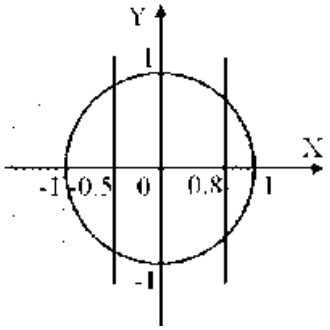
15.



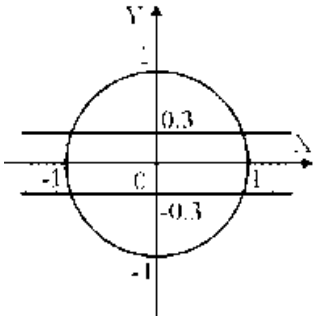
16.



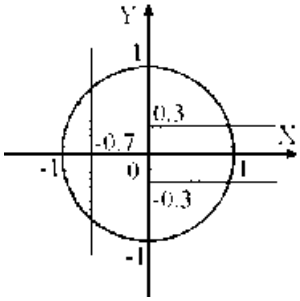
17.



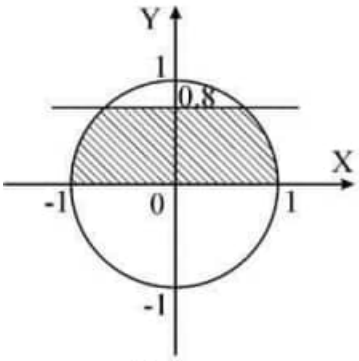
18.



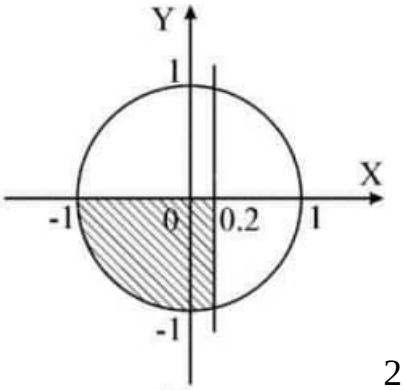
19.



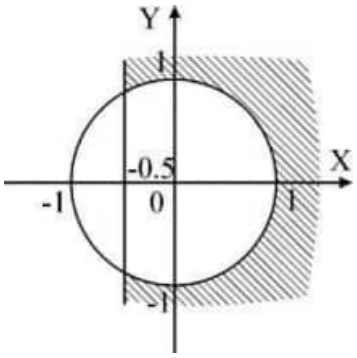
20.



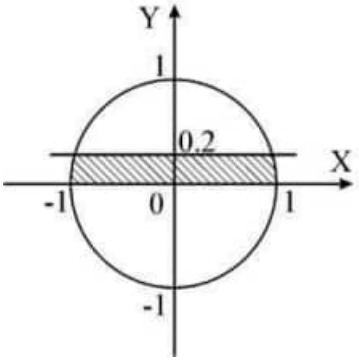
21.



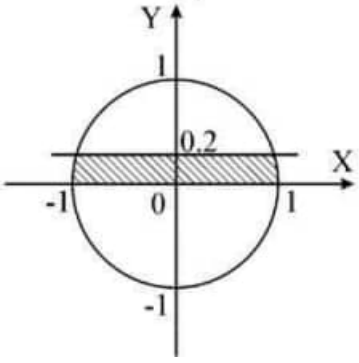
22.



23.



24.



25.

Практическое занятие № 2
Разработка циклических алгоритмов

Цель работы: научиться составлять одну из базовых структор алгоритмов – циклические алгоритмы

Краткие теоретические сведения

Цикл – процесс повторения одних и тех же операций (группы команд).

Тело цикла–это последовательность операций (команд), которая повторяется многократно заданное количество раз или до тех пор, пока не будет выполнено условие. Существуют три схемы представления циклических алгоритмов.

Рисунок 1

Цикл с предусловием

Особенностью первой схемы (цикл с предусловием) является то, что проверка условия проводится до тела цикла. В том случае, если условие выхода из цикла выполняется, то тело цикла не выполняется ни разу. Например, по условию необходимо вывести все точки графика $y=x^2$ на отрезке для x от -5 до 5 с шагом 1 сантиметр. Алгоритм решения задачи может быть представлен блок-схемой (рисунок 2).

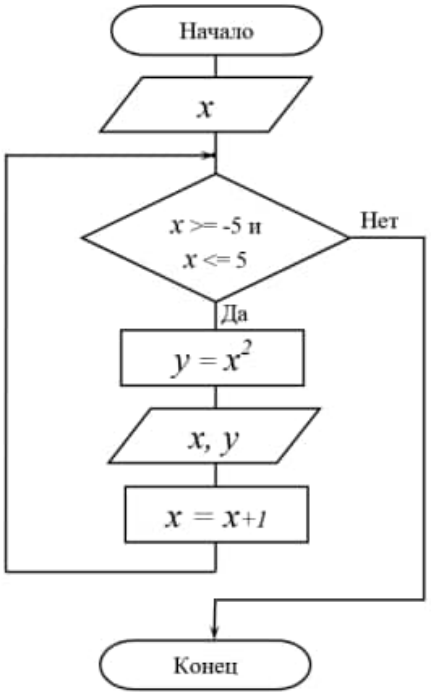


Рисунок 2

По представленному алгоритму с предусловием, если с клавиатуры будет введено стартовое значение хне удовлетворяющее условию $-5 \leq x \geq 5$, то решение задачи закончится после первого же неверного шага.

Цикл с постусловием

При решении задачи по циклическому алгоритму с постусловием в любом случае цикл будет выполнен хотябы один раз, так как первая проверка условия выхода из цикла осуществляется после того, как тело цикла выполнено. Например, блок-схема циклического алгоритма с постусловием для решения той же выше поставленной задачи представлена на рисунке 3.

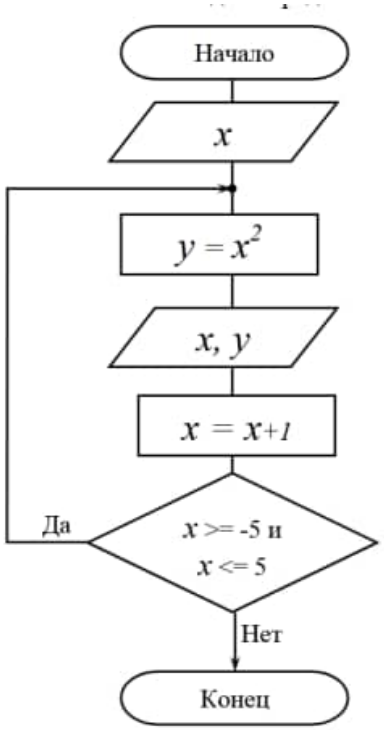


Рисунок 3

На практике обе схемы равноценны по применению частота и логичность их применения зависит от условия решаемой задачи.

Цикл с параметром

В условии многих задач четко оговаривается количество повторений операций тела цикла или, например, задано выполнить действия для всех значений переменной от первого до последнего с заданным шагом изменении.

Параметр цикла – это переменная, через которую организуется (описывается) цикл. Параметр цикла определяет количество повторений (итераций), так как у него обязательно известно - первое значение, при котором тело цикла будет выполнено первый раз, последнее значение, при котором тело цикла будет выполнено последний раз и величина изменения параметра (шаг), на которую надо увеличить параметр после каждой итерации. Шаг – числовой показатель изменения параметра после каждой итерации. Циклы с параметром более просты в понимании и организации. В блок–схеме выделена специальная фигура для обозначения цикла с параметром. В блоке через «;» прописываются все значения параметра - «первое; последнее; шаг». Применительно к решению выше поставленной задачи, параметром цикла, по условию, является *x*.

Для всех значений x от -5 до 5 через 1 сантиметр рассчитывается $y=x^2$ и выводится точка графика (координаты x, y). Блок-схема решения задачи с применением цикла с параметром, представлена на рисунке 4.

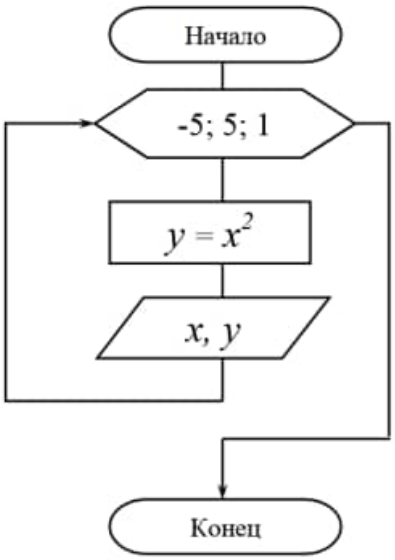


Рисунок 4

Варианты заданий

1 Составьте программу для вычисления степени числа *a* с целым показателем n.
2 Вычислите сумму всех двухзначных чисел.
3 Выведите на экран все чётные числа от 2 до n включительно и подсчитайте их количество.
4 Вычислите значение функции y=cos(x)*tg(pi*x) на интервале [0; pi), где шаг изменения равен 0.1.
5 Имеется товар в ящиках по 16, 17 и 21 кг. Как получить 185 кг этого товара, не вскрывая ящиков?
6 Найдите все делители некоторого целого числа.
7 Определите 5 первых совершенных чисел. Натуральное число называется совершенным, если оно равно сумме всех своих делителей за исключением самого себя. Например, 6 – совершенное число, т.к. 6=1+2+3, число 8 – несовершенное, т.к. 8<>1+2+4).
8 Найти сумму чётных натуральных чисел, меньших 100.
9 Найти сумму нечётных натуральных чисел, меньших 100.
10 Найти сумму целых положительных чисел, больших 20, меньших 100 и кратных 3.
11 Вычисление F = 10! выполнить каждым из трёх операторов цикла.
12 Начав тренировки, спортсмен в первый день пробежал 10 км, каждый последующий день он увеличивал дневную норму на 10% от нормы предыдущего дня. Какой суммарный путь пробежит спортсмен за 7 дней?
13 Самолёт летит из пункта А в пункт Б со средней скоростью υ. Найти время полёта t, если есть встречный ветер, скорость которого $\upsilon_в$ изменяется от 0 до 15 м/с с шагом $\Delta\upsilon_в$= 0.5 м/с. Расстояние между пунктами А и Б S= 437 км, υ= 720 км/ч.(Для тех, кто плохо знает физику: $t=S/(\upsilon-\upsilon_в)$)

14 Составить программу, печатающую таблицу значений градусов температуры по Цельсию и Фаренгейту. Значения градусов температуры по Цельсию изменяются от 0˚до 20˚с шагом 1˚. (℉ = ℃ * 1.8 + 32)

15 Пользователь сберегательной кассы внёс в неё вклад S_0= 3000 руб. До какой суммы он возрастёт через N= 5 лет, если процент годовых начислений P= 30 % ?

16 Даны действительные числа x, y. Вывести в порядке возрастания все целые числа, расположенные между x и y, а также количество этих чисел.

17 Даны действительные числа x, y. Вывести в порядке убывания все целые числа, расположенные между x и y, а также количество этих чисел

18 . Даны действительные числа x, y. Найти произведение всех целых чисел, расположенных между x и y, а также количество этих чисел.

19 Даны действительные числа x, y. Найти сумму квадратов всех целых чисел, расположенных между x и y, а также количество этих чисел.

20 Даны действительные числа x, y. Найти сумму кубов всех целых чисел, расположенных между x и y, а также количество этих чисел.

Практическое занятие № 3

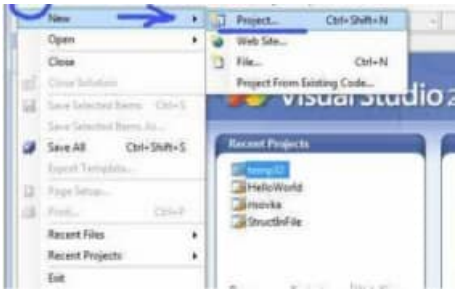
Знакомство с инструментальной средой программирования

Цель работы: научиться создавать консольные приложения

Microsoft Visual Studio — линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данные продукты позволяют разрабатывать как консольные приложения, так и приложения с графическим интерфейсом.

Создание проекта и добавление исходного файла состоит из следующих шагов.

1 Создание проекта, последовательно выбрав меню Файл пункты Создать_и_Проект (File-New-Project).



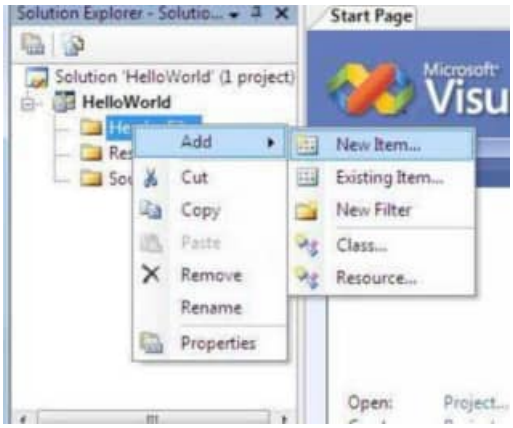
2 В области типов проектов Visual C++ выберите группу Win32 и щелкните элемент Консольное приложение (Win32 Console Application).

3 Введите имя проекта.

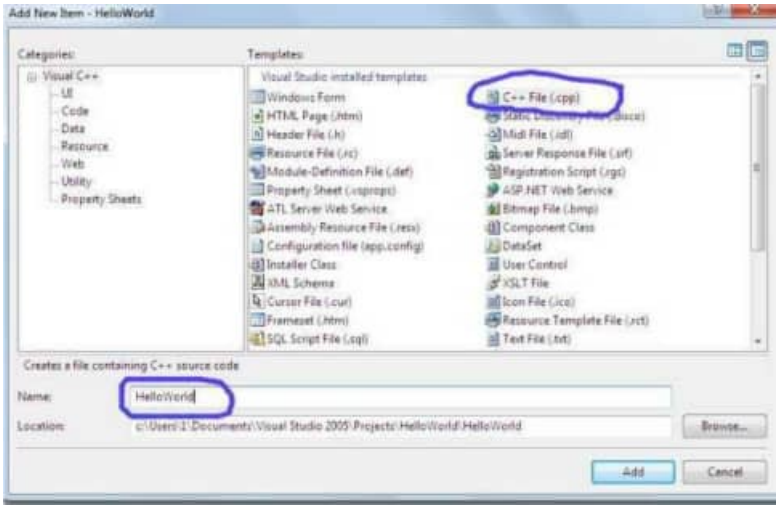
По умолчанию имя решения, содержащего проект, совпадает с именем проекта, однако можно ввести другое имя. Также можно указать другое расположение для проекта. Нажмите кнопку ОК, чтобы создать проект.

4 В мастере приложений Win32 нажмите кнопку Далее (Next), выберите вариант Пустой проект (EmptyProject)и нажмите кнопку Готово (Finish).

5 Далее появилось окно Solution Explorer, в котором должно быть название вашего проекта. Если у вас Solution Explorer не появляется, то скорее всего надо нажать View-Solution Explorer (это в основном меню). Под названием проекта отображены значки трех папок. Нас интересует Header Files. — жмем на ней правой клавишей мыши открывается контекстное меню, где надо выбрать New Item



6 Дальше появляется новое окно, в котором выбираем C++ File (.cpp), вводим название проекта и нажимаем Add.



Если все правильно, то можно приступать к написанию первой программы
Написав текст программы и можно проверить её нажав ctrl+F5.

Каждое задание необходимо решить в соответствии с изученными методами обработки данных и преобразования типов данных в языке C++.
Следует реализовать каждое задание в соответствии с приведенными этапами:

1. изучить словесную постановку задачи, выделив при этом все виды данных;
2. сформулировать математическую постановку задачи;
3. выбрать метод решения задачи, если это необходимо;
4. разработать графическую схему алгоритма;
5. записать разработанный алгоритм на языке C++;
6. разработать контрольный тест к программе;
7. отладить программу;
8. представить отчет по работе.

Практическое занятие № 4
Разработка программ последовательной структуры

Цель работы: изучить классификацию типов и их внутреннее представление в языке С++, научиться работать со стандартными функциями. Изучить структуру программы на языке С++. Ознакомиться с программированием математических формул.

Краткие теоретические сведения.

Базовые типы – это типы данных, которые предопределены *стандартом языка*, указываются зарезервированными ключевыми словами, характеризуются одним значением и внутренним представлением.

Вещественный тип – это *базовый тип данных*, который применяется для хранения дробных чисел в формате с плавающей точкой.

Логический (булевый) тип – это *базовый тип данных*, который применяется для хранения значений двузначной логики.

Неявное приведение типа – это преобразование значения переменной к новому типу, которое происходит автоматически, по правилам, заложенным в языке программирования.

Перечисляемый тип – это *производный тип данных*, он определяется как набор идентификаторов, являющихся именованными целыми константами, которым приписаны уникальные обозначения

Преобразование типов – это приведение значения переменной одного типа в *значение* другого типа.

Производные типы – это типы данных, которые задаются пользователем.

Символьный тип – это *базовый тип данных*, который применяется для хранения символов или *управляющих последовательностей* в виде кода.

Тип данных – это множество допустимых значений, которые может принимать тот или иной *объект*, а также множество допустимых операций, которые применимы к нему.

Типы класса – это типы данных, экземплярами которых являются объекты.

Целочисленный тип – это *базовый тип данных*, который применяется для хранения целых чисел.

Явное приведение типа – это преобразование значения переменной к новому типу, при котором указывается *тип переменной*, к которому необходимо привести исходную переменную.

1. Для организации хранения данных и корректного выполнения операций над ними в языках программирования определены типы данных.
2. Типы характеризуются схожим внутренним представлением данных в памяти компьютера; объемом памяти, выделяемым под данные; множеством (диапазоном) принимаемых значений; допустимыми операциями и функциями.
3. В языке С++ типы классифицируются на базовые, производные и классы.
4. Для базовых типов определены их подмножества и расширения, что обеспечивает повышение точности расчетов или экономный расход памяти.
5. Над типами данных определена операция преобразования типов. Ее следует применять с осторожностью при переходе к типу, у которого меньше по модулю границы диапазонов.

Варианты заданий

При выполнении лабораторной работы для каждого задания требуется написать программу на языке С++, которая получает на входе числовые данные, выполняет их обработку в соответствии с требованиями задания и выводит результат на экран. Ввод данных осуществляется с клавиатуры с учетом требований к входным данным, содержащихся в постановке задачи. Ограничениями на *входные данные* является допустимый*диапазон* значений используемых *числовых типов* в языке С++.

1. Найдите сумму первых трех цифр дробной части *вещественного числа*. Например, для числа 23,16809 она равна 15.

2. Составьте программу вычисления стоимости поездки на автомобиле на дачу (туда и обратно). Исходными данными являются: расстояние до дачи (в километрах); количество бензина, которое потребляет автомобиль на 100 км пробега; цена одного литра бензина. Ниже представлен рекомендуемый вид диалога во время работы программы.
    1. Вычисление стоимости поездки на дачу.
    2. Расстояние до дачи (км) – 67
    3. Расход бензина (л на 100 км) – 8.5
    4. Цена литра бензина (руб.) – 23.7
    5. Поездка на дачу обойдется в 269 руб. 94 коп.
3. Составьте *линейную программу*, печатающую значение 1, если указанное *высказывание* является истинным, и 0 – в противном случае. Величина d является корнем только одного из уравнений $ax^2 + bx + c = 0$ и $mx + n = 0$ относительно x.
4. Составьте программу, которая преобразует введенное с клавиатуры дробное число в денежный формат. Например, число 12,348 должно быть преобразовано к виду 12 руб. 35 коп. Ниже представлен рекомендуемый вид диалога во время работы программы. Данные, вводимые пользователем:
    1. Преобразование числа в денежный формат.
    2. Введите дробное число – 23,6
    3. 23.6 руб. – это 23 руб. 60 коп.

Указания к выполнению работы

Каждое задание необходимо решить в соответствии с изученными методами обработки данных и преобразования типов данных в языке C++.

Следует реализовать каждое задание в соответствии с приведенными этапами:

9. изучить словесную постановку задачи, выделив при этом все виды данных;
10. сформулировать математическую постановку задачи;
11. выбрать метод решения задачи, если это необходимо;
12. разработать графическую схему алгоритма;
13. записать разработанный алгоритм на языке C++;
14. разработать контрольный тест к программе;
15. отладить программу;
16. представить отчет по работе.

Требования к отчету

Отчет по лабораторной работе должен соответствовать следующей структуре.

1. Титульный лист.
2. Словесная постановка задачи. В этом подразделе проводится полное описание задачи. Описывается суть задачи, анализ входящих в нее физических величин, область их допустимых значений, единицы их измерения, возможные ограничения, анализ условий при которых задача имеет решение (не имеет решения), анализ ожидаемых результатов.
3. Математическая модель. В этом подразделе вводятся математические описания физических величин и математическое описание их взаимодействий. Цель подраздела – представить решаемую задачу в математической формулировке.
4. Алгоритм решения задачи. В подразделе описывается разработка структуры алгоритма, обосновывается абстракция данных, задача разбивается на подзадачи. Схема алгоритма выполняется по ЕСПД (ГОСТ 19.003-80 и ГОСТ 19.002-80).

5. Листинг программы. Подраздел должен содержать текст программы на языке программирования C++, реализованный в среде MS Visual Studio 2010.
6. Контрольный тест. Подраздел содержит наборы исходных данных и полученные в ходе выполнения программы результаты.
7. Выводы по лабораторной работе.
8. Ответы на контрольные вопросы.

Контрольные вопросы

1. Почему в языке C++ определена строгая типизация данных, используемых в программе?
2. Как определяются границы диапазона базового типа в зависимости от выделяемой под этот тип памяти?
3. С какой целью в C++ определен тип void?
4. Какой объем памяти выделяется под переменную типа void? Какие значения может принимать переменная типа void?
5. Почему наблюдается *асимметрия* значений границ диапазонов целочисленных типов?
6. Чему будет равно значение операции *инкремента* для максимального числа в целочисленном типе? А каков результат*декремента* для минимального значения в таком же типе?
7. Почему запись целых чисел нельзя начинать с незначащих нулей?
8. Каким образом представлено число ноль в вещественных типах?
9. Почему в C++ *символьный тип* считается подмножеством целочисленного типа?
10. Каким образом можно инициализировать переменную перечисляемого типа?
11. При преобразовании целого со знаком к целому без знака всегда ли будет получено исходное числовое значение? Ответ обоснуйте.

Практическое занятие № 5

Разработка программ разветвляющейся структуры

Цель работы: научиться писать приложения на языке с++, используя операторы ветвления

Оператор if

```
int main()
{
setlocale(LC_ALL, "Russian");
double num;
cout << "Введите произвольное число: ";
cin >> num;
if (num < 10) { // Если введенное число меньше 10.
cout << "Это число меньше 10." << endl;
} else { // иначе
cout << "Это число больше либо равно 10." << endl;
}
return 0;
}
```

Если вы запустите эту программу, то при вводе числа, меньшего десяти, будет выводиться соответствующее сообщение. Если введенное число окажется большим, либо равным десяти — отобразится другое сообщение.

Оператор if служит для того, чтобы выполнить какую-либо операцию в том случае, когда условие является верным. Условная конструкция в С++ всегда записывается в круглых скобках после оператора if. Внутри фигурных скобок указывается тело условия. Если условие выполнится, то начнется выполнение всех команд, которые находятся между фигурными скобками.

Оператор переключатель switch

Оператор выбора имеет следующий синтаксис

```
switch выражение_целого_типа
{
case значение_1: оператор1;break;
...
case значение_n: оператор_n;break;
default: оператор_d;
}
```

Выполняется оператор следующим образом.

Вычисляется выражение, записанное как условие.

Если значение условия равно значение_i , то выполняется оператор_i и далее - выход из оператора switch.

Если значение условия не равно ни одному значение_i , то выполняется оператор_d .

Пример 1.

```
int x=2;
switch x
{
case 1: cout<<1<<endl;break;
case 2: cout<<2<<endl;break;
default: cout<<”x not equal 1 or 2”<<endl;
}
```

Ввод-вывод в С++

Обмен данными между программой и внешними устройствами осуществляется с помощью операций ввода-вывода. Типичным внешним устройством является консоль.

Другим типичным устройством является жесткий диск, на котором расположены *файлы*. Программа может создавать *файлы*, в которых хранится информация. Другая (или эта же) программа может читать информацию из *файла*.

В языке C++ нет особых операторов для ввода или вывода данных. Вместо этого имеется набор классов, стандартно поставляемых вместе с компилятором, которые и реализуют основные операции ввода-вывода.

Причиной является как слишком большое разнообразие операций ввода и вывода в разных операционных системах, особенно графических, так и возможность определения новых типов данных в языке C++. Библиотека классов для ввода-вывода решает две задачи. Во-первых, она обеспечивает эффективный ввод-вывод всех встроенных типов и простое, но тем не менее гибкое, определение операций ввода-вывода для новых типов, разрабатываемых программистом. Во-вторых, сама библиотека позволяет при необходимости развивать её и модифицировать.

Потоки. Механизм для ввода-вывода в C++ называется ***потоком***. Название произошло от того,что информация вводится и выводится в виде ***потока*** байтов – символ за символом.

Класс istream реализует *поток* ввода, класс ostream – *поток* вывода. Эти классы определены в файле заголовков iostream.h. Библиотека *потоков* ввода-вывода определяет три глобальных объекта: cout, cin и cerr. cout называется стандартным выводом, cin – стандартным вводом, cerr – стандартным *потоком* сообщений об ошибках. cout и cerr выводят на консоль и принадлежат к классу ostream, cin имеет тип istream и вводит с консоли.

Вывод осуществляется с помощью операции <<, ввод с помощью операции >>. Выражение

```
cout << "Пример вывода: " << 34;
```

напечатает на строку "Пример вывода", за которым будет выведено число 34. Выражение

```
int x;
cin >> x;
```

введет целое число в переменную x.

Манипуляторы и форматирование ввода-вывода. Часто бывает необходимо вывести строку или число в определенном формате. Для этого используются так называемые ***манипуляторы***.

Манипуляторы – это объекты особых типов, которые управляют тем, как ostream или istream обрабатывают последующие аргументы. Некоторые *манипуляторы* могут также выводить или вводить специальные символы.

endl	при выводе перейти на новую строку;
dec	выводить числа в десятичной системе (действует по умолчанию);
oct	выводить числа в восьмеричной системе;
hex	выводить числа в шестнадцатиричной системе счисления;
setw (int n)	установить ширину поля вывода в n символов (n – целое число);

Использовать *манипуляторы* просто – их надо вывести в выходной *поток*. Предположим, мы хотим вывести одно и то же число в разных системах счисления:

```
int x = 53;
cout << "Десятичный вид: " << dec
 << x << endl
 << "Восьмиричный вид: " << oct
 << x << endl
 << "Шестнадцатиричный вид: " << hex
 << x << endl
```

Аналогично используются *манипуляторы* с параметрами. Вывод числа с разным количеством цифр после запятой:

```
double x;
// вывести число в поле общей шириной
// 6 символов (3 цифры до запятой,
// десятичная точка и 2 цифры после запятой)
cout << setw(6) << setprecision(2) << fixed << x << endl;
```

Те же *манипуляторы* (за исключением endl) могут использоваться и при вводе. В этом случае они описывают представление вводимых чисел.

```
int x;
// ввести шестнадцатиричное число
cin >> hex >> x;
```

Поля вывода. Функция **width()** устанавливает минимальное число символов, использующееся в последующей операции вывода числа или строки. Так в результате следующих операций

```
cout.width(4);
cout << '(' << 12 << ')'; //получим число 12 в поле размером 4 символа.
```

Функция width() задает минимальное число символов. Если появится больше символов, они будут напечатаны все, поэтому

```
cout.width(4);
cout << '(' << "121212" << ")\n";
```

напечатает (121212)

Причина, по которой разрешено переполнение поля, а не усечение вывода,в том, чтобы избежать зависания при выводе.

3. Порядок выполнения работы

1. Ознакомиться с теоретическими сведениями.
2. Получить вариант задания у преподавателя.
3. Создание проекта. В строке меню выберите Файл> Создать> Проект. В левой области диалогового окна Новый проект последовательно разверните узлы Установленные шаблоны и Visual C++, а затем щелкните Win32. В списке установленных шаблонов в центральной области выберите шаблон Консольное приложение Win32. В поле Имя введите имя проекта. Вы можете принять расположение по умолчанию в раскрывающемся списке Расположение, ввести другое расположение или нажать кнопку Обзор, чтобы перейти к каталогу, в котором требуется сохранить проект. Во время создания проекта среда Visual Studio помещает проект в решение. По умолчанию имя решения совпадает с именем проекта. Можно изменить имя в поле *Имя решения*, но для этого примера оставьте имя по умолчанию. На странице *Обзор* окна *Мастер приложений Win32* нажмите кнопку *Далее*. На странице *Параметры приложения* выберите в поле *Тип приложения* пункт *Консольное приложение***.** Нажмите кнопку *Готово*, чтобы создать проект. Добавьте код:

```
#include "stdafx.h"
#include <iostream>
#include <cmath> // Работа с математическими функциями
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
  setlocale(LC_ALL, "Russian");
  .
  .
```

```
.
cin.get();
return 0;}
```

4. Выполнить задание по варианту.
5. Продемонстрировать выполнение работы преподавателю.
6. Оформить отчет.

Контрольные вопросы

1. Что такое решение (solution) в Visual Studio .NET? Зачем они нужны?
2. Как создать решение? Как добавить туда проект?
3. Что такое список задач? Как можно добавить задачу?
4. Зачем нужно окно свойств?
5. Зачем нужно окно вывода и окно ошибок? Чем они отличаются?
6. Зачем нужно окно решения?
7. Как добавить в проект новый или существующий файл.
8. Какой синтаксис имеет цикл while?.
9. Какой синтаксис имеет цикл do-while?.
10. Какой синтаксис имеет цикл for?
11. Для чего используется оператор break?
12. Каково назначение оператора continue?

Варианты заданий.

1. Даны три действительных числа. Возвести в квадрат те из них, значения которых неотрицательны, и в четвертую степень — отрицательные.

2. Даны две точки $A(x_1, y_1)$ и $B(x_2,y_2)$. Составить алгоритм, определяющий, которая из точек находится ближе к началу координат.

3. Даны два угла треугольника (в градусах). Определить, существует ли такой треугольник, и если да, то будет ли он прямоугольным.

4. Даны действительные числа x и y, не равные друг другу. Меньшее из этих двух чисел заменить половиной их суммы, а большее — их удвоенным произведением.

5. На плоскости XOY задана своими координатами точка A. Указать, где она расположена (на какой оси или в каком координатном угле).

6. Даны целые числа m, n. Если числа не равны, то заменить каждое из них одним и тем же числом, равным большему из исходных, а если равны, то заменить числа нулями.

7. Подсчитать количество отрицательных среди чисел a, b, c.

8. Подсчитать количество положительных среди чисел a, b, c.

9. Услуги телефонной сети оплачиваются по следующему правилу: за разговоры до A минут в месяц — B руб., а разговоры сверх установленной нормы оплачиваются из расчета C руб. за минуту. Написать программу, вычисляющую плату за пользование телефоном для введенного времени разговоров за месяц.

13. Грузовой автомобиль выехал из одного города в другой со скоростью v_1 км/ч. Через t ч в этом же направлении выехал легковой автомобиль со скоростью v_2 км/ч. Составить программу, определяющую, догонит ли легковой автомобиль грузовой через t_1 ч после своего выезда.

14. Перераспределить значения переменных x и y так, чтобы в x оказалось большее из этих значений, а в y - меньшее.

15. Определить правильность даты, введенной с клавиатуры (число — от 1 до 31, месяц — от 1 до 12). Если введены некорректные данные, то сообщить об этом.

16. Составить программу, определяющую результат гадания на ромашке — «любит—не любит», взяв за исходное данное количество лепестков n.

17. Написать программу — модель анализа пожарного датчика в помещении, которая выводит сообщение «Пожароопасная ситуация», если температура в комнате превысила 60° С.

18. Рис расфасован в два пакета. Масса первого — n кг, второго — m кг. Составить программу, определяющую:

а) какой пакет тяжелее — первый или второй;

б) массу более тяжелого пакета.

19. Написать программу, которая анализирует данные о возрасте и относит человека к одной из четырех групп: дошкольник, ученик, работник, пенсионер. Возраст вводится с клавиатуры.

20. Составить программу, определяющую, пройдет ли график функции $y = ax^2 + bx +$ с через заданную точку с координатами (m, n).

21. К финалу конкурса лучшего по профессии «Специалист электронного офиса» были допущены трое: Иванов, Петров, Сидоров. Соревнования проходили в три тура. Иванов в первом туре набрал m_1 баллов, во втором — n_1, в третьем — p_1. Петров — m_2, n_2, p_2 соответственно; Сидоров — m_3, n_3 , p_3. Составить программу, определяющую, сколько баллов набрал победитель.

Практические занятия № 6,7,8

Разработка программ с использованием цикла с предусловием.

Разработка программ с использованием цикла с постусловием.

Разработка программ с использованием цикла с параметром

Цель работы: научиться писать приложения на языке с++, используя операторы цикла

Оператор цикла while

Для организации повторения, управляемого счетчиком, требуется задать:

– Имя управляющей переменной (или счетчика циклов).

– Приращение (или уменьшение), на которое изменяется управляющая переменная в каждом цикле.

– Условие проверки, не достигнуто ли конечное значение управляющей переменной (т.е. надо ли продолжать циклы).

Оператор цикла while организует цикл с предусловием, т.е. проверка условия окончания цикла происходит в начале цикла.

Рассмотрим простую программу, которая выводит на консоль числа от 1 до 10:

```
// повторение, управляемое счетчиком
#include <iostream>
main() {
        int counter = 1; // задание начального значения
        while(counter <= 10) { // условие повторения
                count << counter << endl;
                ++counter; // увеличение
        }
return 0;
}
```

Вывод.

1 2 3 4 5 6 7 8 9 10

Объявление int counter = 1; определяет имя управляющей переменной (counter), объявляет переменную как целую, резервируя для нее соответствующее место в памяти, и задает ее начальное значение 1. Объявления, в которых задается начальное значение, являются фактически выполняемыми операторами. Это объявление и задание начального значения переменной counter можно было бы сделать операторами int counter; counter = 1;

Здесь само объявление не является выполняемым оператором, а присваивание осуществляется отдельным оператором. Можно использовать оба эти метода для задания начальных значений переменных. Оператор ++counter увеличивает счетчик на 1 при каждом выполнении цикла. Условие продолжения цикла в структуре while проверяет, меньше или равно значение управляющей переменной числа 10 (последнего значения, при котором условие истинно). Заметьте, что тело этой структуры while выполняется и при значении управляющей переменной, равном 10. Выполнение цикла заканчивается, когда значение управляющей переменной превысит 10 (т.е. переменная counter станет равной 11).

Программа может быть сделана более компактной, если переменной counter задать начальное значение 0 и заменить структуру while следующей:

```
while(++counter <= 10)
        cout << counter << endl;
```

Этот код экономит один оператор, поскольку инкремент выполняется непосредственно в условии структуры while до того, как это условие проверяется. Этот код также устраняет скобки, заключающие в себе тело while, поскольку теперь while содержит всего один оператор.

Оператор цикла do/while

Структура повторения do/while похожа на структуру while. В структуре while условие продолжения циклов проверяется в начале цикла, до того, как выполняется тело цикла. В структуре do/while проверка условия продолжения циклов производится после того, как тело цикла выполнено, следовательно, тело цикла будет выполнено по крайней мере один раз. Когда do/while завершается, выполнение программы продолжается с оператора, следующего за предложением while.

Таки образом оператор цикла do/while организует цикл с постусловием.

Отметим, что в структуре do/while нет необходимости использовать фигурные скобки, если тело состоит только из одного оператора. Но фигурные скобки обычно все же ставят, чтобы избежать путаницы между структурами while и do/while. Например,

while (условие)

обычно рассматривается как заголовок структуры while. Структура do/while без фигурных скобок и с единственным оператором в теле имеет вид

do

оператор while(условие);

что может привести к путанице. Последняя строка — while (условие); может ошибочно интерпретироваться как заголовок структуры while, содержащий пустой оператор. Таким образом, чтобы избежать путаницы, структура do/while даже с одним оператором часто записывается в виде

do {

оператор } while (условие);

Некоторые программисты всегда включают фигурные скобки в структуру do/while, даже если в них нет необходимости. Это помогает устранить двусмысленность, проистекающую из совпадения предложений структуры while и структуры do/while, содержащей один оператор.

Следующая программа использует структуру do/while, чтобы напечатать числа от 1 до 10. Обратите внимание, что к управляющей переменной counter в проверке окончания цикла применяется инкремент в префиксной форме. Отметьте также использование фигурных скобок, заключающих единственный оператор в теле do/while.

```
// Применение структуры повторения do/while
int main() {
int counter = 1;
do {
cout << counter << " ";
} while(++counter <= 10);
return 0;
}
```

Оператор цикла for

Структура повторения for содержит все элементы, необходимые для повторения, управляемого счетчиком, т.е. параметром. Чтобы проиллюстрировать мощь структуры for, перепишем программу.

```
// Повторение, управляемое счетчиком, со структурой for
int main() {
        // задание начального значения,условие повторения и
        // приращение – включено в заголовок структуры for
        for(int counter = 1; counter <= 10; counter++)
                cout << counter << endl;
        return 0;
```

}

Когда структура for начинает выполняться, управляющей переменной counter задается начальное значение 1. Затем проверяется условие продолжения цикла counter <= 10. Поскольку начальное значение counter равно 1, это условие удовлетворяется, так что оператор тела структуры печатает значение counter, равное 1. Затем управляющая переменная counter увеличивается на единицу в выражении counter++ и цикл опять начинается с проверки условия его продолжения. Поскольку значение counter теперь 2, предельная величина не превышена, так что программа снова выполняет тело цикла. Этот процесс продолжается, пока управляющая переменная counter не увеличится до 11 — это приведет к тому, что условие продолжения цикла нарушится и повторение прекратится. Выполнение программы продолжится с первого оператора, расположенного после структуры for (в данном случае с оператора return в конце программы).

Общая форма структуры for: for (выражение1; выражение2; выражение3) оператор, где выражение1 задает начальное значение переменной, управляющей циклом, выражение2 является условием продолжения цикла, а выражение3 изменяет управляющую переменную. В большинстве случаев структуру for можно представить эквивалентной ей структурой while следующим образом:

```
Выражение1;
while(Выражение2) {
        оператор
        Выражение3;
        }
```

«Приращение» структуры for может быть отрицательным (в этом случае в действительности происходит не приращение, а уменьшение переменной, управляющей циклом). Если условие продолжения цикла с самого начала не удовлетворяется, то операторы тела структуры for не выполняются и управление передается оператору, следующему за for. Управляющая переменная иногда печатается или используется в вычислениях в теле структуры for, но обычно это делается без изменений ее величины. Чаще управляющая переменная используется только для контроля числа повторений и никогда не упоминается в теле структуры. Хотя управляющая переменная может изменяться в теле цикла for, избегайте делать это, так как такая практика приводит к неявным, неочевидным ошибкам.

Операторы break и continue

Операторы break и continue изменяют поток управления. Когда оператор break выполняется в структурах while, for, do/while или switch, происходит немедленный выход из структуры. Программа продолжает выполнение с первого оператора после структуры. Обычное назначение оператора break — досрочно прерывать цикл или пропустить оставшуюся часть структуры switch.

Оператор continue в структурах while, for или do/while вызывает пропуск оставшейся части тела структуры и начинается выполнение следующей итерации цикла. В структурах while и do/while немедленно после выполнения оператора continue производится проверка условия продолжения цикла. В структуре for выполняется выражение приращения, а затем осуществляется проверка условия продолжения.

Оператор переключатель switch

Оператор выбора имеет следующий синтаксис

```
switch выражение_целого_типа
{
case значение_1: оператор1;break;
...
case значение_n: оператор_n;break;
default: оператор_d;
```

}

Выполняется оператор следующим образом.

Вычисляется выражение, записанное как условие.

Если значение условия равно значение_i , то выполняется оператор_i и далее - выход из оператора switch.

Если значение условия не равно ни одному значение_i , то выполняется оператор_d .

Пример 1.

```
int x=2;
switch x
{
case 1: cout<<1<<endl;break;
case 2: cout<<2<<endl;break;
default: cout<<"x not equal 1 or 2"<<endl;
}
```

Варианты заданий

1. Имеется серия измерений элементов треугольника. Группы элементов пронумерованы. В серии в произвольном порядке могут встречаться такие группы элементов треугольника:

1) основание и высота;

2) две стороны и угол между ними (угол задан в радианах);

3) три стороны.

Разработать программу, которая запрашивает номер группы элементов, вводит соответствующие элементы и вычисляет площадь треугольника. Вычисления прекратить, если в качестве номера группы введен 0.

2. Начав тренировки, спортсмен в первый день пробежал 10 км. Каждый день он увеличивал дневную норму на 10% нормы предыдущего дня. Какой суммарный путь пробежит спортсмен за 7 дней?

3. Одноклеточная амеба каждые 3 часа делится на 2 клетки. Определить, сколько амеб будет через 3, 6, 9, 12,..., 24 часа.

4. Около стены наклонно стоит палка длиной *x* м. Один ее конец находится на расстоянии *y* м от стены. Определить значение угла *a* между палкой и полом для значений *x*=к м и *y*, изменяющегося от 2 до 3 м с шагом h м.

5. У гусей и кроликов вместе 64 лапы. Сколько может быть кроликов и гусей (указать все сочетания)?

6. Сколько можно купить быков, коров и телят, платя за быка 10 т. руб., за корову — 5 т. руб., а за теленка — 0,5 т.руб., если на 100 т.руб. надо купить 100 голов скота?

8. Составить программу для проверки утверждения: «Результатами вычислений по формуле x^2 +x+ 17 при 0<=x< =15 являются простые числа». Все результаты вывести на экран.

9. Составить программу для проверки утверждения: «Результатами вычислений по формуле x^2 + x + 41 при 0<x<=40 являются простые числа». Все результаты вывести на экран.

10. Составить программу-генератор простых чисел, в основу положить формулу $2x^2$ + 29 при 0 <= x <= 28.

12. Составить программу-генератор чисел Пифагора a, b, c ($c^2=a^2+b^2$). В основу положить формулы: $a = m^2 - n^2$, b = 2mn, $c = m^2+ n^2$ (m, n - натуральные, 1 < m< к, 1<n< к, к — данное число). Результат вывести на экран в виде таблицы из пяти столбцов: m, n, a, b, c.

13. Покупатель должен заплатить в кассу 5 руб. У него имеются купюры по 1, 5, 10, 50, 100, 500, 1000 и 10000 руб. Сколько купюр разного достоинства отдаст покупатель, если он начинает платить с самых крупных купюр?

14. Ежемесячная стипендия студента составляет А руб., а расходы на проживание превышают стипендию и составляют В руб. в месяц. Рост цен ежемесячно увеличивает расходы на 3%. Составьте программу расчета суммы денег, которую необходимо единовременно попросить у родителей, чтобы можно было прожить учебный год (10 месяцев), используя только эти деньги и стипендию.

15. Составить программу, которая печатает таблицу умножения и сложения натуральных чисел в десятичной системе счисления.

16. Составить программу, которая печатает таблицу умножения и сложения натуральных чисел в шестнадцатеричной системе счисления.

17. Найти сумму всех n-значных чисел (1 <= n < =4).

18. Найти сумму всех n-значных чисел, кратных к (1 <= n <= 4).

19. Заменить буквы цифрами так, чтобы соотношение оказалось верным (одинаковым буквам соответствуют одинаковые цифры, разным — разные):

ХРУСТ*ГРОХОТ = РРРРРРРРРРР.

20. Составить программу, которая запрашивает пароль (например, четырехзначное число) до тех пор, пока он не будет правильно введен.

21. Составить программу, которая находит наибольшее значение отношения трехзначного числа к сумме его цифр.

22. Составить алгоритм решения ребуса РАДАР = $(Р + А + Д)^4$ (различные буквы обозначают различные цифры, старшая — не 0).

23. Составить алгоритм решения ребуса МУХА + МУХА + МУХА = СЛОН (различные буквы обозначают различные цифры, старшая — не 0).

24. Составить алгоритм решения ребуса ДРУГ — ГУРД = 2727 (различные буквы обозначают различные цифры, старшая — не 0).

25. Составить алгоритм решения ребуса КОТ + КОТ = ТОК (различные буквы обозначают различные цифры, старшая — не 0).

Практическое занятие № 9
Разработка программ с использованием одномерных массивов

Цель работы: освоение средств языка С++ для изучения способов описания, ввода-вывода и обработки одномерных массивов.

Краткие теоретические сведения

Одномерный массив — массив, с одним параметром, характеризующим количество элементов одномерного массива. Фактически одномерный массив — это массив, у которого может быть только одна строка, и n-е количество столбцов. Столбцы в одномерном массиве — это элементы массива. На рисунке 1 показана структура целочисленного одномерного массива *a*. Размер этого массива — 16 ячеек.

5	-12	-12	9	10	0	-9	-12	-1	23	65	64	11	43	39	-15
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]	a[12]	a[13]	a[14]	a[15]

Рисунок 1— Одномерный массив

Максимальный индекс одномерного массива *a* равен 15, но размер массива 16 ячеек, потому что нумерация ячеек массива всегда начинается с 0. Индекс ячейки - это целое неотрицательное число, по которому можно обращаться к каждой ячейке массива и выполнять какие-либо действия над ней (ячейкой). Синтаксис объявления одномерного массива в С++:

тип данных имя одномерного массива [размерность одномерного массива];
//пример объявления одномерного массива, изображенного на рисунке 1:
int a[16];

где, int — целочисленный тип данных; а — имя одномерного массива; 16 — размер одномерного массива, 16 ячеек.

Всегда сразу после имени массива идут квадратные скобочки, в которых задаётся размер одномерного массива, этим массив и отличается от всех остальных переменных.

Рассмотрим на примерах способы ввода-вывода одномерного массива:

1) размер массива можно не указывать только при его инициализации, при обычном объявлении массива обязательно нужно указывать размер массива

```
#include "stdafx.h"
#include <iostream>
using namespace std;

int main(int argc, char* argv[])
{
   cout << "obrabotka massiva" << endl;
// объявление и инициализация одномерного массива
   int array1[16] = { 5, -12, -12, 9, 10, 0, -9, -12, -1, 23, 65, 64, 11, 43, 39, -
   15 };
   cout << "indeks" << "\t\t" << "element massiva" << endl; // печать
   заголовков
    for (int counter = 0; counter < 16; counter++) //начало цикла
   {
   //вывод на экран индекса ячейки массива, а затем содержимого этой
   ячейки cout << "array1[" << counter << "]" << "\t\t" << array1[counter]
   << endl;
```

```
    }
    system("pause");
    return 0;
}
```

Результат работы представлен на рисунке 2.

```
obrabotka massiva indeks               element massiva
array1[0]             5
array1f1]             -12
array1[2]             -12
array1[3]             9
array1[4]             10
array1[5]             0
array1[6]             -9
array1[7]             -12
array1[8]             -1
array1[9]             23
array1[10]            65
array1[11]            64
array1[12]            11
array1[13]            43
array1[14]            39
array1[15]            -15
Для продолжения нажмите любую клавишу . . .
```

Рисунок 2

2) программа должна последовательно считывать десять введённых чисел с клавиатуры. Все введённые числа просуммировать, результат вывести на экран. #include "stdafx.h"

```
#include <iostream>
using namespace std;
int main(int argc, char* argv[])
{
    int array1[10]; // объявляем целочисленный массив
    cout << "Enter elementi massiva: " << endl;
    int sum = 0;
    for ( int counter = 0; counter < 10; counter++ ) // цикл для считывания чисел
        cin >> array1[counter]; // считываем вводимые с клавиатуры числа
    cout << "array1 = {";
for ( int counter = 0; counter < 10; counter++ ) // цикл для вывода элементов массива cout << array1[counter] << " "; // выводим элементы массива на стандартное
устройство вывода
    for ( int counter = 0; counter < 10; counter++ ) // цикл для суммирования чисел
массива
        sum += array1[counter]; // суммируем элементы массива
    cout << "}\nsum = " << sum << endl;
    system("pause");
    return 0;
}
```

Результат представлен на рисунке 3.

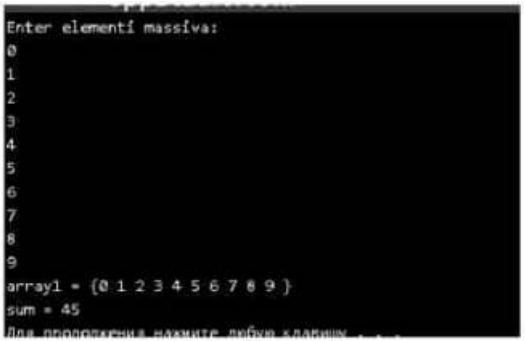


Рисунок 3.

Пример выполнения задания.

Задание: произвести следующую обработку 15 целых чисел: подсчитать сумму чисел, входящих в диапазон [-5..5] и количество нечетных чисел.

Схема программы к данному заданию с использованием одного цикла представлена на рисунке 4.

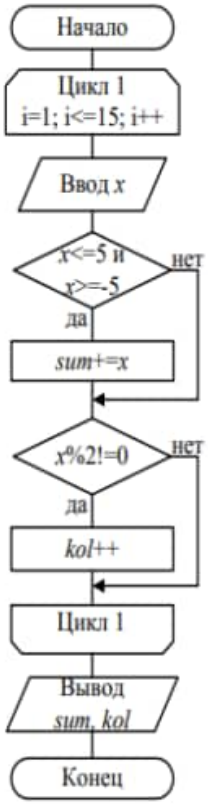


Рисунок 4

Текст программы без использования массива:

```
#include <iostream>

void main()
{
int x,sum=0,i,kol=0;
printf("Enter numbers\n");
for (i=1;i<=15;i++)
{
scanf("%d",&x);
if ((x>=-5)&&(x<=5)) sum+=x;
```

```
if (x%2!=0) kol++;
}
printf("Summa v diapazone [-5,5]=%d\n", sum);
printf("Kolichestvo nechetnih=%d", kol);
}
```

Схема программы с циклами для ввода и обработки массива (рисунок 5).

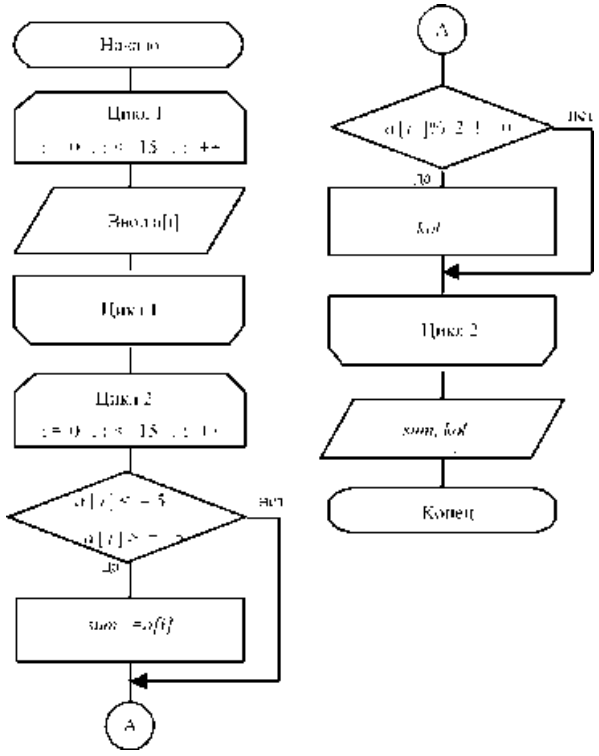


Рисунок 5

Пример программы с использованием одномерного массива

```
#include<iostream>
void main()
{
int a[15],s("E um=0,i,kol=0;
printf(“Enter numbers\n");
for (i=0;i<15;i++)
scanf("%d",&a[i]);
for (i=0;i<15;i++)
{
if ((a[i]>=-5)&&(a[i]<=5)) sum+=a[i];
if (a[i]%2!=0) kol++;
}
printf("Summa v diapazone [-5,5]=%d\n", sum);
printf("Kolichestvo nechetnih=%d", kol);
}
```

Контрольные вопросы

1. Массивы в языке С++: понятие массива в языке С++, описание массива в программе, представление элементов массива в памяти, обращение к элементам массива.

Варианты заданий

1. Произвести следующую обработку 15 целых чисел: найти количество отрицательных чисел, количество нулевых и подсчитать сумму положительных чисел.
2. Произвести следующую обработку 15 целых чисел: найти количество четных чисел, а нечетные числа, входящие в диапазон [1..11] возвести в квадрат.
3. Произвести следующую обработку 15 вещественных чисел: найти количество отрицательных чисел, а числа, входящие в диапазон [0..10] возвести в квадрат.
4. Произвести следующую обработку 10 вещественных чисел: найти количество чисел, больших или равных 1,5, и подсчитать сумму отрицательных чисел.
5. Произвести следующую обработку 10 вещественных чисел: найти количество чисел, равных нулю, и найти сумму чисел, входящих в диапазон [15..15].
6. Произвести следующую обработку 15 целых чисел: найти количество отрицательных чисел и подсчитать разность положительных чисел.
7. Произвести следующую обработку 15 вещественных чисел: найти среднее арифметическое положительных чисел и подсчитать количество чисел, входящих в диапазон [-15..5].
8. Произвести следующую обработку 10 целых чисел: найти количество отрицательных чисел и подсчитать сумму положительных чисел, делящихся без остатка на 3.
9. Произвести следующую обработку 10 целых чисел: найти количество отрицательных чисел, а числа, входящие в диапазон [0..10], умножить на 10.
10. Произвести следующую обработку 10 целых чисел: найти количество отрицательных чисел, а числа, входящие в диапазон [0..10], умножить на 3.
11. Произвести следующую обработку 15 вещественных чисел: найти среднее арифметическое отрицательных чисел и подсчитать количество чисел, входящих в диапазон [0..5].
12. Произвести следующую обработку 15 вещественных чисел: найти среднее арифметическое нечетных чисел и подсчитать сумму чисел, входящих в диапазон [15..5].
13. Произвести следующую обработку 10 вещественных чисел: найти количество чисел, равных нулю, и найти синус чисел, входящих в диапазон [-15..15].
14. Произвести следующую обработку 10 целых чисел: подсчитать сумму положительных чисел и определить номера отрицательных чисел.
15. Произвести следующую обработку 15 вещественных чисел: найти количество отрицательных чисел и номера нулевых чисел.
16. Произвести следующую обработку 12 целых чисел: подсчитать количество чисел, делящихся без остатка на 5, и сумму чисел, входящих в диапазон [-5..5].
17. Произвести следующую обработку 10 вещественных чисел: подсчитать количество чисел, отличающихся от числа 3 не более чем на 0.5, и сумму отрицательных чисел.
18. Произвести следующую обработку 15 целых чисел: подсчитать количество нулевых чисел и вычислить квадраты чисел, входящих в диапазон [-5..5].
19. Произвести следующую обработку 12 целых чисел: подсчитать количество нечетных чисел и сумму отрицательных чисел.
20. Произвести следующую обработку 15 вещественных чисел: подсчитать количество чисел, отличающихся от заданного не более чем на 0.5, и сумму положительных чисел.
21. . Произвести следующую обработку 10 вещественных чисел: найти количество отрицательных чисел, находящихся в диапазоне от -5 до 5 и подсчитать сумму положительных чисел.
22. Произвести следующую обработку 15 целых чисел: найти количество чисел,

входящих в диапазон [1..11] и каждое число возвести в квадрат.

23. Произвести следующую обработку 15 вещественных чисел: найти количество чисел, равных 0, а числа, входящие в диапазон [-1..1] возвести в куб.

24. Произвести следующую обработку 10 вещественных чисел: найти количество чисел, больших или равных 1,5, и подсчитать сумму отрицательных чисел, входящих в диапазон [-1..0].

25. Произвести следующую обработку 10 вещественных чисел: найти количество чисел, меньших 15, и найти произведение чисел, входящих в диапазон [10..15].

Практическое занятие № 10

Разработка программ с использованием двумерных массивов. Указатели

Цель работы: изучение способов описания, ввода-вывода и обработки двумерных массивов, использование указателей при работе с массивами.

Краткие теоретические сведения

Двумерный массив — это обычная таблица, со строками и столбцами. Фактически двумерный массив — это одномерный массив одномерных массивов. Структура двумерного массива, с именем *a*, размером *m* на *n* показана ниже (рисунок 1).

a[0][0]	a[0][1]	a[0][2]	a[0][3]	. . .	a[0][n]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	. . .	a[1][n]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	. . .	a[2][n]
. . .	. . .	. . .	. . .	. . .	. . .
a[m][0]	a[m][1]	a[m][2]	a[m][3]	. . .	a[m][n]

Рисунок 1 — Двумерный массивы в С++

где, *m* — количество строк двумерного массива; *n* — количество столбцов двумерного массива;

m x *n* — количество элементов массива.

Наиболее распространенный синтаксис объявления двумерного массива:

/*тип данных*/ /*имя массива*/[/*количество строк*/][/*количество столбцов*/];

Примеры инициализации двумерного массива:

— способ 1:

```
int arr[5][3];
```

— способ 2:

```
int arr[5][3] = { {4, 7, 8}, {9, 66, -1}, {5, -5, 0}, {3, -3, 30}, {1, 1, 1} };
```

В последнем случае представление массива и обращение к элементу массива имеет вид, показанный на рисунке 2,3.

Рисунок 2 — Двумерный массив в С++

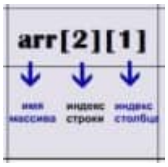


Рисунок 3 — Обращение в элементу массива

Ввод массива с клавиатуры и его вывод на экран выполняется следующим образом:

```
int m,n,ij;
 cout<<"Введите размеры матрицы:"<<endl;
 cin>>m;
 cin>>n;
 int elem;
 int ar [m][n];
  for (i=0; i<m; i++)
 {for (j=0; j<n; j++){
  cout<<«"Введите элемент:"«endl;
```

```
  cin>>elem;
   ar [i][j]=elem;
   }
}
for (i=0; i<m; i++)
{for (j=0; j<n; j++){
  cout<<ar[i][j]«endl;
  }
}
system("pause");
return 0;
}
```

При работе с массивами можно использовать указатели. Указатель - переменная, значением которой является адрес ячейки памяти. То есть указатель ссылается на блок данных из области памяти, причём на самое его начало. Указатель может ссылаться на переменную или функцию. Для этого нужно знать адрес переменной или функции. Так вот, чтобы узнать адрес конкретной переменной в C++ существует унарная операция взятия адреса &. Такая операция извлекает адрес объявленных переменных, для того, чтобы его присвоить указателю.

Указатели используются для передачи по ссылке данных, что намного ускоряет процесс обработки этих данных (в том случае, если объём данных большой), так как их не надо копировать, как при передаче по значению, то есть, используя имя переменной. В основном указатели используются для организации динамического распределения памяти, например при объявлении массива, не надо будет его ограничивать в размере. Ведь программист заранее не может знать, какого размера нужен массив тому или иному пользователю, в таком случае используется динамическое выделение памяти под массив. Любой указатель необходимо объявить перед использованием, как и любую переменную:

/*тип данных*/ * /*имя указателя*/;

Работу с указателями можно представить следующим образом:

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int main(int argc, char* argv[])
{
   int var = 123; // инициализация переменной var числом 123
   int *ptrvar = &var; // указатель на переменную var (присвоили адрес переменной
указателю)
   cout << "&var = " << &var << endl;// адрес переменной var содержащийся в памяти,
извлечённый операцией взятия адреса
   cout << "ptrvar = " << ptrvar << endl;// адрес переменной var, является значением
указателя ptrvar
   cout << "var = " << var << endl; // значение в переменной var
   cout << "*ptrvar = " << *ptrvar << endl; // вывод значения содержащегося в переменной var
через указатель, операцией разименования указателя
   system("pause");
   return 0;
}
```

Результат работы представлен ниже на рисунке 4.

Рисунок 4

Пример выполнения задания.

Задание. Найти максимальную сумму элементов строк матрицы 3x5. Написать программы без использования указателей и с использованием указателей.

Схема программы без использования указателей представлена на рисунке 5:

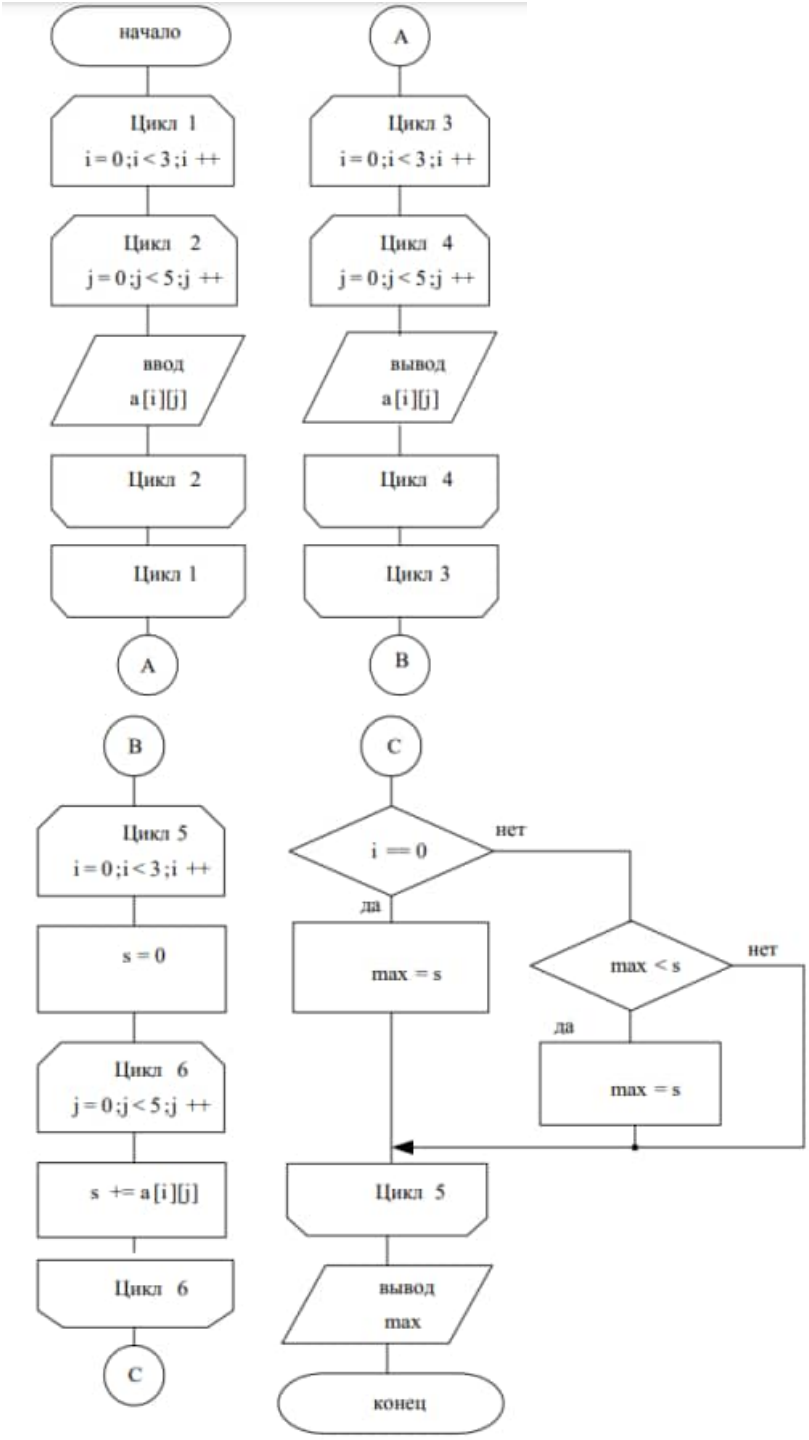
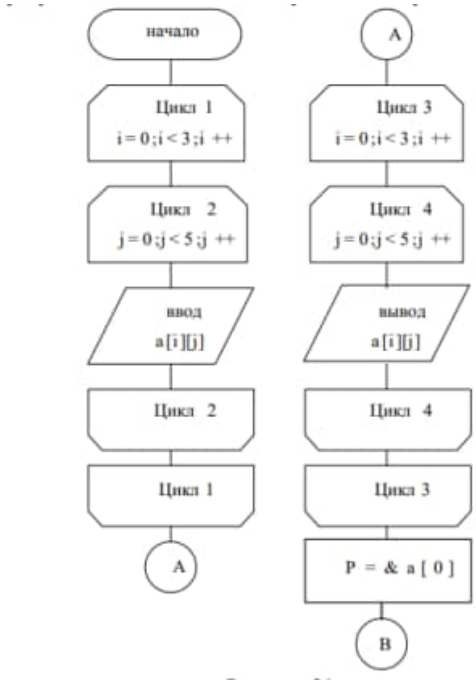


Рисунок 5

Текст программы:

```
#include <stdio.h>
void main()
{
int a[3][5], i, j, s, max;
printf (“Введите 3 строки по 5 чисел”);
for (i=0;i<3;i++)
  for (j=0;j<5;j++)
  scanf("%d",&a[i][j]);
   printf (“Матрица a :\n”);
      for (i=0; i<3; i++)
       {for (j=0; j<5; j++)
        printf (“%5d”, a[i][j]);
       printf (“\n”);
  }
 for(i=0;i<3;i++)
  {s=0;
      for (j=0;j<5;j++)
       s+=a[i][j];
    if (i==0) max=s;
    else if (max<s) max=s;}
printf ("Максимальная сумма строки = %d",max);
}
```

Схема программы с использованием указателей представлена на рисунке 6



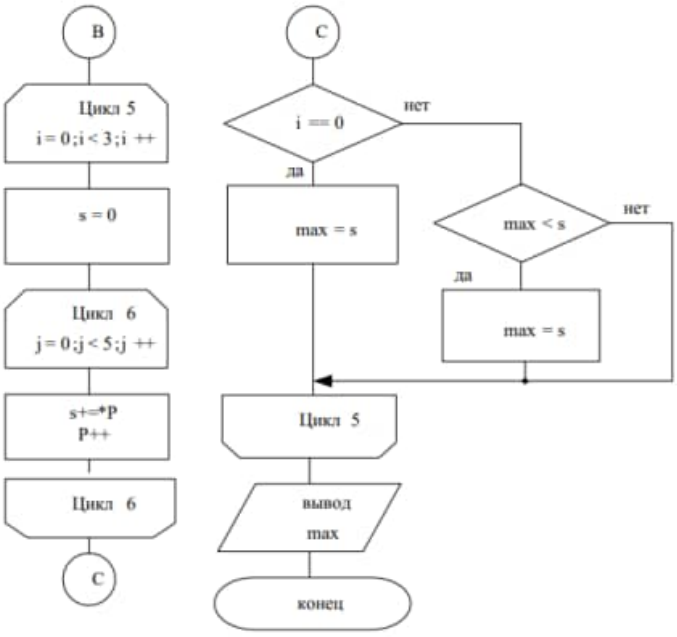


Рисунок 6

Пример программы с использованием указателей:

```
#include <stdio.h>
void main()
{
int a[3][5], *P, i, j, s, max;
printf (“Введите 3 строки по 5 чисел”);
for (i=0;i<3;i++)
 for (j=0;j<5;j++)
  scanf("%d",&a[i][j]);
printf (“Матрица a :\n”);
 for (i=0; i<3; i++)
  {for (j=0; j<5; j++)
    printf (“%5d”, a[i][j]);
    printf (“\n”);
  }
P=&a[0][0];
for(i=0;i<3;i++)
  {s=0;
    for (j=0;j<5;j++)
      {s+=*P;
       P++;
      }
    if (i==0) max=s;
    else if (max<s) max=s;
  }
printf ("Максимальная сумма строки = %d",max);
}
```

Контрольные вопросы

1. Особенности организации двумерных массивов: понятие массива в языке С++,

описание массива в программе, представление элементов массива в памяти, обращение к элементам массива.

2. Указатели в языке С++: понятие указателя, описание указателя в программе.
3. Операции над указателями.
4. Связь массивов и указателей.

Варианты заданий

1. Вычислить сумму положительных элементов каждого столбца матрицы A(txn).
2. Из матрицы *X* (mxn) построить матрицу *Y,* поменяв местами строки и столбцы.
3. Найти наименьший элемент матрицы *X* (mxn) и записать нули в ту строку и столбец, где он находится.
4. Переписать первые элементы каждой строки матрицы *A* (mxn), большие *C*, в массив *B***.** Если в строке нет элемента, большего *C*, то записать ноль в массив *B*.
5. Дана действительная матрица размера *m*x*n*. Найти сумму наибольших значений элементов ее строк.
6. В данной действительной матрице размера mxn поменять местами строку, содержащую элемент с наибольшим значением, со строкой, содержащей элемент с наименьшим значением. Предполагается, что такой элемент единственный.
7. В данной действительной квадратной матрице порядка *n* найти сумму элементов строки, в которой расположен элемент с наименьшим значением. Предполагается, что такой элемент единственный.
8. Дана действительная матрица размера *m*x*n*, все элементы которой различны. В каждой строке выбирается элемент с наименьшим значением, затем среди этих чисел выбирается наибольшее. Указать индексы элемента с найденным значением.
9. Дана целочисленная матрица размера mxn. Найти матрицу, получающуюся перестановкой столбцов (первого с последним, второго с предпоследним и т.д.).
10. Дана целочисленная матрица размера mxn. Найти матрицу, получающуюся перестановкой строк (первой с последней и т.д.).
11. Дана действительная матрица [*aij*], где i, j = 1..*n*. Получить действительную матрицу [*bij*], где i, j = 1..*n*, элемент *bij* которой равен сумме элементов данной матрицы, расположенных в области, определяемой индексами i, j (область заштрихована на рисунке 12):

Рисунок 12

12. Дана действительная квадратная матрица порядка *n*. Преобразовать матрицу по правилу: строку с номером *n* сделать столбцом с номером *n*, а столбец с номером *n* сделать строкой с номером *n*.
13. Просуммировать элементы матрицы X (4,5), сумма индексов которых равна заданной константе K.
14. Дана матрица M (4x5). Вычислить вектор D, компоненты которого равны сумме элементов строк матрицы.
15. Дана матрица M (6x6). Вычислить сумму элементов главной диагонали.
16. Дана матрица N (6x5). Найти столбец с минимальной суммой элементов.
17. Дана матрица M (4x5) и константа C. Вычислить матрицу D, равную произведению элементов матрицы M на константу.

18. Дана матрица M (4x6). Вычислить вектор D, компоненты которого равны сумме элементов столбцов матрицы.

19. Дана действительная квадратная матрица порядка *n*, все элементы которой различны. Найти наибольший элемент, среди стоящих на главной и побочной диагоналях и поменять его местами с элементом, стоящим на пересечении этих диагоналей.

20. Дана действительная квадратная матрица порядка n. Найти наибольшее из значений элементов, расположенных в заштрихованной части матрицы (рисунок 13):

Рисунок 13

21. Дана матрица M (2x5), определить максимальный и минимальный элементы. Поменять местами максимальный и минимальный элементы.

22. В матрице A(пхп) вычислить сумму элементов матрицы (n-2xn-2) и определить максимальный элемент в ней.

23. Дана матрица M (6x6). Вычислить произведение элементов главной диагонали с константой C.

24. Дана матрица N (6x5). Найти строку с минимальной суммой элементов, а элемент с номером nij возвести в квадрат.

25. Дана матрица вещественных чисел A (mxn). В сроке m определить максимальный элемент, а в столбце n количество элементом, меньших порога k.

Практическое занятие № 11

Разработка программ с использованием строк

Цель работы: изучение правил описания, ввода-вывода и основных функций обработки символьных данных.

Разработка программ с использованием строк **Краткие теоретические сведения**.

Строки в С++ позволяют работать с символьными данными. С помощью строк возможно осуществить чтение с клавиатуры текста, его обработка и вывод.

В С++ существует 2 типа строк:

1. Массив переменных типа char, заканчивающийся нуль-терминатором \0.

Символьные строки состоят из набора символьных констант заключённых в двойные кавычки. При объявлении строкового массива необходимо учитывать наличие в конце строки нуль-терминатора, и отводить дополнительный байт под него.

char string[10]; // string - имя строковой переменной, 10 - размер массива, (в строке может поместиться 9 символов, последнее место отводится под нуль-терминатор)

2. Специальный класс string

Для его работы необходимо в начале программы подключить заголовочный файл string: *#include <string>*

Для создания строки необходимо в начале программы написать

using namespace std;

Теперь чтоб создать строку достаточно написать: *string s;*

Для записи в строку можно использовать оператор =

s="Hello";

Доступ к i-му элементу строки **s** типа string осуществляется стандартным образом s[i]. Над строками типа string определены следующие операции:

1. присваивания, например s1=s2;
2. объединения строк (s1+=s2 или s1=s1+s2) — добавляет к строке si строку s2, результат храниться в строке si, пример объединения строк:
3. сравнения строк: si=s2, si!=s2, si<s2, si>s2, si<=s2, si>=s2 — результатом будет логическое значение.

Существует множество функций для работы со строками (таблица 1).

Таблица 1- Функции для работы со строками

Функция	Объяснение
s.append(str)	добавляет в конец строки строку str
s.assign(str)	присваивает строке s значение строки str
s.clear()	отчищает строку, т.е. удаляет все элементы в ней
s.compare(str)	сравнивает строку s со строкой str и возвращает 0 в случае совпадение
s.copy(C, I, N)	копирует из строки s в строку C(может быть как строка типа string, так и строка типа char) I символов, начиная с N-го символа
bool b=s.empty()	если строка пуста, возвращает true, иначе false
s.erase(I,N)	удаляет N элементов с I-го символа
s.find(str,I)	ищет строку str начиная с I- го символа
s.insert(pos, s1)	вставляет строку **si** в строку **s**, начиная с позиции **pos**
int len=s.length()	записывает в len длину строки
s.push back(symbol)	добавляет в конец строки символ

Функция	Объяснение
s.replace(index, n,str)	берет n первых символов из str и заменяет символы строки s на них, начиная с позиции index
str=s.substr(n,	возвращает m символов начиная с позиции n
s.swap(str)	меняет содержимое s и str местами.
s.size()	возвращает число элементов в строке.

Функции для работы со строками, прототипы которых описаны в заголовочном файле string.h:

1 Сравнение строк. Для сравнения строк используются функции strcmp и strncmp. Функция

int strcmp (const char *str1, const char *str2);

лексикографически сравнивает строки strl, str2 и возвращает -1, 0 или 1, если строка str1 соответственно меньше, равна или больше строки str2.

Функция

int strncmp (const char *str1, const char *str2, size_t n);

лексикографически сравнивает не более чем n первых символов из строк str1 и str2. Функция возвращает -1, 0 или 1, если первые n символов из строки str1 соответственно меньше, равны или больше первых n символов из строки str2.

Пример:

```
// пример сравнения строк
#include #include
int main() {
char str1[] = "aa bb";
char str2[] = "aa aa";
char str3[] = "aa bb cc";
printf("%d\n", strcmp(str1, str3)); // печатает: -1
printf("%d\n", strcmp(str1, str1)); // печатает: -0
printf("%d\n", strcmp(str1, str2)); // печатает: 1
printf("%d\n", strncmp(str1, str3, 5)); // печатает: 0 return 1;
}
```

2 Копирование строк. Для копирования строк используются функции strcpy и strncpy. Функция

char *strcpy (char *str1, const char *str2);

копирует строку str2 в строку str1. Строка str2 копируется полностью, включая завершающий нулевой байт. Функция возвращает указатель на str1. Если строки перекрываются, то результат непредсказуем.

Функция

char *strncpy (char *str1, const char *str2, size_t n);

копирует n символов из строки str2 в строку str1. Если строка str2 содержит меньше чем n символов, то последний нулевой байт копируется столько раз, сколько нужно для расширения строки str2 до n символов. Функция возвращает указатель на строку str1.

Пример:

```
char str1[80];
char str2 = "Copy string.";
strcpy ( str1, str2 );
printf ( str1 ); // печатает: Copy string.
```

3 Соединение строк. Для соединения строк в одну строку используются функции strcat и strncat. Функция

char* strcat (char *str1, const char *str2);

присоединяет строку str2 к строке str1, причем завершающий нулевой байт строки str1 стирается. Функция возвращает указатель на строку str1.

Функция

char* strncat (char *str1, const char *str2, size_t n);

присоединяет n символов из строки str2 к строке str1, причем завершающий нулевой байт строки str1 стирается. Функция возвращает указатель на строку str1. если длина строки str2 меньше n, то присоединяются только символы, входящие в строку str2. После соединения строк к строке str1 всегда добавляется нулевой байт. Функция возвращает указатель на строку str1.

Пример:

```
#include <stdio.h>
#include <string.h>
int main ( )
{
char str1[80] = "String ";
char str2 = "catenation ";
char str3 = "Yes No";
strcat ( str1, str2 );
printf ("%s\n", str1 );                    // печатает: String catenation
strncat ( str1, str3, 3 );
printf ("%s\n", str1 );                    // печатает: String catenation Yes
return 1;
}
```

4 Поиск символа в строке. Для поиска символа в строке используются функции strchr, strrchr, strspn, strcspn и strpbrk. Функция char* strchr (const char *str, int c);

ищет первое вхождение символа, заданного параметром c, в строку str. В случае успеха функция возвращает указатель на первый найденный символ, а в случае неудачи - NULL.

Функция

char* strrchr (const char *str, int c);

ищет последнее вхождение символа, заданного параметром c, в строку str. В случае успеха функция возвращает указатель на последний найденный символ, а в случае неудачи - NULL.

Пример:

```
#include <stdio.h>
#include <string.h>
int main ( )
{
char str[80] = "Char search";
printf ("%s\n", strchr ( str, 'r' ));                    // печатает: r search
printf ("%s\n", strrchr ( str, 'r' )); // печатает: rch
return 1;
}
```

Функция

size_t strspn (const char *str1, const char *str2);

возвращает индекс первого символа из строки str1, который не входит в строку str2.

Функция

size_t strcspn (const char *str1, const char *str2);

возвращает индекс первого символа из строки str1, который входит в строку str2.

Пример:

```
int main ( )
{
char str[80] = "123 abc";
printf ("n = %d\n", strspn ( str, "321" ); // печатает: n = 3
printf ("n = %d\n", strcspn ( str, "cba" ); // печатает: n = 4 return 1;
}
```

Функция

char* strpbrk (const char *str1, const char *str2);

находит первый символ в строке str1, который равен одному из символов в строке str2. В случае успеха функция возвращает указатель на этот символ, а в случае неудачи - NULL.

Пример.

```
char str[80] = "123 abc";
printf ("%s\n", strpbrk ( str, "bca" )); // печатает: abc
```

5. Сравнение строк. Для сравнения строк используются функция strstr. Функция

```
char* strstr ( const char *str1, const char *str2 );
```

находит первое вхождение строки str2 (без конечного нулевого байта) в строку str1. В случае успеха функция возвращает указатель на найденную подстроку, а в случае неудачи - NULL. Если указатель str1 указывает на строку нулевой длины, то функция возвращает указатель str1.

Пример:

```
char str[80] = "123 abc 456;
printf ("%s\n", strstr ( str, "abc"); // печать: abc 456
```

6. Разбор строки на лексемы. Для разбора строки на лексемы используется функция strtok. Функция

```
char* strtok ( char *str1, const char *str2 );
```

возвращает указатель на следующую лексему (слово) в строке str1, в которой разделителями лексем являются символы из строки str2. В случае если лексемы закончились, то функция возвращает NULL. При первом вызове функции strtok параметр str1 должен указывать на строку, которая разбирается на лексемы, а при последующих вызовах этот параметр должен быть установлен в NULL. После нахождения лексемы функция strtok записывает после этой лексемы на место разделителя нулевой байт.

Пример.

```
#include <stdio.h>
#include <string.h>
int main( )
{
char str[ ] = "12 34 ab cd";
char *p;
p = strtok ( str, " " );
while (p)
{
printf ( "%s\n", p ); // печатает в столбик значения: 12 34 ab cd
p = strtok ( NULL, " " );
}
return 1;
}
```

7. Определение длины строки. Для определения длины строки используется функция strlen. Функция

```
size_t strlen ( const char *str);
```

возвращает длину строки, не учитывая последний нулевой байт. Например, char str[] = "123";

```
printf ("len = %d\n", strlen ( str )); // печатает: len = 3
```

Пример выполнения задания.

Задание. Найти слова во введенной с клавиатуры строке, вывести их на экран и подсчитать их количество.

Схема программы представлена на рисунке 1:

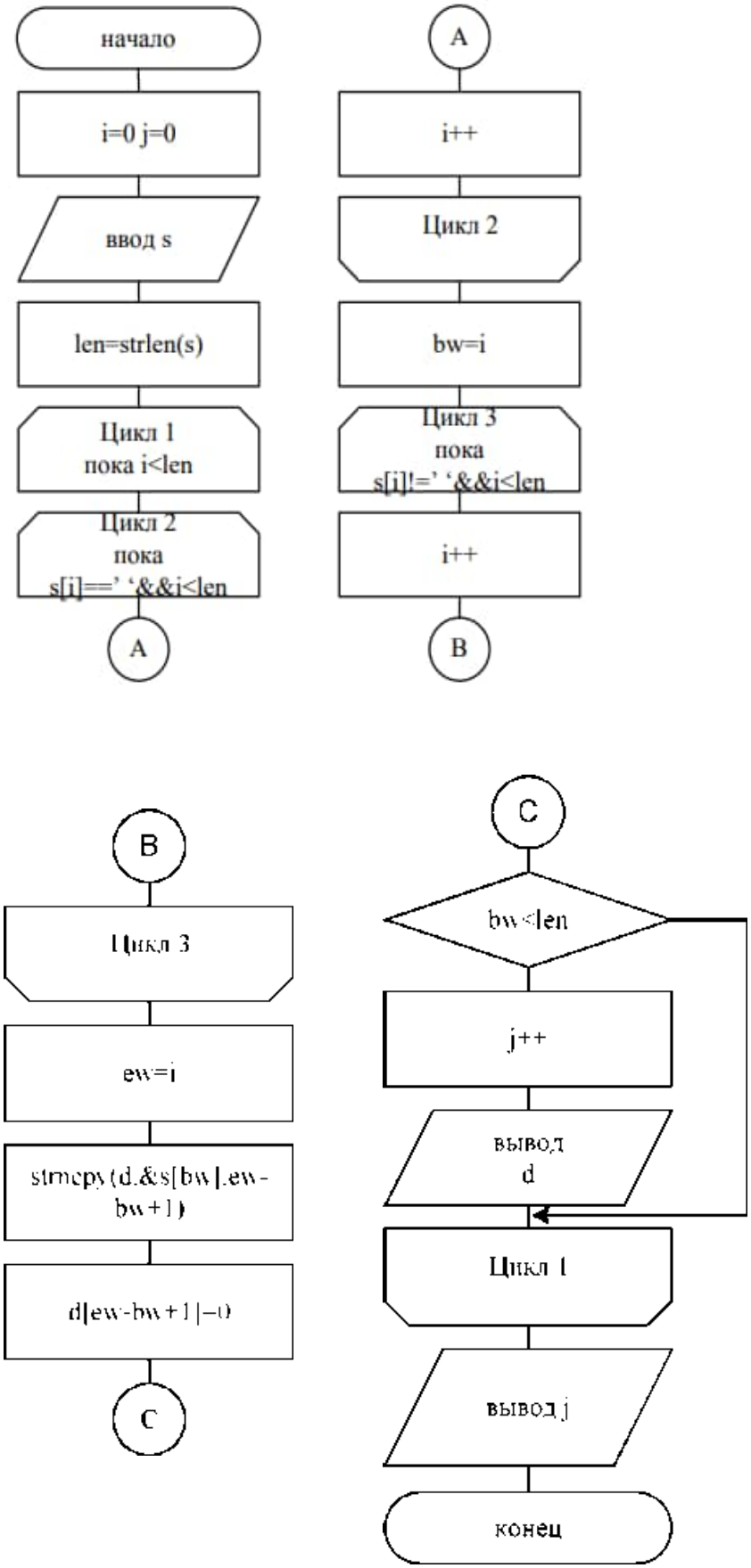

начало
i=0 j=0
ввод s
len=strlen(s)
Цикл 1
пока i<len
Цикл 2
пока
s[i]==' '&&i<len
A
A
i++
Цикл 2
bw=i
Цикл 3
пока
s[i]!=' '&&i<len
i++
B
B
Цикл 3
ew=i
strncpy(d,&s[bw],ew-bw+1)
d[ew-bw+1]=0
C
C
bw<len
j++
вывод
d
Цикл 1
вывод j
конец

Рисунок 1

Пример программы:

```
#include<stdio.h>
#include<string.h>
vid main()
{
 char s[100],d[100];
 int i=0,j=0,bw,ew,len;
 gets(s); len=strlen(s);
 while (i<len)
 {
   while((s[i]==' ')&&(i<len)) i++;
   bw=i;
   while((s[i]!=' ')&&(i<len)) i++;
   ew=i;
   strncpy(d,&s[bw],ew-bw+1);
   d[ew-bw+1]=0;
   if (bw<len)
      printf("%s\n",d);}
 }
printf("Vsego slov %d\n", j);
}
```

Контрольные вопросы

1. Строки в языке С++: понятие строки, описание строк в программе, обращение к элементам строки.
2. Три способа ввода строк в С++.
3. Три способа вывода строк в С++.
4. Способы инициализации строк (задание значений в программе).
5. Стандартные функции для обработки строк.

Варианты заданий

Дана последовательность символов S1, ..., SN. Группы символов, разделенные пробелом (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами.

1. Определить число символов в самом длинном слове строки. Слова отделяются знаком “/”.
2. В произвольном тексте выделить и отпечатать слова, начинающиеся с буквы А.
3. В произвольном тексте вставить между первым и вторым словом новое слово.
4. В произвольном тексте найти и отпечатать слова, содержащие букву Е.
5. Отпечатать второе и третье слова произвольного текста.
6. В произвольном тексте вставить между вторым и третьим словом новое слово.
7. В произвольном тексте найти и отпечатать все слова длиной 5 символов.
8. В произвольном тексте найти самое короткое слово.
9. В последовательности из 10 пятибуквенных слов найти и поменять местами пару слов, у которых первые три буквы одного совпадают с последними тремя буквами другого.
10. Упорядочить в алфавитном порядке последовательность из 10 пятибуквенных слов.
11. В строке из 50 символов отдельные слова разделены пробелом. Упорядочить

строку так, чтобы каждое следующее слово было не короче предыдущего.

12. Расположить слова строки в порядке, обратным исходному.

13. Подсчитать количество букв ‘а’ в последнем слове строки.

14. Найти количество слов, у которых первый и последний символы совпадают между собой.

15. Исключить из строки слова, расположенные между скобками (,). Сами скобки должны быть исключены.

16. В произвольном тексте найти и отпечатать слова, содержащие букву А.

17. Отпечатать первое и второе слова произвольного текста.

18. В произвольном тексте вставить после первого слова новое слово.

19. В произвольном тексте найти и отпечатать все слова длиной 4 символа.

20. В произвольном тексте найти самое длинное слово.

21. Выполнить сравнение двух строк s и d. Результат вывести в виде сообщения «идентичны» или «не идентичны».

22. Добавить в конец строки новое слово, длинною 5 символов, иначе выдать сообщение об ошибке.

23. Добавить в начало строки новое слово, начинающееся с буквы а, иначе, если слово начинается с другой буквы вывести сообщение о невозможности добавления.

24. Посчитать какое количество раз встречается буква n (задается при каждом выполнении алгоритма).

25. Проанализировать массив символов, состоящий из n символов. Если массив состоит из n-5 символов, добавить в конец набор символов rrrrr.

Практическое занятие № 12

Разработка программ с использованием функций

Цель работы: Освоение методов определения функций, передачи аргументов, использования библиотек функций, основ нисходящей технологии программирования.

Краткие теоретические сведения

Функции разбивают большие вычислительные задачи на маленькие подзадачи и позволяют использовать в работе то, что уже сделано другими, а не начинать каждый раз с пустого места. Соответствующие функции часто могут скрывать в себе детали проводимых в разных частях программы операций, знать которые нет необходимости, проясняя тем самым всю программу, как целое, и уменьшая трудности при внесении изменений.
Широко используются «библиотечные» функции. Каждая функция языка С имеет имя и список аргументов (формальных параметров).

Функции могут возвращать значение. Это значение может быть использовано далее в программе. Так как функция может вернуть какое-нибудь значение, то обязательно нужно указать тип данных возвращаемого значения. Если тип не указан, то по умолчанию предполагается, что функция возвращает целое значение (тип int). После имени функции принято ставить круглые скобки (это касается вызова функции, её объявления и описания). В этих скобках перечисляются параметры функции, если они есть. Если у функции нет параметров, то при объявлении и при описании функции вместо <список параметров> в приведенном далее примере кода надо поставить void - пусто.
Основная форма описания функции имеет вид:

```
        тип <имя функции>(список параметров)
{
      тело функции
}
```

Объявление (прототип) функции имеет вид:

```
        тип <имя функции>(список параметров);
```

Обратите внимание на то, что при описании функции после заголовка функции точка с запятой не ставится, а при объявлении функции точка с запятой ставится.
Вызов функции делается следующим образом:

```
        <имя функции>(параметры);
```

или

```
        <переменная>=<имя функции>(параметры);
```

При вызове функции так же ставится точка с запятой.

Почему надо объявлять функцию до использования? Дело в том, что для правильной работы кода функции машине надо знать тип возвращаемого значения, количество и типы аргументов. При вызове какой-либо функции копии значений фактических параметров записываются в стек, в соответствии с типами указанными в ее прототипе. Затем происходит переход в вызываемую функцию.
Приведем пример вызова функции

```
        /* Используем свою функцию */
void function1(void);      // Объявление функции
void main(void)       // Точка входа в программу
{
    function1();             //  Вызов функции
}
/* Описание функции  */
void function1(void)          // Заголовок функции
{       // Начало тела функции
```

```
}       //  Конец тела функции
```

Мы объявили функцию function1(), затем её вызвали. Необходимо обратить внимание ещё на то, что тип возвращаемого значения у функции void (пусто). Это значит, что функция не будет возвращать никакого значения.

Рассмотрим пример, в котором описаны две функции, для одной из которых указан тип возвращаемого значения - int.

```
        int x;// Объявляем переменную x (глобальная переменная)
        void main(void)
{
      void function1(void);      // Объявляем функцию
      int function2();           // function2() будет
         // возвращать значение типа int
               x = 10; // Присваиваем переменной x значение
               function1(); // Вызываем функцию function1()
               x = function2();// Вызываем функцию function2()
}

/* Описание функций */
void function1(void)
{

        }
int function2(void)
{
      int y; // Объявляем локальную переменную
      y = x + 10;
      return y; // Возвращаем значение y
}
```

Теперь давайте посмотрим текст программы. Объявляем глобальную переменную x. Так как x - глобальная переменная, то она будет видна всем функция программы. В теле main() объявляем две функции, одна из которых может возвращать значение типа int. Далее присваиваем переменной x значение 10, так как x это глобальная переменная, то эта переменная будет видна функции main() т.е. функция main() может использовать эту переменную.

Далее мы вызываем функцию function1(). В следующей строке x = function2(); переменная x принимает значение которое вернет функция function2(). Посмотрите на описание функции function2(). В теле этой функции объявляем переменную y, а дальше переменной y мы присваиваем значение переменной x + 10. Так как x - глобальная переменная (она видна для функции function2), все будет работать.

Далее идет строка return y; с помощью оператора return мы возвращаем значение переменной y. Запомните, что если функция возвращает значение, то в теле этой функции обязательно должен присутствовать оператор return (он может быть и не один). Ну так вот с помощью оператора return мы возвращаем значение локальной переменной y в вызывающую функцию main().

Функции языка С могут иметь параметры. Эти параметры передаются в функцию и там обрабатываются. Ещё раз рассмотрим основную форму описания функции:

```
        тип <имя функции>(список параметров)
{
      тело функции
}
```

В списке параметров для каждого параметра должен быть указан тип.
Пример правильного списка параметров:

function(int x, char a, float z)

Пример неправильного списка параметров:

function(int x, a, float z)

Рассмотрим все это на примере. Пусть будет функция, у которой присутствует один параметр x. Функция будет возвращать квадрат значения x.

```
int square(int x)
{
   x = x * x;
   return x;
}
```

Формальные параметры - это параметры которые объявляются в заголовке функции при описании.

Фактические параметры - это параметры которые подставляются при вызове функции.

```
void myfunc(int x); // Объявление функции
void main(void)
{
  int a;
  a=5;
  myfunc(a); // a- фактический параметр
}
// Описание функции
void myfunc(int x) // x - формальный параметр
{
  x = x + 10;
}
```

В языке С функция может возвращать несколько значений. Чтобы функция могла вернуть несколько значений необходимо пользоваться указателями. Этот механизм называется - передача параметров по ссылке.

В С++ при вызове функций можно опускать параметры. В таких случаях для опущенных параметров будут использоваться значения по умолчанию. Обеспечение значений по умолчанию для параметров упрощает возможность повторного использования функций (их использования несколькими программами).

Применение библиотечных функций сокращает объем программирования, который программист должен выполнить самостоятельно. Вместо этого программа просто вызывает функции библиотеки этапа выполнения. В зависимости от компилятора библиотека этапа выполнения может состоять из тысяч функций.

Ранее было сказано, что до того, как программа сможет вызвать функцию, компилятор С++ должен узнать определение или прототип функции. Поскольку функции библиотеки этапа выполнения не определены в основной программе, необходимо указать прототип для каждой библиотечной функции, которую намерен использовать программист. Для упрощения использования библиотечных функций компилятор С++ предоставляет заголовочные файлы, содержащие корректные прототипы. Таким образом, программам необходимо просто включить требуемый заголовочный файл с помощью оператора #include, а затем вызвать необходимую функцию.

3. Порядок выполнения работы

1. Ознакомиться с теоретическими сведениями.
2. Получить вариант задания у преподавателя.
3. Выполнить задание.

4. Продемонстрировать выполнение работы преподавателю.
5. Оформить отчет.
6. Защитить лабораторную работу.

4. Требования к оформлению отчета

Отчет по лабораторной работе должен содержать следующие разделы:
титульный лист;
цель работы:
задание на лабораторную работу;
техническое описание выполненного задания;
блок-схему;
ответы на контрольные вопросы;
выводы по проделанной работе.

5. Варианты заданий

Во всех вариантах необходимо использовать пользовательские функции. Предусмотреть режим диалога.

1. Треугольник задан координатами своих вершин. Составить программу для вычисления его площади.
2. Составить программу для нахождения наибольшего общего делителя четырех натуральных чисел.
3. Составить программу для нахождения наименьшего общего кратного трех натуральных чисел.
4. Написать программу для нахождения суммы большего и меньшего из трех чисел.
5. Вычислить площадь правильного шестиугольника со стороной a, используя функцию вычисления площади треугольника.
6. На плоскости заданы своими координатами n точек. Составить программу, определяющую, между какими из пар точек самое большое расстояние. Указание. Координаты точек занести в массив.
7. Составить программу, которая в массиве A[10] находит второе по величине число (вывести на печать число, которое меньше максимального элемента массива, но больше всех других элементов).
8. Написать программу для вычисления суммы факториалов всех нечетных чисел от 1 до 9.
9. Даны две дроби A/B и C/D (A, B, C, D — натуральные числа). Составить программу для деления дроби на дробь. Результат должен быть несократимой дробью.
10. Задан массив D. Определить следующие суммы: D[1] + D[2] + D[3]; D[3] +D[4] +D[5]; D[4] + D[5] + D[6]. Пояснение. Составить подпрограмму для вычисления суммы трех последовательно расположенных элементов массива с номерами k до m.
11. На плоскости заданы своими координатами n точек. Создать массив размером n(n-1), элементами которого являются расстояния от каждой из точек до n -1 других.
12. Даны числа X, Y, Z, T — длины сторон четырехугольника. Вычислить его площадь, если угол между сторонами длиной X и Y— прямой.
13. Составить программу для вычисления суммы факториалов всех четных чисел от m до n.
14. Заменить отрицательные элементы линейного массива их модулями, не пользуясь стандартной функцией вычисления модуля. Подсчитать количество произведенных замен.
15. Дано простое число. Составить функцию, которая будет находить следующее за ним простое число.
16. Составить функцию для нахождения наименьшего нечетного натурального делителя k (k не равно 1) любого заданного натурального числа n.

17. Дано натуральное число N. Составить программу для формирования массива, элементами которого являются цифры числа N.
18. Составить программу, определяющую, в каком из данных двух чисел больше цифр.
19. Заменить данное натуральное число на число, которое получается из исходного записью его цифр в обратnov порядке.
20. Даны три квадратных матрицы A, B, C размерность 10. Вывести на печать ту из них, норма которой наименьшая. Пояснение. Нормой матрицы назовем максимум из абсолютных величин ее элементов.
21. Два натуральных числа называются «дружественными», если каждое из них равно сумме всех делителей (кроме его самого) другого числа (например, числа 220 и 284). Найти все пары «дружественных чисел», которые не больше данного числа N.
22. Два простых числа называются «близнецами», если они отличаются друг от друга на 2 (например, 41 и 43). Напечатать все пары «близнецов» из отрезка [n, 2n], где n — заданное натуральное число больше 2.

Контрольные вопросы

1. Зачем нужно использовать функции ?
2. Чем отличается прототип и определение функции?
3. Для чего нужны прототипы функций?
4. Что такое формальные и фактические параметры? Пример.

Практическое занятие № 13

Разработка программ с использованием рекурсивных функций

Цель работы: Научиться решать задачи, используя рекурсивные функции.

Краткие теоретические сведения

Суть рекурсивных методов—сведение задачи к самой себе. Вы уже знаете, что в С существует возможность рекурсивного определения функций. Эта возможность представляет собой способ программной реализации рекурсивных алгоритмов. Однако увидеть рекурсивный путь решения задач и (рекурсивный алгоритм) часто очень непросто.

Классическая задача - «Ханойская башня». На площадке (назовем ее А) находится пирамида, составленная из дисков уменьшающегося от основания к вершине размера. Эту пирамиду в том же виде требуется переместить на площадку В. При выполнении этой работы необходимо соблюдать следующие ограничения:

•перекладывать можно только по одному диску, взятому сверху пирамиды;

•класть диск можно либо только на основание площадки, либо на диск большего размера;

•в качестве вспомогательной можно использовать площадку С.

Название «Ханойская башня» связано с легендой, согласно которой в давние времена монахи одного ханойского храма взялись переместить по этим правилам башню, состоящую из 64 дисков. С завершением их работы наступит конец света. Монах и все еще работают и, надеемся, еще долго будут работать!

Нетрудно решить эту задачу для двух дисков. Обозначая перемещения диска, например, с площадки А на В так: А—>В, напишем алгоритм для этого случая

A-)C;A->B;C->B.

Функция Сложения двух натуральных чисел ($Sum(a,\ b) = a + b$) может быть рассмотрена в качестве рекурсивной функции двух переменных

$Sum(x,\ y+1) = F(x,\ y,\ Sum(x,\ y))$;

Умножение двух натуральных чисел ($Mul(a,\ b) = a \times b$) может быть рассмотрено в качестве рекурсивной функции двух переменных

$Mul(x,\ y+1) = G(x,\ y,\ Mul(x,y))$;

Рекурсивной называется функция, которая вызывает саму себя. Такая рекурсия называется прямой. При косвенной рекурсии, две или более функций вызывают друг друга. При рекурсивном вызове функции в стеке создается копия значений ее параметров, как и при вызове обычной функции, после чего управление передается первому исполняемому оператору функции. При повторном вызове этот процесс повторяется. Ясно, что для завершения вычислений каждая рекурсивная функция должна содержать хотя бы одну нерекурсивную ветвь алгоритма, заканчивающуюся оператором возврата.

При завершении функции соответствующая часть стека освобождается, и управление передается вызывающей функции, выполнение которой продолжается с точки, следующей за рекурсивным вызовом.

Приведем пример рекурсивной функции, вычисляющей факториал.

```
long fact(long n){
if (n==0 || n==1) return 1;
return (n * fact(n - 1);
}
```

Содержание работы.

1. Ознакомиться с теоретической частью методических указаний.
2. Получить вариант задания у преподавателя.
3. Решить поставленную задачу с использованием языка программирования С++.
4. Представить работающую программу преподавателю.

Варианты заданий

Решить следующие задачи, используя рекурсивную функции.

1. Найти сумму цифр заданного натурального числа.
2. Подсчитать количество цифр в заданном натуральном числе.
3. Составить программу для вычисления наибольшего общего делителя двух натуральных чисел.
4. Составить программу для нахождения числа, которое образуется из данного натурального числа при записи его цифр в обратном порядке. Например, для числа1234 получаем результат 4321.
5. Составить программу для перевода данного натурального числа в 7 -ричную систему счисления.
6. Дана символьная строка, представляющая собой запись натурального числа в 7-ичнойсистемесчисления(2<p<9).Составить программу для перевода этого числа в десятичную систему счисления.
7. Составить программу для вычисления суммы: 2! + 4! + 6! +
+ ... + «! (п< 16, п — четное).

Примечание. Тип результата значения функции — Longlnt.

8. Дано п различных натуральных чисел. Напечатать все перестановки этих чисел.
9. Логическая функция возвращает True, если ее аргумент — простое число.
10. Описать функцию, которая удаляет из строки все лишние пробелы. Пробелы считаются лишними, если их подряд идет более двух, если они стоят в конце строки после последней точки, если стоят после открывающегося парного знака препинания.

Содержание отчета.

1. Цель работы и задание.
2. Пункты, соответствующие порядку выполнения работы.
3. Выводы по работе.

Практическое занятие № 14
Разработка программ работы с файлами

Цель работы: научиться выполнять ввод-вывод из файла с помощью стандартной библиотеки

Краткие теоретические сведения

Частью стандартной библиотеки C++ является библиотека iostream – объектно-ориентированная иерархия классов, где используется и множественное, и виртуальное наследование. В ней реализована поддержка для файлового ввода/вывода данных встроенных типов. Кроме того, разработчики классов могут расширять эту библиотеку для чтения и записи новых типов данных.

Для использования библиотеки iostream в программе необходимо включить заголовочный файл

#include <iostream>

Операции ввода/вывода выполняются с помощью классов istream (потоковый ввод) и ostream (потоковый вывод). Третий класс, iostream, является производным от них и поддерживает двунаправленный ввод/вывод. Для удобства в библиотеке определены три стандартных объекта-потока:

- cin – объект класса istream, соответствующий стандартному вводу. В общем случае он позволяет читать данные с терминала пользователя;
- cout – объект класса ostream, соответствующий стандартному выводу. В общем случае он позволяет выводить данные на терминал пользователя;
- cerr – объект класса ostream, соответствующий стандартному выводу для ошибок. В этот поток мы направляем сообщения об ошибках программы.

Помимо чтения с терминала и записи на него, библиотека iostream поддерживает чтение и запись в файлы. Для этого предназначены следующие классы:

- ifstream, производный от istream, связывает ввод программы с файлом;
- ofstream, производный от ostream, связывает вывод программы с файлом;
- fstream, производный от iostream, связывает как ввод, так и вывод программы с файлом.

Чтобы использовать часть библиотеки iostream, связанную с файловым вводом/выводом, необходимо включить в программу заголовочный файл

#include <fstream>

Если файл будет использоваться только для вывода, мы определяем объект класса ofstream. Например:

```
ofstream outfile( "copy.out", ios::base::out );
```

Передаваемые конструктору аргументы задают имя открываемого файла и режим открытия. Файл типа ofstream может быть открыт либо – по умолчанию – в режиме вывода (ios_base::out), либо в режиме дозаписи (ios_base::app). Такое определение файла outfile2 эквивалентно приведенному выше:

```
// по умолчанию открывается в режиме вывода
ofstream outfile2( "copy.out" );
```

Если в режиме вывода открывается существующий файл, то все хранившиеся в нем данные пропадают. Если же мы хотим не заменить, а добавить данные, то следует открывать файл в режиме дозаписи: тогда новые данные помещаются в конец. Если указанный файл не существует, то он создается в любом режиме.

Прежде чем пытаться прочитать из файла или записать в него, нужно проверить, что файл был успешно открыт:

```
if ( ! outfile ) { // открыть файл не удалось
  cerr <<"не могу открыть "copy.out" для записи\n";
  exit( -1 );
}
```

Класс ofstream является производным от ostream. Все определенные в ostream операции применимы и к ofstream.

Следующая программа читает из стандартного ввода символы и копирует их в стандартный вывод:

```
#include <fstream>
int main()
{
   // открыть файл copy.out для вывода
   ofstream outFile( "copy.out" );
   if ( ! outFile ) {
      cerr << "Не могу открыть 'copy.out' для вывода\n";
      return -1;
   }
   char ch;
   while ( cin.get( ch ) )
  outFile.put( ch );
}
```

Чтобы открыть файл только для чтения, применяется объект класса ifstream, производного от istream. Следующая программа читает указанный пользователем файл и копирует его содержимое на стандартный вывод:

```
#include <fstream>
#include <string>
int main()
{
   cout << "filename: ";
   string file_name;
   cin>> file_name;
   // открыть файл для ввода
   ifstream inFile( file_name.c_str() );
   if ( !inFile ) {
       cerr < <"не могу открыть входной файл: "
           <<file_name << " -- аварийный останов!\n";
       return -1;
   }
   char ch;
   while ( inFile.get( ch ))
         cout.put( ch );
}
```

Объекты классов ofstream и ifstream разрешено определять и без указания имени файла. Позже к этому объекту можно присоединить файл с помощью функции-члена open():

```
ifstream curFile;
// ...
curFile.open( filename.c_str() );
if ( ! curFile )  // открытие успешно?
  // ...
```

Чтобы закрыть файл, вызываем функцию-член close():

```
.
.
.
.
```

```
    curFile.close();
  }
}
```

Объект класса fstream может также открывать файл одновременно для ввода и вывода. Например, приведенная инструкция открывает файл word.out для ввода и дозаписи:

```
fstream io( "word.out", ios_base::in|ios_base::app );
```

Для задания нескольких режимов используется оператор побитового ИЛИ.

Порядок выполнения работы

1. Ознакомиться с теоретическими сведениями.
2. Получить вариант задания у преподавателя.
3. Выполнить задание.
4. Продемонстрировать выполнение работы преподавателю.
5. Оформить отчет.
6. Защитить лабораторную работу.

Требования к оформлению отчета

Отчет по лабораторной работе должен содержать следующие разделы:
титульный лист;
цель работы:
задание на лабораторную работу;
техническое описание выполненного задания;
блок-схему;
выводы по проделанной работе.

Варианты заданий

1. Заполнить файл f целыми числами, полученными с помощью генератора случайных чисел. Получить в файле g те компоненты файла f, которые являются четными.
2. Записать в файл 10 действительных чисел. Вычислить произведение компонентов файла и вывести на печать.
3. Заполнить файл f целыми числами, полученными с помощью генератора случайных чисел. Получить в файле g все компоненты файла f, которые делятся на 2.
4. Записать в файл 10 целых чисел, полученных с помощью генератора случайных чисел. Подсчитать количество пар противоположных чисел среди компонентов этого файла.
5. Заполнить файл f целыми числами, полученными с помощью генератора случайных чисел. Из файла f получить файл g, исключив повторные вхождения чисел.
6. Записать в файл 10 произвольных натуральных чисел. Переписать в другой файл те элементы, которые кратны К.
7. Заполнить файл 10 действительными числами, полученными с помощью датчика случайных чисел. Найти сумму минимального и максимального элементов этого файла.
8. Записать в файл 15 натуральных чисел: a_1, a_2,..., a_n (числа получить с помощью датчика случайных чисел). Сформировать новый файл, элементами которого являются числа a_1, a_1*a_2, $a_1*a_2*a_3$, ….
9. Записать в файл f 10 натуральных чисел. Получить в другом файле все компоненты файла f, кроме тех, которые кратны К.
10. Заполнить файл f целыми числами, полученными с помощью генератора случайных чисел. Найти количество удвоенных нечетных чисел среди компонентов файла.

11. Заполнить файл f натуральными числами, полученными с помощью генератора случайных чисел. Найти количество квадратов нечетных чисел среди компонентов.
12. Записать в файл 10 действительных чисел. Найти наибольшее из значений модулей компонентов с нечетными номерами.
13. Заполнить файл f целыми числами, полученными с помощью генератора случайных чисел. Из файла f получить файл g, исключив повторные вхождения чисел. Порядок следования чисел сохранить.
14. Записать в файл 10 действительных чисел. Найти разность первого и последнего компонентов файла.
15. Записать в файл f 10 целых чисел, полученных с помощью генератора случайных чисел. Заполнить файл g числами, которые являются произведениями соседних компонентов файла f.
16. Багаж пассажира характеризуется количеством вещей и их общим весом. Дан файл Bagazh, содержащий сведения о багаже нескольких пассажиров. Сведения о багаже каждого пассажира представляют собой запись с двумя полями: одно поле целого типа (количество вещей) и одно — действительного (вес в килограммах). Найти багаж, средний вес одной вещи в котором отличается не более чем на t кг от общего среднего веса одной вещи.
17. В условиях предыдущей задачи найти число пассажиров, имеющих более двух вещей, и число пассажиров, количество вещей которых превосходит среднее число вещей.
18. В условиях задачи 17 выяснить, имеется ли пассажир, багаж rоторого состоит из одной вещи весом менее t кг.
19. Дан файл Bibl, содержащий сведения о книгах. Сведения о каждой из книг — это фамилия автора, название и год издания. Найти названия книг данного автора, изданных начиная с 2000 г.
20. Дан файл Assort, содержащий сведения об игрушках: указываются название игрушки, ее стоимость в рублях и возрастные границы (например, игрушка может предназначаться для детей от двух до пяти лет). Получить название игрушек, цена которых не превышает 1400 руб. и которые подходят детям 5 лет.
21. Дан файл Assort, содержащий сведения об игрушках: указываются название игрушки, ее стоимость в рублях и возрастные границы (например, игрушка может предназначаться для детей от двух до пяти лет).
22. Дан файл Assort, содержащий сведения об игрушках: указываются название игрушки, ее стоимость в рублях и возрастные границы (например, игрушка может предназначаться для детей от двух до пяти лет). Определить стоимость самого дорогого конструктора.
23. Дан файл Assort, содержащий сведения об игрушках: указываются название игрушки, ее стоимость в рублях и возрастные границы (например, игрушка может предназначаться для детей от двух до пяти лет). Напечатать название наиболее дорогих игрушек (цена которых отличается от цены самой дорогой игрушки не более чем на 500 руб.);
24. Дан файл Assort, содержащий сведения об игрушках: указываются название игрушки, ее стоимость в рублях и возрастные границы (например, игрушка может предназначаться для детей от двух до пяти лет). Получить названия игрушек, которые подходят детям как четырех, так и десяти лет.
25. Дан файл Assort, содержащий сведения об игрушках: указываются название игрушки, ее стоимость в рублях и возрастные границы (например, игрушка может предназначаться для детей от двух до пяти лет). Получить сведения о

том, можно ли подобрать игрушку, любую, кроме мяча, подходящую ребенку трех лет.

26. Дан файл Assort, содержащий сведения об игрушках: указываются название игрушки, ее стоимость в рублях и возрастные границы (например, игрушка может предназначаться для детей от двух до пяти лет). Получить название самой дешевой игрушки.
27. Дан файл Assort, содержащий сведения об игрушках: указываются название игрушки, ее стоимость в рублях и возрастные границы (например, игрушка может предназначаться для детей от двух до пяти лет). Получить название самой дорогой игрушки для детей до четырех лет.
28. Дан файл Assort, содержащий сведения об игрушках: указываются название игрушки, ее стоимость в рублях и возрастные границы (например, игрушка может предназначаться для детей от двух до пяти лет). Получить названия игрушек для детей четырех-пяти лет.
29. Дан файл Assort, содержащий сведения об игрушках: указываются название игрушки, ее стоимость в рублях и возрастные границы (например, игрушка может предназначаться для детей от двух до пяти лет). Получить название самой дорогой игрушки, подходящей детям двух-трех лет.
30. Дан файл Assort, содержащий сведения об игрушках: указываются название игрушки, ее стоимость в рублях и возрастные границы (например, игрушка может предназначаться для детей от двух до пяти лет). Определить стоимость самой дорогой куклы.
31. Дан файл Assort, содержащий сведения об игрушках: указываются название игрушки, ее стоимость в рублях и возрастные границы (например, игрушка может предназначаться для детей от двух до пяти лет). Определить суммарную стоимость кукол для детей шести лет.

Практическое занятие № 15
Основы работы с интегрированной средой
Краткие теоретические сведения

Microsoft Visual Studio .NET 20xx. В Visual Studio .NET (далее VS .NET) каждый проект является частью того, что Microsoft называет решением (solution). Любой код, созданный в VS .NET IDE, относится к некоторому решению. Решение можно рассматривать как хранилище всей информации, необходимой для компиляции программы и ее перевода в форму, пригодную для исполнения. Таким образом, решение состоит из одного или нескольких проектов; различных вспомогательных файлов (графических изображений, ресурсных файлов, метаданных, то есть данных, описывающих другие данные, и т. д.); документации в формате XML. Решение позволяет легко выбрать файлы, задействованные в решении конкретной проблемы.

Новое решение создается командой File > New > Project. Далее требуется выбрать тип проекта, который будет первым в решении, имя решения, и каталог где оно будет находиться. Все новые проекты добавляются в решение лишь с одним отличием, при создании проекта в поле «решение» требуется выбрать «В текущее» (Add to Solution) вместо «в новое решение» (Create new solution). При помощи команды View в главном меню всегда можно вызвать нужное окно на передний план (и передать ему фокус). Все окна IDE свободно перетаскиваются мышью.

Основные окна IDE.

Редактор текста. Редактор обладает полным набором стандартных возможностей, поддерживаемых в редакторах такого рода (вырезание, вставка, поиск/замена и т. д.). Для работы с ними можно использовать стандартные комбинации клавиш Windows (Ctrl+X — вырезать, Ctrl+V — вставить и т. д.). Если вы предпочитаете работать с командами меню, к вашим услугам меню Edit и контекстное меню окна программы. Полный список сочетаний клавиш вызывается из меню Edit; кроме того, он приведен в разделе «Editing, shortcut keys» справочной системы. Например, комбинация Ctrl+I включает режим поиска с приращением.

В распоряжении разработчика имеется средство IntelliSense, выдающее информацию о методах заданного объекта или параметрах, передаваемых при вызове функции. Обычно IntelliSense вызывается автоматически, но его можно вызвать нажатием «Ctrl + Пробел».

Настройка большинства глобальных параметров редактора выполняется в диалоговом окне — выполните команду Tools > Options и выберите в списке строку Text Editor. Например, чтобы выбрать размер позиций табуляции, щелкните в строке Text Editor и выберите нужное значение для всех языков или только для C++. Здесь же выбирается режим создания отступов: None (отступы отсутствуют), Block (курсор выравнивается по началу предыдущей строки) или Smart (автоматическое создание отступов в теле цикла, как того требует хороший стиль программирования). Кстати говоря, устанавливать размер позиций табуляции и форматировать отступы можно в готовом тексте, для чего используются комбинации клавиш Ctrl+K, Ctrl+F (сочетания клавиш требуется нажать подряд, без длительной паузы) или команда Edit > Advanced > Format Selection.

Также изменять отступы выделенного блока можно используя Tab для увеличения отступа и Shift + Tab для уменьшения.

Редактор поддерживает и такую возможность, как свертка фрагментов программы и отображение на их месте заголовков (folding). Обратите внимание на значки «-» рядом с некоторыми строками. Если щелкнуть на таком значке, в листинге будет скрыта соответствующая область (region), а после первой строки кода из блока появится многоточие. Если задержать указатель мыши над многоточием, на экране будет показан свернутый код. Для управления сверткой используется подменю Edit > Outlining. Информация о редакторе IDE находится в разделе справки «Code and Text Editor».

Список задач. В Visual Studio поддерживается список задач (task list). Идея состоит в том, что в программу включаются комментарии с описанием действий, которые предполагается выполнить в будущем; тип задачи определяется специальным ключевым словом, следующим после знака комментария. В настоящее время определены три встроенные категории задач — TODO, HACK и UNDONE. Комментарии с задачами выводятся в окне, вызываемом командой View > Other Windows > Task List (или комбинацией клавиш Ctrl+/, Ctrl + T).

Окно решения. В окне решения (Solution Explorer) выводится список файлов, входящих в решение. По умолчанию имя решения совпадает с именем первого созданного в нем проекта. Используя Solution Explorer, можно добавлять в проект различные файлы. Например, текст или исходный код. Для этого требуется щелкнуть правой кнопкой мыши по папке, куда необходимо добавить файл, в контекстном меню выбрать Add > New Item (Добавить > Новый элемент), чтобы добавить новый файл, или Add > Existent Item (Добавить > Существующий элемент), чтобы добавить существующий файл.

Окно свойств. Функции окна свойств в VS .NET уже не ограничиваются простым заданием свойств элементов управления. Содержимое окна зависит от того, что в настоящий момент выделено в IDE. Имя и тип выделенного элемента указаны в списке, находящемся в верхней части окна. Чтобы изменить значение свойства, щелкните в правой ячейке и начинайте вводить символы. В окне свойств действуют стандартные комбинации клавиш, используемые при редактировании в системе Windows.

Окно вывода и окно ошибок. В окне вывода (вызываемом командой View > Output или комбинацией клавиш Alt + 2) отображается текущая информация состояния. При построении решения в этом окне компилятор выводит сообщения как об успешном завершении, так и о возникших ошибках. В окне ошибок выводятся все ошибки или предупреждения возникшие во время компиляции программы. Оно вызывается сочетанием клавиш Ctrl + /, Ctrl + E.

Порядок выполнения работы

1. Ознакомиться с теоретическими сведениями.
2. Получить вариант задания у преподавателя.
3. Выполнить задание.
4. Продемонстрировать выполнение работы преподавателю.
5. Оформить отчет.
6. Защитить лабораторную работу.

Требования к оформлению отчета

Отчет по лабораторной работе должен содержать следующие разделы:
титульный лист;
цель работы:
задание на лабораторную работу;
техническое описание выполненного задания;
выводы по проделанной работе.

Задание на работу

1. Создать решение, содержащее консольное приложение.
2. Добавить проекту файл с исходным кодом следующего содержания:

```
#include “stdafx.h”
int sum( int a, int b){
   return a + b;
}
```

3. В функцию main добавить следующий код:

```
printf(“Sum: %d”,sum(3, 5));
```

4. Запустить созданный проект (клавиша F5). Объяснить, что выведено в окно вывода.
5. Добавить заголовочный файл с прототипом функции sum:

int sum(int a, int b)

6. Добавить в файл с описанием функции main подключение вновь созданного заголовочного файла #include “имя_файла.h”.
7. Заново запустить программу. Объяснить, что произошло.
8. Добавить в одном из файлов с исходным кодом новые задачи. Добавьте новые задачи через список задач. В чем разница?

Контрольные вопросы

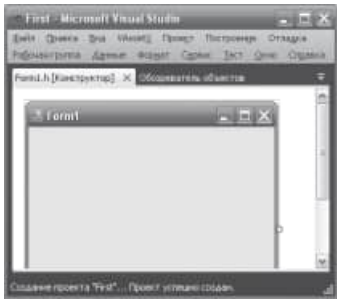
1. Что такое решение (solution) в Visual Studio .NET? Зачем они нужны?
2. Как создать решение? Как добавить туда проект?
3. Что такое IntelliSense? Как он вызывается?
4. Что такое список задач? Как можно добавить задачу?
5. Как осуществляется «сворачивание кода»? Какие области сворачиваются?
6. Зачем нужно окно свойств?
7. Зачем нужно окно вывода и окно ошибок? Чем они отличаются?
8. Зачем нужно окно решения?
9. Как настраиваются параметры форматирования текста в окне редактора?
10. Какие существуют команды форматирования? Как можно менять отступы блоков текста?
11. Как добавить в проект новый или существующий файл.

Практическое занятие № 16

Создание интерфейса пользователя в среде разработки программ

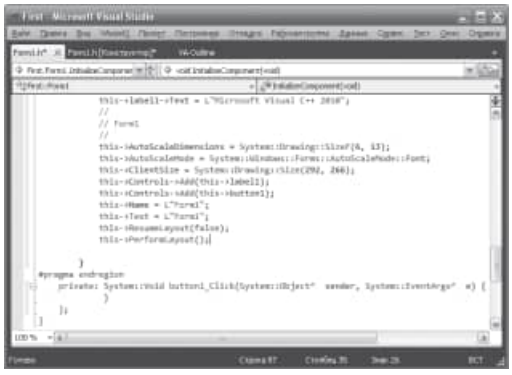
Цель работы: научиться создавать простейшие программы с экранной формой и элементами управления.

Краткие теоретические сведения



Для создания проекта выбираем File->New Project-> Windows Forms Application Visual C++ вводим имя проекта. Экранная форма — Form1, в которой можно расположить элементы управления. Например поля для ввода текста textBox, командные кнопки button, строчки текста label, которые не могут быть отредактированы пользователем. Визуальное программирование предполагает возможность перетаскивания элементов с помощью мыши из панели элементов Toolbox, где расположены всевозможные элементы управления, в форму. Панель Toolbox предоставляет доступ ко множеству элементов управления при создании Web- и Windows-форм. Она дает также доступ почти ко всему, что можно перетащить на визуальный конструктор, используемый для создания форм, XML, схем, классов и т. д.

Добавьте на форму метку label и кнопку button в форму, дважды щелкая на этих элементах на панели Toolbox или перетаскивая мышкой. У каждого объекта есть свойства. Свойств много, их можно увидеть, если щелкнуть правой кнопкой мыши в пределах формы и выбрать в контекстном меню команду Properties, при этом появится панель свойств Properties. Для объекта label1 выберем свойство Text и напишем напротив этого поля «Лабораторная работа № 9» (вместо текста label1). Для объекта button1 также в свойстве Text напишем «Ввод». Объекты не только имеют свойства, но и обрабатываются событиями. Событием, например, является клик на кнопке, загрузка (Load) формы в оперативную память и пр. Управляют событиями с помощью процедуры обработки события. Для получения пустого обработчика события щелчок на командной кнопке в свойствах кнопки button1 кликаем на значке молнии Events (события) и в списке всех возможных событий кнопки button1 выбираем двойным кликом событие Click. После этого мы попадаем на вкладку программного кода Form1.h.



На вкладке Form1.h мы увидим, что управляющая среда Visual C++ 2010 сгенерировала строки программного кода. Например, для свойства Text кнопки button управляющая среда назначила

this->button1->Text = L"Ввод";

Теперь напишем процедуру обработки события Click на кнопке button1. У нас есть обработчик события button1_Click:

private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e) { }. В фигурных скобках обработчика события напишем: *MessageBox::Show("Это моя программа!");*

Здесь вызывается метод Show объекта MessageBox с текстом «Это моя программа!» Оператор разрешения области действия (::) указывает системе найти метод Show среди методов объекта Messagebox. Теперь нажмем F5 и проверим работоспособность программы.

При работе с формой часто ввод данных организуют через элемент управления textBox . Добавляем в форму текстовое поле textBox. Изменим некоторые свойства элементов управления. Получим пустой обработчик загрузки формы (дважды кликаем по экранной форме). Задаем свойствам формы (к форме обращаемся посредством ссылки this), кнопке button1 и текстовому полю textBox1, метке label1 следующие значения:

```
this->Text = "Извлечение квадратного корня";
button1->Text = "Извлечь корень";
textBox1->Clear(); // Очистка текстового поля
label1->String::Empty;
```

Далее программируем событие button1.

```
private: System:: Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
Single X;
bool s1 = Single::TryParse(textBox1->Text,
System::Globalization::NumberStyles::Number,
System::Globalization::NumberFormatInfo::CurrentInfo, X);
if (s1 == false)
{
label1->Text = "Введите чило";
label1->ForeColor = Color::Red; // - цвет текста
return; }
Single Y = (Single)Math::Sqrt(X);
label1->ForeColor = Color::Black;
label1->Text = String::Format("Корень из {0} равен {1:F5}", X, Y);
}
```

При обработке события click проверяется, введено ли число в текстовом поле. Проверка осуществляется с помощью функции TryParse. Первым параметром метода TryParse является анализируемое поле textBox1->Text. Второй параметр — это разрешаемый для преобразования стиль числа. Третий параметр указывает, на какой основе формируется допустимый формат, CurrentInfo – это на основе текущего языка и региональных параметров. Четвертый параметр возвращает результат преобразования. Функция TryParse возвращает булеву переменную true или false, успешно ли выполнено преобразование. Если пользователь ввел число, то будет выполняться оператор извлечения квадратного корня Math::Sqrt(X). Математические функции Visual Studio 2010 являются методами класса Math. Функция Math::Sqrt(X)возвращает значение типа double (двойной точности с плавающей запятой), которое мы приводим с помощью неявного преобразования (Single) к переменной одинарной точности. Затем формируем строку label1->Text с использованием метода String::Format. Использованный формат ≪Корень из {0} равен {1:F5}≫ означает: взять нулевой выводимый элемент, то есть переменную X, и записать эту переменную вместо фигурных скобок; после чего взять первый

выводимый элемент, то есть Y, и записать его вместо вторых фигурных скобок в формате с фиксированной точкой и пятью десятичными знаками после запятой.

Рассмотрим пример работы с CheckBox (Флажком).

```
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
{
this->Text = "Флажок CheckBox";
checkBox1->Text = "Полужирный"; checkBox1->Focus();
label1->Text = "Выбери стиль шрифта";
label1->TextAlign = ContentAlignment::MiddleCenter;
label1->Font = gcnew System::Drawing::Font("Courier New",14.0F);
}
private: System::Void checkBox1_CheckedChanged(System::Object^ sender,
System::EventArgs^ e)
{ // Изменение состояния флажка на противоположное
if (checkBox1->Checked == true) label1->Font = gcnew
System::Drawing::Font("Courier New", 14.0F,FontStyle::Bold);
if (checkBox1->Checked == false) label1->Font = gcnew
System::Drawing::Font("Courier New", 14.0F,FontStyle::Regular);
}
```

При обработке события загрузки формы задаем начальные значения некоторых свойств объектов Form1 (посредством ссылки this), label1 и checkBox1. Изменение состояния флажка соответствует событию CheckedChanged. если флажок установлен (то есть содержит «галочку») Checked = true, то для метки label1 устанавливается тот же шрифт Courier New, 14 пунктов, но Bold, то есть полужирный. Если флажок не установлен, то есть checkBox1.Checked = false, то шрифт устанавливается Regular, то есть обычный.

Элемент управления ComboBox служит для отображения вариантов выбора в раскрывающемся списке. Продемонстрируем работу этого элемента управления Из панели Toolbox перетащим в форму два текстовых поля TextBox, кнопку Button, метку Label и комбинированный список ComboBox.

```
private: System:: Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
{
this->Text = "Калькулятор"; label1->Text = "Равно: ";
button1->Text = "Выбор операции";
comboBox1->Text = "Выбор операции";
array<String^>^ Операции = {"Сложение", "Вычитание",
"Умножение", "Деление", "Очистить"};
comboBox1->Items->AddRange(Операции);
comboBox1->TabIndex = 2;
textBox1->Clear(); textBox1->TabIndex = 0;
textBox2->Clear(); textBox2->TabIndex = 1;
}
private: System::Void comboBox1_SelectedIndexChanged(System::Object^ sender,
System::EventArgs^ e)
{
label1->Text = "Равно: ";
Single x, y, z= 0;
```

```
bool s1 = Single::TryParse(textBox1->Text,
System::Globalization::NumberStyles::Number,
System::Globalization::NumberFormatInfo::CurrentInfo, x);
bool s2 = Single::TryParse(textBox2-
>Text,System::Globalization::NumberStyles::Number,
System::Globalization::NumberFormatInfo::CurrentInfo, y);
if (s1 == false || s2 == false)
{
MessageBox::Show("Следует вводить числа!", "Ошибка",
MessageBoxButtons::OK, MessageBoxIcon::Error);
return;
}
switch (comboBox1->SelectedIndex)
{
case 0: z = x + y; break;
case 1: z = x - y; break;
case 2: z = x * y; break;
case 3: z = x / y; break;
case 4: textBox1->Clear(); textBox2->Clear(); label1->Text = "Равно: ";
 return;
}
label1->Text = String::Format("Равно {0:F5}", Z);
}
private: System::Void button1_Click(System::Object^ sender,System::EventArgs^ e)
{
comboBox1->DroppedDown = true;
}
};
}
```

При обработке события загрузки формы присваиваем начальные значения некоторым свойствам, задаем коллекцию элементов комбинированного списка, задаем табличные индексы TabIndex для текстовых полей и комбинированного списка. Табличный индекс определяет порядок обхода элементов. При обработке события comboBox1_SelectedIndexChanged с помощью функции TryParse проверяем, можно ли текстовые поля преобразовать в число. По умолчанию при инсталляции русифицированной версии Windows разделителем целой и дробной частей числа является запятая. Оператор switch осуществляет множественный выбор арифметической
операции в зависимости от индекса выбранного элемента списка SelectedIndex. Оператор switch передает управление той или иной метке case. Последний оператор в процедуре обработки события изменения индекса выбранного элемента осуществляет формирование строки с помощью метода String::Format для вывода ее на метку label1. Формат {0:F5} означает, что значение переменной z следует выводить по фиксированному формату с пятью знаками после запятой. Последняя процедура обработки события обеспечивает раскрытие комбинированного списка через нажатие на кнопку.

Рассмотрим пример, который позволяет пользователю вводить в текстовое поле
цифры, а также разделитель целой и дробной части числа.

```
String^  razd;
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
{
this->Text = "Введите число";
razd = Globalization::NumberFormatInfo::CurrentInfo->NumberDecimalSeparator;
}
private: System::Void textBox1_KeyPress(System::Object^ sender,
System::Windows::Forms::KeyPressEventArgs^ e)
{
bool razd_find = false;
if (Char::IsDigit(e->KeyChar) == true) return;
if (e->KeyChar == (char)Keys::Back) return;
if (textBox1->Text->IndexOf(razd) != -1)
razd_find = true;
if (razd_find == true) { e->Handled = true; return; }
if (e->KeyChar.ToString() == razd) return;
e->Handled = true;
}
};
```

Выясняем, что установлено в данной системе в качестве разделителя целой дробной части — точка или запятая. Этот разделитель записываем в строковую переменную razd, которая видна из всех процедур данной программы, поскольку объявлена вне всех процедур. Далее в процедуре обработки события KeyPress разрешаем ввод десятичных цифр и нажатие клавиши Backspace путем обхода с помощью return последнего оператора процедуры e->Handled = true, запрещающего ввод символа в текстовое поле. В данной задаче разрешить ввод разделителя мы можем только один раз, но при этом надо помнить, что пользователь может его удалить и ввести в другом месте числовой строки. Каждый раз при очередном нажатии клавиши, разрешив ввод десятичных цифр, в текстовом поле ищем искомый разделитель. Если он найден, то запрещаем ввод любых нецифровых символов, включая злосчастный разделитель. А если не найден, то разрешаем его ввод.

Содержание работы.

1. Ознакомиться с теоретической частью методических указаний.
2. Получить вариант задания у преподавателя.
3. Решить поставленную задачу с использованием языка программирования C++.
4. Представить работающую программу преподавателю.

Варианты заданий

Разработать программу с использованием экранной формы и элементов управления по варианту.

1. Вычислить периметр и площадь прямоугольного треугольника по длинам *a* и *b* двух катетов.

2. Заданы координаты трех вершин треугольника. Найти его периметр и площадь.
3. Вычислить длину окружности и площадь круга одного и того же заданного радиуса *R*.
4. Даны два числа. Найти среднее арифметическое кубов этих чисел и среднее геометрическое модулей этих чисел.
5. Вычислить расстояние между двумя точками с данными координатами.
6. Даны два действительных числа *хну.* Вычислить их сумму, разность, произведение и частное.
7. Дана длина ребра куба. Найти площадь грани, площадь полной поверхности и объем этого куба.
8. Дана сторона равностороннего треугольника. Найти площадь этого треугольника, его высоту, радиусы вписанной и описанной окружностей.
9. Известна длина окружности. Найти площадь круга, ограниченного этой окружностью.
10. Найти площадь кольца, внутренний радиус которого равен r, а внешний — *R (R> r).*
11. Треугольник задан величинами своих углов и радиусом описанной окружности. Найти стороны треугольника.
12. Найти площадь равнобедренной трапеции с основаниями *a* и *b* и углом а при большем основании *a.*
13. Вычислить корни квадратного уравнения *ax2+ Ьx + c* = 0 с заданными коэффициентами *a, b* и *c* (предполагается, что *a±0* и что дискриминант уравнения неотрицателен).
14. Найти площадь треугольника, две стороны которого равны *a* и b, а угол между этими сторонами *у.*
15. Написать программу, которая выводит на экран первые четыре степени числа л.
16. Найти сумму членов арифметической прогрессии, если известны ее первый член, знаменатель и число членов прогрессии.
17. Найти (в радианах в градусах) все углы треугольника со сторонами *a, b, c.*
18. Составить программу перевода радианной меры угла в градусы, минуты и секунды.
19. Составить программу для вычисления пути, пройденного лодкой, если ее скорость в стоячей воде *v* км/ч, скорость течения реки *v1* км/ч, время движения по озеру t1 ч, а против течения реки — *t2* ч.
20. Текущее показание электронных часов: *m* ч (0 < *m* < 23) *n* мин (0 < *n* < 59) *к* с (0 < *к* < 59). Какое время будут показывать часы через *р ч q* мин *r* с?
21. Вычислить высоты треугольника со сторонами *a, b, c.*
22. Полторы кошки за полтора часа съедают полторы мышки. Сколько мышек съедят А кошек за К часов?
23. Составить программу вычисления объема цилиндра и конуса, которые имеют одинаковую высоту и одинаковый радиус основания.
24. Дана величина *A,* выражающая объем информации в байтах. Перевести *A* в более крупные единицы измерения информации.
25. Окружность вписана в квадрат заданной площади. Найти площадь квадрата, вписанного в эту окружность. Во сколько раз площадь вписанного квадрата меньше площади заданного?

Содержание отчета

1. Цель работы и задание.
2. Пункты, соответствующие порядку выполнения работы.
3. Выводы по работе.

Практическое занятие № 17
Организация классов и принцип инкапсуляции

Цель работы: ознакомление с основными концепциями объектно-ориентированного программирования; изучение классов языка C++, способов их описания и использования, получение представления о перегрузке операторов и функций; получение навыков применения объектов в прикладных программах.

Краткие теоретические сведения

Объектно-ориентированный подход впитал в себя лучшие идеи структурированного и комбинирует их с новыми мощными концепциями, позволяющими увидеть задачу программирования в новом свете. Объектно-ориентированное программирование позволяет легко разложить сложную задачу на подзадачи, взаимодействующие друг с другом. Затем можно преобразовать эти подзадачи в единицы, называемые объектами. Все объектно-ориентированные языки имеют три общие концепции: инкапсуляцию, полиморфизм и наследование.

Инкапсуляция представляет собой механизм, который связывает вместе код и данные и который хранит их от внешнего воздействия и от неправильного использования. Более того, именно инкапсуляция позволяет создавать объекты. Попросту говоря, объект представляет собой логическое целое, включающее в себя данные и код для работы с этими данными. Мы можем определить часть кода и данных как собственность объекта, которая недоступна извне. На этом пути объект обеспечивает существенную защиту против случайной модификации или некорректного использования таких своих частных (*private*) членов. Во всех случаях объект представляет собой переменную, тип которой определяется пользователем. На первый взгляд может показаться странным представлять себе объект, который соединяет в себе вместе код и данные, как переменную. Тем не менее, в объектном программировании дело обстоит именно так. Когда создается объект, неявным образом создается новый тип переменной.

Объектно-ориентированные языки программирования поддерживают полиморфизм, который можно охарактеризовать следующей фразой: «один интерфейс – множество методов». Т.е. полиморфизм представляет собой механизм, который позволяет использовать один и тот же интерфейс при реализации целого набора различных действий. Выбор того, какое именно действие будет совершено, определяется конкретной ситуацией. Например, рассмотрим пример программы, которая определяет три различных типа списков. Один из них используется для целых чисел, другой – для символов, третий – для значений с плавающей запятой. Благодаря полиморфизму, можно создать три набора функций, имеющих одинаковое имя *push()* (поместить) и *pop()* (извлечь) – по одной на каждый тип данных. Общая концепция (интерфейс) заключается в том, чтобы вставлять и извлекать данные в список и из списка. Функции определяют специфические способы (методы), с помощью которых эти операции выполняются для каждого типа данных. Когда информация вставляется в список, автоматически вызывается та версия функции *push(),* которая соответствует типу обрабатываемых данных. Задача выбора специфического действия, т.е. метода, в зависимости от конкретной ситуации возлагается на компилятор.

Наследование представляет собой процесс, благодаря которому один объект может наследовать, приобретать свойства другого объекта. Это свойство поддерживает концепцию классификации, чем и обуславливается его важность. Например, красное яблоко представляет собой часть класса яблоко, который, в свою очередь, представляет собой часть класса фрукт, входящий в больший класс продукты питания. Без использования классификации каждый объект должен был бы определять все свои характеристики явным образом. На основе классификации объект нуждается только в определении таких качеств, которые отличают его от других объектов этого класса. Благодаря механизму наследования объект может характеризоваться в рамках классификации общего и частного.

Механизм объектов реализуется в языке программирования C++ с помощью классов. Общий вид описания класса следующий:

```
class <имя класса> {
    private:
        <частные данные и функции>;
protected:
        <защищенные данные и функции>;
    public:
        <публичные данные и функции>;
} <список объектов>;
```

Класс может содержать как частные (*private*) и защищенные (*protected*), так и публичные (*public*) данные. По умолчанию все члены класса являются частными, т.е. доступ к таким данным и функциям имеют только члены данного класса. Публичные элементы доступны для обращения из других мест программы. Защищенные члены аналогичны частным, отличия касаются только механизмов их наследования, которые будут рассматриваться в следующей лабораторной работе.

Как только определен класс, можно создать объект этого типа, используя имя класса. Фактически имя класса становится спецификатором нового типа данных. Рассмотрим, например, класс, описывающий окружность:

```
class circle {
        int x; // координаты центра
        int y;
        int r; // значение радиуса
    public:
        float get_length(void);
        float get_square(void);
        void set_params(int c_x, int c_y, int c_r);
        friend void func(void);
}
circle one, *two;
// создаем объект окружность one и указатель на объект two
```

Сравните: *int a, *b;*

Внутри объявления класса используются прототипы функций. Для написания реального кода функции-члена компилятору необходимо указать, к какому именно классу относится данная функция:

```
float circle::get_length(void)
{
   return 2*3.1415*r
}

float circle:: get_square(void)
{
   return 3.1415*r*r;
}

void circle:: set_params(int c_x, int c_y, int c_r)
  {  x = c_x, y = c_y, r = c_r;
  }
```

Последовательность символов :: называется оператором области видимости. Он показывает принадлежность функции конкретному классу. В C++ несколько различных классов могут использовать одинаковые имена функций. Компилятор понимает, какая из них принадлежит какому классу благодаря оператору области видимости и имени класса.

Функция, не являющаяся членом класса, может иметь доступ к его частным членам в случае, если она объявлена дружественной (*friend*) этому классу (см. пример выше). Одна из причин, почему C++ допускает существование таких функций, связана с той ситуацией, когда два класса для определенной цели должны использовать одну и ту же функцию.

Перед использовании объекта может потребоваться инициализировать некоторые его данные. Поскольку требования инициализации являются весьма распространенными, то C++ позволяет производить инициализацию объектов во время их создания. Такая

автоматическая инициализация выполняется с помощью функции, называемой конструктором класса. Функция конструктор, являющаяся членом класса и имеющая имя, совпадающее с именем класса, представляет собой специальный тип функции. Конструктор объекта вызывается автоматически при создании объекта. При инициализации глобальных или статических объектов конструктор вызывается только один раз, для локальных объектов конструктор вызывается каждый раз, когда встречается объявление объекта. Следует иметь в виду, что в C++ конструкторы не могут возвращать значений. Теперь в классе circle функция set_params() может быть заменена конструктором:

```
class circle {
        int x; // координаты центра
        int y;
        int r; // значение радиуса
    public:
        circle(int c_x, int c_y, int c_r);
        ~circle();
        float get_length(void);
        float get_square(void);
        friend void func(void);
}
circle::circle(int c_x, int c_y, int c_r)
  {
        x = c_x; y = c_y; r = c_r;
  }
```

Для передачи аргументов конструктору необходимо задать его значение при объявлении объекта. C++ поддерживает два способа решения этой задачи. В первом вызывается непосредственно с передачей ему значений переменных:

```
circle a = circle(1, 2, 3);
```

Во втором способе аргументы следуют непосредственно за объектом в круглых скобках:

```
circle a(1, 2, 3);
```

Дополнением конструктора является деструктор. Во многих случаях перед уничтожением объекта необходимо выполнить определенные действия. Локальные объекты создаются при входе в блок кода и уничтожаются при выходе из него. Глобальные объекты уничтожаются при завершении работы программы. Имеется много причин, чтобы существовал деструктор. Например, может потребоваться освободить память, которая была ранее зарезервирована. В C++ за дезактивацию отвечает деструктор. Он должен иметь то же имя, что и конструктор только к нему добавляется значок ~: *~circle().*

Если рассматривать часть программы, которая не входит в состав класса, то для доступа к его членам необходимо использовать следующую конструкцию:

<имя объекта>.<имя члена класса>, если мы работаем непосредственно с объектом или

<имя указателя>-><имя члена класса>, если мы работаем с указателем на объект. Например:

```
circle one, *two;
float a, b;
a = one.get_length();
b = two->get_square();
```

Напомним, что созданные объекты в C++ являются обычными переменными, поэтому они могут передаваться в функции в качестве аргументов, а также служить

типом возвращаемого функцией значения. Помимо этого, возможно создавать массивы объектов и получать указатели на объекты:

```
circle function(circle a, circle* b)
void main(void) {
circle one, *two, three[10], *four, *five, six;
two = &one; four = three; five = &three[2];
six = function(one, five);
}
```

Для реализации полиморфизма в C++ существует механизм, называемый перегрузкой функций. Смысл его заключается в том, что две и более функции могут иметь одинаковое имя, если они отличаются набором параметров в интерфейсе. Компилятор определяет, какую функцию можно использовать в конкретной ситуации, благодаря типу аргумента. По существу перегрузка функций позволяет создавать единое имя для операции, а компилятор устанавливает, какую именно функцию следует использовать в конкретной ситуации для выполнения данной операции. Например, опишем функции для ввода символов, целых и вещественных чисел, которые могут выдавать подсказку пользователю:

```
#include<iostream.h>
void promt(char *str, char *ch)
    {
        cout<<str;
        cin>>*ch;
    }
void promt(char *str, int *i)
    {
        cout<<str;
        cin>>*i;
    }
void promt(char *str; float *f);
    {
        cout<<str;
        cin>>*f;
    }
void main(void)
    {
        char a; int b; float c;
        promt(“Enter a char: ”, &a);
        promt(“Enter an integer: ”, &b);
        promt(“Enter a float: ”, &c);
        cout<<a<<” “<<b<<” “<<c;
    }
```

Другим способом реализации полиморфизма в языке C++ служит перегрузка операторов. Например, можно перегрузить операторы +, = и т.д. для выполнения специфичных действий. В общем случае можно перегружать операторы C++, определяя, что они означают применительно к определенному классу. Можно перегрузить оператор + по отношению к объектам типа <множество> таким образом, что он производит объединение двух произвольных множеств. Другой класс может использовать + совершенно другим образом. Например, для добавления элементов в список. Для того, чтобы перегрузить оператор, необходимо определить, что именно означает оператор по отношению к тому классу, к которому он применяется. Для этого определяется функция-

оператор, задающая действие оператора. Общая форма записи функции оператора для случая, когда она является членом класса, имеет вид:

```
<тип> <имя класса>::operator#(<список аргументов>)
    {
    <действия, выполняемые оператором>;
    }
```

Здесь перегружаемый оператор подставляется вместо символа #, а тип задает тип значений возвращаемых оператором. Для того, чтобы упростить использование перегруженного оператора в сложных выражениях, в качестве возвращаемого значения часто выбирают тот же самый тип, что и класс, для которого перегружен оператор. Например, перегрузим операторы + и = для класса *circle*.

```
class circle {
            int x; // координаты центра
            int y;
            int r; // значение радиуса
    public:
            circle(int c_x, int c_y, int c_r);
            ~circle();
            circle operator+(circle);
            circle operator=(circle);
}
circle::circle(int c_x, int c_y, int c_r)
  {
            x = c_x; y = c_y; r = c_r;
  }
circle circle::operator+(circle a)
// смысл оператора определяет сами – среднее //арифметическое двух окружностей
    {
            circle temp;
            temp.x = (x+a.x)/2;
            temp.y = (y + a.y)/2;
            temp.r = (r + a.r)/2;
            return temp;
    }
circle circle::operator=(circle a)
    {
            x = a.x; y = a.y; r = a.r;
            return *this;
    }
void main(void)
    {
            circle one(1, 1, 1), two(2, 2, 2), three(3, 3, 3);
            one = two + three;
    }
```

Необходимо отметить два момента. Во-первых, обе функции-оператора имеют по одному параметру, хотя и перегружают бинарные операторы + и =. Во-вторых, мы использовали ключевое слово языка C++ *this*. Всякий раз, когда вызывается функция-член класса, в нее неявно передается указатель на объект, вызывающий данную функцию. Чтобы получить доступ к этому указателю, и используется *this*.

Теперь вернемся к рассматриваемому примеру. Во всех случаях именно объект, стоящий слева от знака операции вызывает функцию-оператор. К нему можно обратиться через *this*. Объект, стоящий справа от знака операции, передается функции как аргумент. Поэтому строка *x = a.x* эквивалентна *this->x = a.x*, а *this* в операторе присваивания также используется для возвращения значения.

При перегрузке унарных операций функции-операторы не будут иметь параметров, а при перегрузке бинарных операторов, функции-операторы имеют по одному аргументу.

Порядок выполнения работы

1. Получить задание у преподавателя.
2. Разработать алгоритм решения задачи и написать программу, реализующую задание.
3. Проверить правильность ее работы.
4. Составить отчет и защитить работу.

Варианты заданий

1. Разработать класс, позволяющий выполнять действия над матрицами произвольных размерностей (сложение, вычитание, умножение, транспонирование, нахождение определителя). Интерфейс класса должен включать перегруженные операторы. Предусмотреть возможность загрузки данных из файла.

2. Разработать класс, позволяющий выполнять действия над векторами произвольной длины (сложение, вычитание, скалярное и векторное умножение, нахождение модуля и направляющих косинусов). Интерфейс класса должен включать перегруженные операторы. Предусмотреть возможность загрузки данных из файла.

3. Разработать класс, позволяющий оперировать с комплексными числами (сложение, вычитание, умножение, деление, возведение в степень, извлечение корня, перевод из одной формы записи в другую). Интерфейс класса должен включать перегруженные операторы. Предусмотреть возможность загрузки данных из файла.

4. Разработать класс, позволяющий оперировать с числами произвольной длины (сложение, вычитание, умножение). Интерфейс класса должен включать перегруженные операторы. Длинными называются числа, для операций с которыми нельзя использовать стандартные типы данных. Например, 1234567890987654321. Предусмотреть возможность загрузки данных из файла.

5. Разработать класс, позволяющий оперировать с множествами целых чисел (находить объединение, пересечение, инверсию и мощность множеств). Интерфейс класса должен включать перегруженные операторы. Предусмотреть возможность загрузки данных из файла.

6. Разработать класс, позволяющий хранить и обрабатывать информацию о студентах группы (добавление информации, сортировку по различным критериям, поиск по различным критериям, вывод информации об успеваемости, выбор всех отличников, хорошистов, троечников, выбор всех студентов, имеющих средний балл не ниже заданного). Интерфейс класса должен включать перегруженные операторы. Предусмотреть возможность загрузки данных из файла.

7. Разработать класс для хранения текста произвольного объема. Предусмотреть ввод и поиск информации, добавление и удаление данных в произвольное место текста, объединение нескольких текстов в один и разбиение текста на части. Интерфейс класса должен включать перегруженные операторы. Предусмотреть возможность загрузки данных из файла.

Контрольные вопросы

1. В чем заключается отличие объектно-ориентированного программирования от структурированного? Перечислите достоинства и недостатки объектно-ориентированного подхода.

2. В чем заключается смысл инкапсуляции?

3. В чем заключается смысл полиморфизма?

4. В чем заключается смысл наследования?

5. Классы в C++. Их объявление, описание и использование.

6. Что такое конструкторы и деструкторы? Для чего они используются?

7. Каким образом можно передавать объекты в функции? Может ли функция возвращать объекты?

8. Что такое перегрузка функций? Для чего она используется?

9. Что такое перегрузка операторов? Для чего она используется?

10. Что означает ключевое слово *this* в тексте программы на C++?

11. Объясните работу перегруженного оператора присваивания в примере, предложенном в теоретических положениях. Почему он возвращает значение?

Практическое занятие № 18
Программная реализация принципов наследования и полиморфизма

Цель работы: Ознакомление с механизмом наследования классов и возможностями, которые он предоставляет; получение навыков использования наследования в прикладных программах.

Краткие теоретические сведения

Наследование и полиморфизм являются одними из основных элементов объектно-ориентированного подхода в программировании. С помощью наследования можно создать общий класс, определяющий общие черты совокупности объектов. Этот класс может наследоваться другими более специфическими классами, каждый из которых может добавить свои элементы в описании объекта. Есть еще одна причина, по которой наследованию придается большая важность: с его помощью поддерживается полиморфизм во время выполнения программы. В C++ полиморфизм поддерживается как на этапе выполнения программы, так и на этапе ее компиляции. В качестве примера полиморфизма на этапе компиляции можно указать перегрузку операторов и функций. Но, к сожалению, при ее использовании нельзя решить всех задач, возникающих в объектно-ориентированном программировании. Поэтому в C++ используется также полиморфизм времени исполнения программы, для чего используются производные классы и виртуальные функции.

Класс, который наследуется другим классом, называется базовым классом. Иногда его также называют родительским классом. Класс, выполняющий наследование, называется производным классом или потомком.

Когда один класс наследует другой, то все публичные члены базового класса доступны в производном классе. В противоположность этому частные члены базового класса не доступны внутри производного класса. Если необходимо оставить член частным и вместе с тем позволить использовать его производным классам, то для этих целей и используется ключевое слово *protected*. Защищенный член подобен частному за исключением механизма наследования. При наследовании защищенного члена производный класс также имеет к нему доступ. Таким образом, указав спецификатор доступа *protected*, можно позволить использовать член внутри иерархии классов и запретить доступ к нему извне этой иерархии. Общая форма наследования классов имеет следующий вид:

```
class <имя производного класса>: <доступ> <имя базового класса> {
……
<описание производного класса>;
……
};
```

Здесь доступ определяет, каким способом наследуется базовый класс. Спецификатор доступа может принимать три значения: *private*, *public* и *protected*. В случае, если спецификатор доступа опущен, то по умолчанию подразумевается на его месте спецификатор *public*. При этом все публичные и защищенные члены базового класса становятся соответственно публичными и защищенными членами производного класса. Если спецификатор доступа имеет значение *private*, то все публичные и защищенные члены базового класса становятся частными членами производного класса. И, наконец, если спецификатор доступа принимает значение *protected*, то все публичные и защищенные члены базового класса становятся защищенными членами производного класса. Например:

```
class X {
        protected:
                int i, j;
        public:
                void get_ij() {cin>>i>>j;}
```

```
                void put_ij() {cout<<i<<" "<<j<<endl;}
};
// В классе Y i и j класса X становятся частными членами
class Y: private X {
        protected:
                int k;
        public:
                int get_k() {return k;}
                void make_k() {k = i * j;}
};
/* Класс Z имеет доступ к k класса Y, т.к. он защищенный, но не имеет доступа к i и
j, т.к. они частные */
class Z : public Y {
public:
                void f();
};
void Z::f()
        {
        k = 5; // корректно
                i = j = 10; // не корректно, т.к. нет доступа
        }
```

При использовании производных классов важно представлять себе, каким образом и когда выполняются конструкторы и деструкторы базового и производного классов. Конструкторы вызываются в том же самом порядке, в каком классы следуют друг за другом в иерархии классов. Поскольку базовый класс ничего не знает про свои производные классы, то его инициализация может быть отделена от инициализации производных классов и производится до их создания, так что конструктор базового класса вызывается перед вызовом конструктора производного класса. В противоположность этому деструктор производного класса вызывается перед деструктором базового класса.

Когда базовый класс имеет конструктор с аргументами, производные классы должны явным образом обрабатывать эту ситуацию путем передачи базовому классу необходимых аргументов. Для этого используется расширенная форма конструкторов производных классов, в которые передаются аргументы конструкторам базовых классов:

<порожденный конструктор>(<список аргументов>): <базовый конструктор> (<список аргументов>)

```
        {
                ……
        }
```

Список аргументов, ассоциированный с базовым классом, может состоять из констант, глобальных переменных или параметров конструктора производного класса. Поскольку инициализация объекта происходит во время выполнения программы, можно использовать в качестве аргумента любой идентификатор, определенный в области видимости класса.

```
class X {
        protected:
                int a, b;
        public:
                X(int i, int j) {a = i; b = j;}
};
class Y : public X {
        private:
```

```
        int c;
    public:
        Z(int l, int m, int n): X(l, m) { c = n;}
}
void main(void)
    {
        Z ob(0,1,2);
    }
```

Также один класс может наследовать атрибуты двух и более классов одновременно. Для этого используется список базовых классов, в котором каждый из базовых классов отделен друг от друга запятой. Например:

```
class X {
    protected:
        int a;
public:
    void make_a(int p) { a = p; }
};
class Y {
    protected:
        int b;
public:
    void make_b(int q) { a = q; }
}
class Z: public X, public Y {
public:
    int make_ab() { return a*b; }
}
```

Поскольку класс *Z* наследует оба класса *X* и *Y*, то он имеет доступ к публичным и защищенным данным обоих классов. При множественном наследовании конструкторы базовых классов вызываются слева направо по порядку их следования в описании производного класса. Деструкторы же вызываются в обратном порядке – справа налево.

Имеется одна особенность использования указателей на классы при наследовании. В общем случае указатель одного типа не может указывать на объект другого типа. Из этого правила есть исключение, которое относится только к производным классам. В C++ указатель на базовый класс может указывать на объект производного класса, полученный из этого базового класса. Пусть имеем некий базовый класс *B_class* и его производный класс *D_class*. В C++ любой указатель типа *B_class** может также указывать на объект типа *D_class*. Например, если имеются следующие объявления переменных:

```
B_class *p;
B_class b_ob;
D_class d_ob;
```

то следующие присвоения абсолютно законны:

```
p = &b_ob;
p = &d_ob;
```

Используя указатель *p*, можно получить доступ ко всем членам *d_ob*, которые наследованы от *b_ob*. Однако специфические члены *d_ob* не могут быть получены с использованием указателя *p* (по крайней мере до тех пор, пока не будет осуществлено преобразование типов). Это является следствием того, что указатель «знает» только о членах базового типа и не знает ничего о специфических членах производных типов. Если необходимо получить доступ к элементам производного класса с помощью указателя,

имеющего тип указателя на базовый класс, необходимо воспользоваться приведением типов. Например: *((D_class*)p)->function();*

Хотя указатель, имеющий тип указателя на базовый класс, может использоваться в качестве указателя на производный объект, обратное не имеет место. Это означает, что указатель на производный класс не может использоваться для доступа к объектам базового типа.

Полиморфизм времени исполнения обеспечивается не только за счет использования производных классов, но и применением виртуальных функций. Виртуальная функция – это функция, объявленная с ключевым словом *virtua*l в базовом классе и переопределенная в одном или нескольких производных классах. Виртуальные функции являются особыми функциями, потому что при вызове объекта производного класса с помощью указателя или ссылки на него C++ генерирует код, который определяет во время исполнения программы, какую функцию вызвать, основываясь на типе объекта. Для разных объектов вызываются разные версии одной и той же виртуальной функции. Класс, содержащий одну или более виртуальных функций, называется полиморфным классом. Если функция объявлена как виртуальная, то она остается таковой вне зависимости от количества уровней в иерархии классов, через которые она прошла. Например, рассмотрим классы, содержащие информацию о геометрических фигурах:

```
class figure {
        protected:
                double x, y;
        public:
                void set(double a, double b) { x = a; y = b; }
                virtual double area() { cout<<”No area”; return 0; }
}
class triangle: public figure {
        public:
                double area() { return 0.5*x*y;}
}
class square: public figure {
        public:
                double area() { return x*y; }
}
class circle: public figure {
        protected:
                double r;
        public:
                circle(double a) { r = a;}
                double area() { return 3.14*r*r};
}
void main(void)
        {       double res;
                figure *p; // создание указателя базового типа
                triangle t;     square s;       circle c(10); // объекты производных типов
                p = &t;         p->set(1, 1);
                res = p->area(); // находим площадь треугольника
                p = &s;         p->set(2, 2);
                res = p->area(); // находим площадь четырехугольника
                p = &c;         p->set(20, 30);
                res = p->area(); // находим площадь окружности
        }
```

Ключевым моментом в использовании виртуальных функций для обеспечения полиморфизма времени исполнения служит то, что используется указатель именно на базовый класс. Полиморфизм времени исполнения достигается только при вызове виртуальной функции с использованием указателя или ссылки на базовый класс. Ничего не мешает использовать виртуальные функции, как и любые другие нормальные функции, однако достичь полиморфизма времени исполнения на этом пути не удастся. Также стоит отметить, что виртуальным может быть и деструктор класса, хотя конструктор виртуальным быть не может.

Когда виртуальная функция не переопределена в производном классе, то при вызове ее в объекте производного класса вызывается версия из базового класса. Однако во многих случаях невозможно ввести содержательное определение виртуальной функции в базовом классе. Например, при объявлении класса *figure* в предыдущем примере реализация функции *area()* не несет никакого смысла. Могут быть также такие виртуальные функции, которые обязательно должны быть переопределены в производных классах, без чего эти классы не будут иметь никакого значения. В таких случаях необходим метод, гарантирующий, что производные классы действительно определят все необходимые функции. Язык C++ предлагает в качестве решения этой проблемы чисто виртуальные функции. Это такие функции, которые объявлены в базовом классе, но не имеют в нем определения. Т.к. они не имеют определений, т.е. тел в этом базовом классе, то всякий производный класс обязан иметь свою собственную версию реализации. Для объявления чистой виртуальной функции используется следующая общая форма: virtual <тип возвращаемого значения> <имя функции> (<список параметров>) = 0;

Например, для определения чисто виртуальной функции area() в классе figure необходимо написать следующее: *virtual double area() = 0;*

Если какой-либо класс имеет хотя бы одну чисто виртуальную функцию, то такой класс называется абстрактным. Важной особенностью абстрактных классов является то, что нельзя создать ни одного объекта данного класса. Вместо этого абстрактный класс служит в качестве базового для других производных классов. Причина, по которой абстрактный класс не может быть использован для объявления объектов, заключается в том, что одна или несколько его функций-членов не имеют определения. Тем не менее, даже если базовый класс является абстрактным, все равно можно объявлять указатели или ссылки на него, с помощью которых затем поддерживается полиморфизм времени исполнения.

Порядок выполнения работы

1. Ознакомиться с теоретическими сведениями.
2. Получить вариант задания у преподавателя.
3. Выполнить задание.
4. Продемонстрировать выполнение работы преподавателю.
5. Оформить отчет.
6. Защитить лабораторную работу.

Варианты заданий

1. Разработать иерархию классов, которые позволяют оперировать с геометрическими фигурами: точка, линия, фигура, окружность, треугольник, четырехугольник, многоугольник, пирамида и призма. Реализация должна допускать создание объектов с различными параметрами, вычисление различных геометрических характеристик фигур, сопоставление одноименных фигур друг с другом. Интерфейс должен содержать виртуальные функции.

2. Разработать иерархию классов, которые позволяют создавать картотеку с информацией по предметам различного типа (книги, мебель, бытовая техника, компьютеры, продукты питания и т.д.). Реализация должна допускать добавление информации в картотеку, поиск информации, отбор по различным критериям, сортировку и

хранение наиболее часто просматриваемых карточек. Интерфейс должен содержать виртуальные функции.

3. Разработать иерархию классов, которые позволяют обрабатывать информацию о работниках различного уровня на предприятии (уборщики, рабочие, мастера и инженеры различных подразделений, экономисты, дирекция и т.д.). Реализация должна допускать прием на работу и увольнение сотрудников, их перевод из одного подразделения в другое, начисление ежемесячной зарплаты, учет отпусков, поиск и сортировку информации по различным критериям. Интерфейс должен содержать виртуальные функции.

4. Разработать иерархию классов, позволяющих моделировать работу системного блока персонального компьютера. Реализация должна позволять добавлять различные устройства в ПК, эмулировать их взаимодействие между собой, производить апгрейд аппаратуры. Интерфейс должен содержать виртуальные функции.

5. Разработать иерархию классов, описывающих существующее на рынке программное обеспечение. Реализация должна допускать классификацию ПО по различным критериям, добавление и удаление ПО в базу данных и из нее, выработку аргументированных (по каким критериям осуществлен выбор) рекомендаций пользователю по покупке программ для решения определенных задач. Интерфейс должен содержать виртуальные функции.

Контрольные вопросы

1. Что такое полиморфизм? Как он поддерживается в C++?
2. Что такое наследование классов? Для чего оно применяется?
3. В чем заключаются особенности описания конструкторов и деструкторов при наследовании? В каком порядке они вызываются при создании и удалении объектов производного типа?
4. В чем заключаются особенности использования указателей на базовые и производные классы?
5. Что такое виртуальные функции? Для чего они применяются?
6. Какие Вы знаете стандартные классы языка C++ ?

Практическое занятие № 19

Работа с графикой. Рисование графических примитивов

Краткие теоретические сведения

Необходимо спроектировать приложение, интерфейс которого включает форму, элемент вывода графики PictureBox, кнопки button1 и button2.

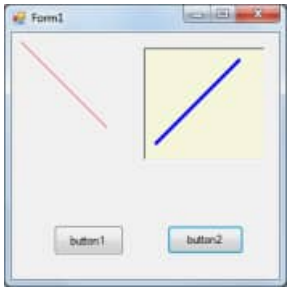
При клике по кнопке button1 на форме рисуется отрезок прямой красного цвета.

При клике по кнопке button2 в элементе PictureBox рисуется отрезок прямой синего цвета.

Разработаем интерфейс

а) после запуска

а) после нажатия button1



а) после нажатия button1

Порядок создания элементов управления

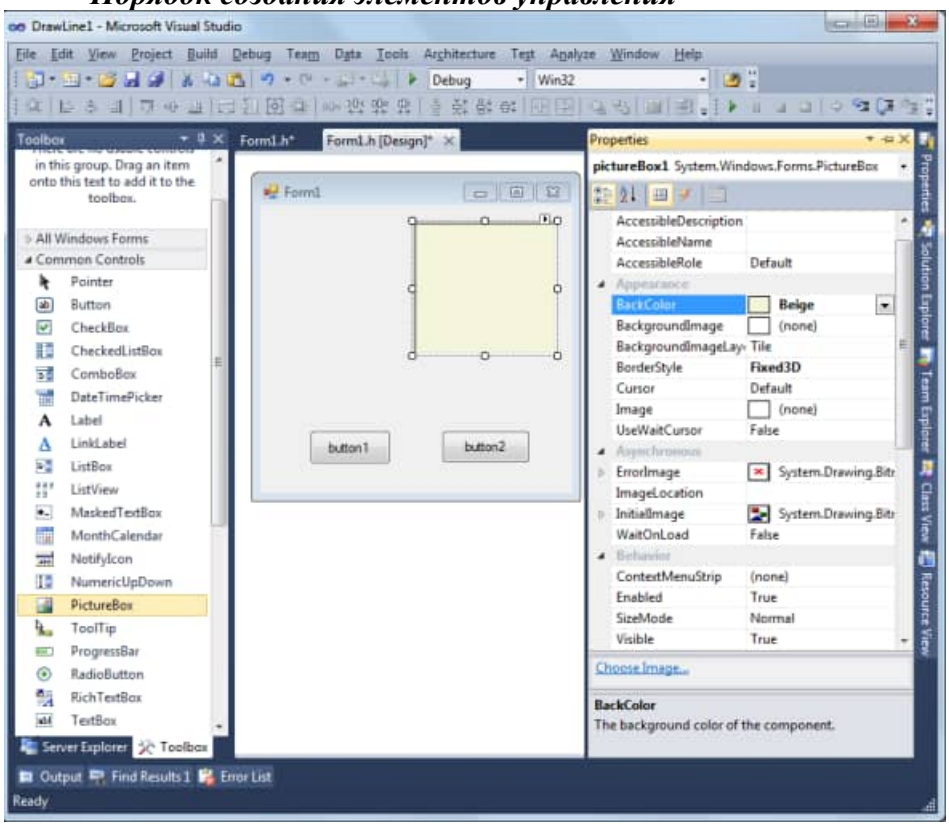


Рис. Окно дизайнера формы

Класс формы

```
#pragma once
namespace DrawLine1 {

	using namespace System;
	using namespace System::ComponentModel;
	using namespace System::Collections;
	using namespace System::Windows::Forms;
	using namespace System::Data;
	using namespace System::Drawing;

	public ref class Form1 : public System::Windows::Forms::Form
	{
	public:
		Form1(void)
		{
			InitializeComponent();
			//TODO: Add the constructor code here
		}

	protected:
		~Form1()
		{
			if (components)
			{
				delete components;
			}
		}
	private: System::Windows::Forms::Button^  button1;
	protected:
	private: System::Windows::Forms::Button^  button2;
	private: System::Windows::Forms::PictureBox^  pictureBox1;

	private:
		System::ComponentModel::Container ^components;
```

```
#pragma region Windows Form Designer generated code
            void InitializeComponent(void)
            {
                  this->button1 = (gcnew System::Windows::Forms::Button());
                  this->button2 = (gcnew System::Windows::Forms::Button());
                  this->pictureBox1 = (gcnew
System::Windows::Forms::PictureBox());
                  (cli::safe_cast<System::ComponentModel::ISupportInitialize^
>(this->pictureBox1))->BeginInit();
                  this->SuspendLayout();
                  // button1
                  this->button1->Location = System::Drawing::Point(45, 205);
                  this->button1->Name = L"button1";
                  this->button1->Size = System::Drawing::Size(74, 31);
                  this->button1->TabIndex = 0;
                  this->button1->Text = L"button1";
                  this->button1->UseVisualStyleBackColor = true;
                  this->button1->Click += gcnew System::EventHandler(this,
&Form1::button1_Click);
                  // button2
                  this->button2->Location = System::Drawing::Point(165, 205);
                  this->button2->Name = L"button2";
                  this->button2->Size = System::Drawing::Size(81, 30);
                  this->button2->TabIndex = 1;
                  this->button2->Text = L"button2";
                  this->button2->UseVisualStyleBackColor = true;
                  this->button2->Click += gcnew System::EventHandler(this,
&Form1::button2_Click);
                  // pictureBox1
                  this->pictureBox1->BackColor = System::Drawing::Color::Beige;
                  this->pictureBox1->BorderStyle =
System::Windows::Forms::BorderStyle::Fixed3D;
                  this->pictureBox1->Location = System::Drawing::Point(140, 16);
                  this->pictureBox1->Name = L"pictureBox1";
                  this->pictureBox1->Size = System::Drawing::Size(129, 120);
                  this->pictureBox1->TabIndex = 2;
                  this->pictureBox1->TabStop = false;
                  // Form1
                  this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
                  this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;
                  this->ClientSize = System::Drawing::Size(284, 262);
                  this->Controls->Add(this->pictureBox1);
                  this->Controls->Add(this->button2);
                  this->Controls->Add(this->button1);
                  this->Name = L"Form1";
                  this->Text = L"Form1";
                  (cli::safe_cast<System::ComponentModel::ISupportInitialize^
>(this->pictureBox1))->EndInit();
                  this->ResumeLayout(false);

            }
#pragma endregion
      private: System::Void button1_Click(System::Object^  sender,
System::EventArgs^  e) {
      Color ^col=gcnew Color;
      Pen ^pen= gcnew Pen(col->Red);
      Graphics ^im  =this->CreateGraphics();
      im->DrawLine(pen,10,10,100,100);
      }
      private: System::Void button2_Click(System::Object^  sender,
System::EventArgs^  e) {
      Color ^col=gcnew Color;
      Pen ^pen= gcnew Pen(col->Blue,4);
      Graphics ^im  =pictureBox1->CreateGraphics();
      im->DrawLine(pen,10,100,100,10);
      }
```

```
};
}
```

Порядок выполнения работы

1. Получить задание у преподавателя.
2. Разработать алгоритм решения задачи и написать программу, реализующую задание.
3. Проверить правильность ее работы.
2. Составить отчет и защитить работу.

Варианты заданий

1. Спроектировать приложение для рисования отрезка прямой линии на форме. На форме предусмотреть кнопку для ввода данных,кнопу для рисования, поле ввода. В поле ввода задаются параметры линии.
2. Спроектировать приложение для рисования эллипса на форме. На форме предусмотреть кнопку для ввода данных,кнопу для рисования, поле ввода. В поле ввода задаются параметры линии.
3. Спроектировать приложение для рисования окружности на форме. На форме предусмотреть кнопку для ввода данных,кнопу для рисования, поле ввода. В поле ввода задаются параметры линии.
4. Спроектировать приложение для рисования сектора эллипса на форме. На форме предусмотреть кнопку для ввода данных,кнопу для рисования, поле ввода. В поле ввода задаются параметры линии.
5. Спроектировать приложение для рисования хорды эллипса на форме. На форме предусмотреть кнопку для ввода данных,кнопу для рисования, поле ввода. В поле ввода задаются параметры линии.
6. Спроектировать приложение для рисования под произвольным углом эллипса на форме. На форме предусмотреть кнопку для ввода данных,кнопу для рисования, поле ввода. В поле ввода задаются параметры линии, значение угла поворота.
7. Спроектировать приложение для рисования графика функции на форме. На форме предусмотреть кнопку для ввода данных,кнопу для рисования, поле ввода. В поле ввода задаются параметры линии графика,таблица значений функции.
8. Спроектировать приложение для рисования изометрии пирамиды на форме. На форме предусмотреть кнопку для ввода данных,кнопу для рисования, поле ввода. В поле ввода задаются параметры линий,таблица значений координат вершин пирамиды.
9. Спроектировать приложение для рисования изометрии параллелепипеда на форме. На форме предусмотреть кнопку для ввода данных,кнопу для рисования, поле ввода. В поле ввода задаются параметры линий,таблица значений координат вершин параллелепипеда.
10. Спроектировать приложение для рисования изометрии параллелограмма в декартовой системе координат на форме. На форме предусмотреть кнопку для ввода данных,кнопу для рисования, поле ввода. В поле ввода задаются параметры линий, таблица значений координат вершин параллелограмма.

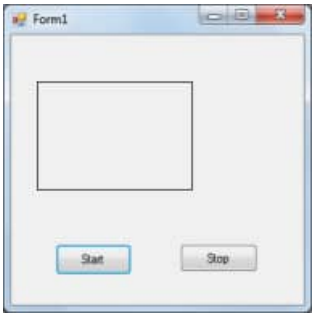
Практическое занятие № 20

Работа с графикой. Рисование анимированных объектов

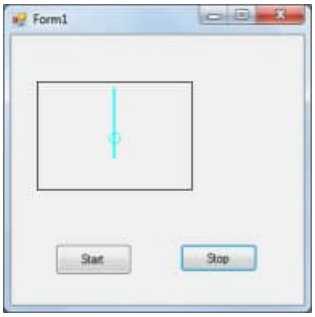
Краткие теоретические сведения

Спроектируем приложение, интерфейс которого включает форму, кнопку Start, кнопку Stop. При инициализации формы на ней рисуется прямоугольный контур таймера. При клике по кнопке Start в на форме рисуется секундная стрелка синим цветом каждую нечетную секунду и красным цветом – каждую четную.

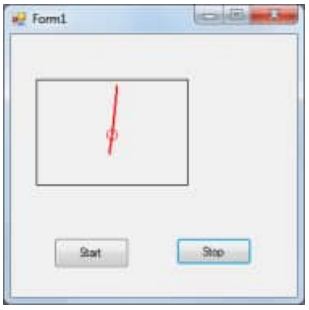
Разработка интерфейса



а) после запуска

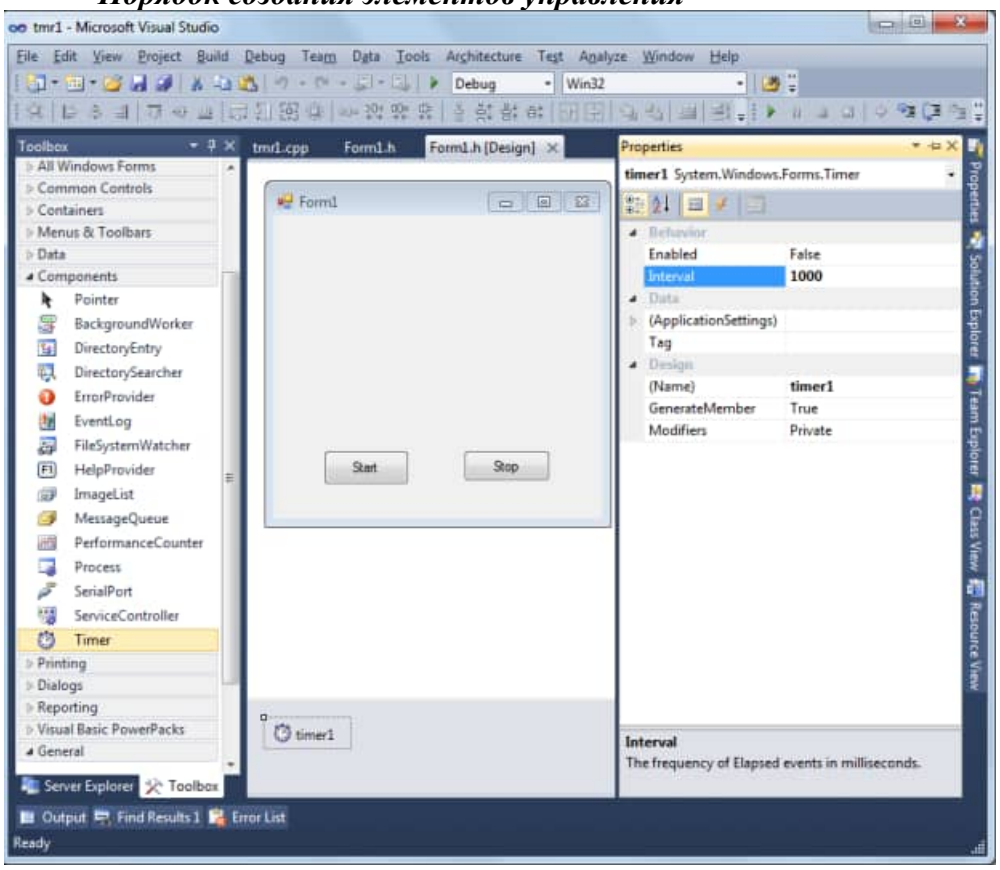


а) после нажатия start



а) после истеченияодной секунды

Порядок создания элементов управления



Класс формы

```
#pragma once
#include <math.h>
#include <stdlib.h>
using namespace std;
namespace tmr1 {

	using namespace System;
	using namespace System::ComponentModel;
	using namespace System::Collections;
	using namespace System::Windows::Forms;
	using namespace System::Data;
	using namespace System::Drawing;

	public ref class Form1 : public System::Windows::Forms::Form
	{
	public:
	Form1(void)
	{
		InitializeComponent();
		//TODO: Add the constructor code here
		im=this->CreateGraphics();
		col=gcnew Color;
		pen =gcnew Pen(col->Red);
		// Build the rectangles from points and size
		Drawing::Point point1 = Drawing::Point(25,45);
		Drawing::Size size = Drawing::Size(150, 105);
		rect1 = Drawing::Rectangle(point1, size);
```

```
        }
        protected:
        ~Form1()
        {
                if (components)
                {
                delete components;
                }
        }
        private: System::Windows::Forms::Button^  button1;
        protected:
        private: System::Windows::Forms::Button^  button2;
        private: System::Windows::Forms::Timer^  timer1;
        private: System::ComponentModel::IContainer^  components;
        private:
         Color ^col;
         Graphics ^im;
         Pen ^ pen ;
         Drawing::Rectangle rect1;
#pragma region Windows Form Designer generated code
        void InitializeComponent(void)
        {
                this->components = (gcnew System::ComponentModel::Container());
                this->button1 = (gcnew System::Windows::Forms::Button());
                this->button2 = (gcnew System::Windows::Forms::Button());
                this->timer1 = (gcnew System::Windows::Forms::Timer(this->components));
                this->SuspendLayout();
                this->Paint +=
                gcnew System::Windows::Forms::PaintEventHandler(this,
                &Form1::Form1_Paint);
                // button1
                this->button1->Location = System::Drawing::Point(43, 203);
                this->button1->Name = L"button1";
                this->button1->Size = System::Drawing::Size(74, 30);
                this->button1->TabIndex = 0;
                this->button1->Text = L"Start";
                this->button1->UseVisualStyleBackColor = true;
this->button1->Click += gcnewSystem::EventHandler(this,&Form1::button1_Click);
                // button2
                this->button2->Location = System::Drawing::Point(164, 203);
                this->button2->Name = L"button2";
                this->button2->Size = System::Drawing::Size(75, 27);
                this->button2->TabIndex = 1;
                this->button2->Text = L"Stop";
                this->button2->UseVisualStyleBackColor = true;
                this->button2->Click += gcnew System::EventHandler(this, &Form1::button2_Click);
                // timer1
                this->timer1->Interval = 1000;
                this->timer1->Tick += gcnew System::EventHandler(this, &Form1::timer1_Tick);
                // Form1
                this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
                this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
```

```
            this->ClientSize = System::Drawing::Size(284, 262);
            this->Controls->Add(this->button2);
            this->Controls->Add(this->button1);
            this->Name = L"Form1";
            this->Text = L"Form1";
            this->ResumeLayout(false);

        }
#pragma endregion
//----------My code--------------
        private:
        System::Void Form1_Paint(System::Object^ sender,
System::Windows::Forms::PaintEventArgs^ e)
            {
            // Draw a rectangle
            e->Graphics->DrawRectangle(Pens::Black, rect1);
            }

        private: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e) {
            if(timer1->Enabled);else timer1->Enabled=true;
             im->DrawRectangle(Pens::Black, rect1);}
        private: System::Void button2_Click(System::Object^  sender, System::EventArgs^  e) {
             if(!timer1->Enabled);else timer1->Enabled=false; }
        private: System::Void timer1_Tick(System::Object^  sender, System::EventArgs^  e) {
            static float x=100,y=100,fi=0,hfi=3.14159/30,fil,pi=3.14159 ,
            x1=100,y1=100,xl,yl,x1l,y1l,cx=100,
            cy=100,l1=20,l=50,ts=0;
            x1l=x1;y1l=y1;xl=x;yl=y;
        static int t;
        t++;
        y=100-l*sin(fi+pi/2);
        x=100-l*cos(fi+pi/2);
        x1=cx+l1*cos(fi+pi/2);
        y1=cy+l1*sin(fi+pi/2);
        fi=fi+hfi;
        pen->Color=this->BackColor;
        pen->Width=2;
        im->DrawLine(pen,int(xl),int(yl),100,100);
        im->DrawLine(pen,int(x1l),int(y1l),int(cx),int(cy));
        if(t%2)pen->Color=col->Cyan;else pen->Color=col->Red;
        im->DrawLine(pen,int(x1),int(y1),int(cx),int(cy));
        im->DrawLine(pen,int(x),int(y),100,100);
        im->DrawEllipse((pen->Width=1,pen),int(cx-5),int(cy-5),10,10);
             }
        };
}
```

Порядок выполнения работы

1. Получить задание у преподавателя.
2. Разработать алгоритм решения задачи и написать программу, реализующую задание.

3. Проверить правильность ее работы.
4. Составить отчет и защитить работу.

Варианты заданий

1. Спроектировать приложение для рисования изометрии отрезка прямой линии в трехмерной системе координат на форме. На форме предусмотреть кнопку для ввода данных,кнопу для рисования, поле ввода, кнопку для однократного поворота объекта. Поворот осуществляется вокруг оси, параметры которой задаются в поле ввода. В поле ввода задаются также параметры линии.
2. Спроектировать приложение для рисования для рисования изометрии эллипса в трехмерной системе координат на форме. На форме предусмотреть кнопку для ввода данных,кнопу для рисования, поле ввода, кнопку для однократного поворота объекта. Поворот осуществляется вокруг оси, параметры которой задаются в поле ввода. В поле ввода задаются также параметры линии.
3. Спроектировать приложение для рисования для рисования изометрии окружности в трехмерной системе координат на форме. На форме предусмотреть кнопку для ввода данных,кнопу для рисования, поле ввода, кнопку для однократного поворота объекта. Поворот осуществляется вокруг оси, параметры которой задаются в поле ввода. В поле ввода задаются также параметры линии.
4. Спроектировать приложение для рисования для рисования изометрии сектора эллипса в трехмерной системе координат на форме. На форме предусмотреть кнопку для ввода данных,кнопу для рисования, поле ввода, кнопку для однократного поворота объекта. Поворот осуществляется вокруг оси, параметры которой задаются в поле ввода. В поле ввода задаются также параметры линии.
5. Спроектировать приложение для рисования для рисования изометрии и хорды эллипса в трехмерной системе координат на форме. На форме предусмотреть кнопку для ввода данных,кнопу для рисования, поле ввода, кнопку для однократного поворота объекта. Поворот осуществляется вокруг оси, параметры которой задаются в поле ввода. В поле ввода задаются также параметры линии.
6. Спроектировать приложение для рисования изометрии графика функции на форме. На форме предусмотреть кнопку для ввода данных, кнопу для рисования, поле ввода, кнопку для однократного поворота объекта. Поворот плоскости графика функции осуществляется вокруг оси, параметры которой задаются в поле ввода. В поле ввода задаются параметры линии графика,таблица значений функции.
7. Спроектировать приложение для рисования изометрии пирамиды на форме. На форме предусмотреть кнопку для ввода данных, кнопу для рисования, поле ввода, кнопку для однократного поворота объекта. Поворот изображения объекта осуществляется вокруг оси, параметры которой задаются в поле ввода. В поле ввода задаются координаты вершин пирамиды.
8. Спроектировать приложение для рисования изометрии параллелепипеда на форме. На форме предусмотреть кнопку для ввода данных, кнопу для рисования, поле ввода, кнопку для однократного поворота объекта. Поворот изображения объекта осуществляется вокруг оси, параметры которой задаются в поле ввода. В поле ввода задаются параметры линий,таблица значений координат вершин параллелепипеда.
9. Спроектировать приложение для рисования изометрии параллелограмма в декартовой системе координат на форме. На форме предусмотреть кнопку для ввода данных, кнопу для рисования, поле ввода, кнопку для однократного поворота объекта. Поворот изображения объекта осуществляется вокруг оси, параметры которой задаются в поле ввода В поле ввода задаются параметры линий, таблица значений координат вершин параллелограмма.

10. Спроектировать приложение для рисования изометрии ограниченного параболоида вращения в декартовой системе координат на форме. На форме предусмотреть кнопку для ввода данных, кнопу для рисования, поле ввода, кнопку для однократного поворота объекта. Поворот изображения объекта осуществляется вокруг оси, параметры которой задаются в поле ввода В поле ввода задаются параметры линий, таблица значений параметров параболоида.

Минобрнауки России
ФГБОУ ВО «Тульский государственный университет»
Технический колледж им.С.И.Мосина

Методические указания по выполнению практических работ

по междисциплинарному курсу
МДК.03.01 Техническое обслуживание и ремонт аппаратной части компьютерных систем и комплексов

профессионального модуля
ПМ.03 Техническое обслуживание и ремонт компьютерных систем и комплексов

специальности 09.02.01
Компьютерные системы и комплексы

Тула 2023

УТВЕРЖДЕНЫ
Цикловой комиссией информационных технологий
Протокол от «13» января 2023 г. № 6
Председатель цикловой комиссии И.В. Миляева

ПРАКТИЧЕСКАЯ РАБОТА № 1

Тема учебной программы: Компоненты компьютерной системы

Тема работы: Блок питания

Цель работы: определять технические характеристики блока питания

Материально-техническое оснащение: конспект лекций; блоки питания разных форм-факторов; видеоурок «Блок питания»; мультиметр

Количество часов: 2

КРАТКИЕ СВЕДЕНИЯ

Блок питания преобразует переменный ток высокого напряжения в постоянный и стабилизированный тон низкого напряжения, который питает все внутренние компоненты системного блока. Если какому-либо компоненту требуется другое напряжение, он либо сам преобразует имеющееся питание, либо использует питание, переработанное стабилизаторами системной платы;

Логически блок питания разделяют на несколько модулей, каждый из которых выполняет определённую задачу. Поступая на блок питания, переменное напряжение проходит через сетевой фильтр и обрабатывается высоковольтным выпрямителем. Выпрямленное напряжение через высоковольтный фильтр поступает на импульсный трансформатор, понижающий напряжение до нужного уровня. Пониженное постоянное напряжение поступает на стабилизатор, который контролирует и при необходимости преобразовывает его характеристики. В итоге получается несколько видов напряжения с необходимыми характеристиками.

ЗАДАЧИ И УПРАЖНЕНИЯ

1 Рассчитать необходимую мощность блока питания для компьютерной системы, имеющей характеристики:

- Pentium II/4 DIMM, AGP, 5 PCI, Flash BIOS/DIMM 64 Mb SDRAM/HDD UDMA-66/FDD/40x CD/SB Creative Labs, PCI
- Pentium III/2 DIMM, AGP, 2 PCI, 1 PCI-E x16 Gigabit LAN/DIMM 1 Gb DDR2/HDD SATA-II/48x DVD/SB Creative Labs, PCI
- AMD K-5/2 DIMM, AGP, 4 PCI/DIMM 256 Mb SDRAM/HDD UDMA-66/FDD/24x CD, SB, AWE, PCI

2 Открыть блок питания и, используя мультиметр, проверить:

- высоковольтный выпрямитель;
- высоковольтный фильтр;
- стабилизатор.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Каково основное отличие блока питания ATX по сравнению с AT?

2 Пояснить роль сигнала Power Good, вырабатываемого блоком питания?

3 Какие выходные напряжения вырабатывает блок питания компьютера?

4 Какой способ модуляции применяется при регулировании уровня выходных напряжений блок питания?

5 Для чего предназначен предохранитель в блоке питания?

ПРАКТИЧЕСКАЯ РАБОТА № 2

Тема учебной программы: Компоненты компьютерной системы

Тема работы: Системная плата

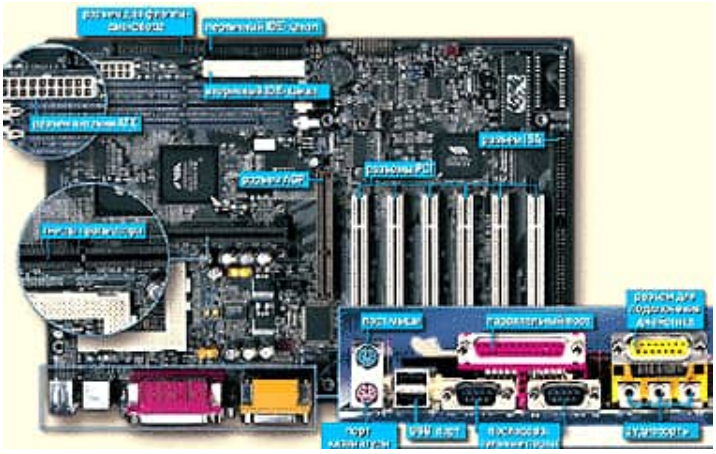
Цель работы: определять технические характеристики системной платы

Материально-техническое оснащение: конспект лекций; системные платы разных форм-факторов; видеоурок «Системная плата»

Количество часов: 2

КРАТКИЕ СВЕДЕНИЯ

Системная плата – определяет тип процессора и оперативной памяти, которые могут быть установлены в компьютер. От форм-фактора системной платы также зависит количество расположенных на ней слотов расширения, интегрированных контроллеров и т.п.;



ЗАДАЧИ И УПРАЖНЕНИЯ

Определить спецификацию системной платы:

- форм-фактор;
- процессорное гнездо;
- чипсет;
- тип BIOS;
- разъём блока питания;
- разъём RAM;
- слоты плат расширения (тип и количество);
- коннекторы и разъёмы (тип и количество);
- локальные порты (название, тип разъёма, количество).

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Почему системную плату также называют «объединительной»?
1 Какие функции выполняет чипсет?
2 Для чего используется CMOS-память?

ПРАКТИЧЕСКАЯ РАБОТА № 3

Тема учебной программы: Компоненты компьютерной системы

Тема работы: Процессор

Цель работы: определять технические характеристики процессоров

Материально-техническое оснащение: конспект лекций; процессоры разных форм-факторов; видеоурок «Процессор»

Количество часов: 2

КРАТКИЕ СВЕДЕНИЯ

Процессор выполняет операции, заданные программой. Процессор имеет множество характеристик, с помощью которых можно сравнивать различные модели. Интерфейс процессора определяет особую форму процессорного слота на системной плате.

ЗАДАЧИ И УПРАЖНЕНИЯ

Определить технические характеристики процессора, применяемого в компьютерной системе.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Что означает термин ZIF?
2 Перечислить три основных режима энергосбережения и объяснить их различия.

ПРАКТИЧЕСКАЯ РАБОТА № 4

Тема учебной программы: Компоненты компьютерной системы

Тема работы: Оперативная память

Цель работы: определять технические характеристики оперативной памяти

Материально-техническое оснащение: конспект лекций; модули оперативной памяти разных форм-факторов; видеоурок «Оперативная память»

Количество часов: 2

КРАТКИЕ СВЕДЕНИЯ

Оперативная память обеспечивает своевременное предоставление компонентам системы необходимой информации. От объёма и алгоритма работы оперативной памяти зависит быстродействие всей системы.

Модули памяти различаются количеством контактов, объёмом, рабочей частотой системной шины, на которую они рассчитаны, типом установленных микросхем и др.

ЗАДАЧИ И УПРАЖНЕНИЯ

Определить характеристики модулей RAM.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 На каком принципе основана работа DRAM?
2 Как определяется общая ёмкость модуля памяти с байтах?
3 В чём заключается идея чередования банков памяти:

ПРАКТИЧЕСКАЯ РАБОТА № 5

Тема учебной программы: Компоненты компьютерной системы

Тема работы: Видеокарта

Цель работы: определять технические характеристики видеокарты

Материально-техническое оснащение: конспект лекций; видеокарты разных форм-факторов; видеоурок «Видеокарта»

Количество часов: 2

КРАТКИЕ СВЕДЕНИЯ

Видеокарта обеспечивает интерфейс между компьютером и монитором, передавая сигналы, которые превращаются в изображение, видимое на экране.

ЗАДАЧИ И УПРАЖНЕНИЯ

1 В целевом компьютере определить тип видеадаптера.

2 Определить характеристики видеокарты.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Что означает характеристика SLI-Ready?
2 В чём особенность передачи аналоговых сигналов?
3 Что характеризует графический процессор?

ПРАКТИЧЕСКАЯ РАБОТА № 6

Тема учебной программы: Компоненты компьютерной системы

Тема работы: Звуковая карта

Цель работы: определять технические характеристики звуковой карты; установка и настройка звуковой карты

Материально-техническое оснащение: конспект лекций; звуковые карты разных форм-факторов; видеоурок «Звуковая карта»

Количество часов: 2

КРАТКИЕ СВЕДЕНИЯ

Частоты звуковых (слышимых) колебаний лежат в диапазоне от 17-20 Гц до 20 кГц. Реальные звуки помимо громкости и частоты также характеризуются тембром – кроме основного тона в сигнале присутствуют также колебания более высоких частот обертона. Именно амплитудами обертонов и характеризуется тембр (насыщенность) звука.

В компьютере имеется несколько возможностей для генерирования звука с использованием звуковой карты. Выбор конкретного способа, в первую очередь, зависит от типа конкретной звуковой карты.

ЗАДАЧИ И УПРАЖНЕНИЯ

1 Указать тип звуковой карты.

2 Определить характеристики звуковой карты:
- разрядность;
- максимальная частота дискретизации;
- наличие MIDI-интерфейса.

3 Отметить количество операторов звуковой карты и рассчитать количество музыкальных инструментов, которые могут быть сгенерированы одновременно.

4 Указать типы звукового синтеза, реализуемые звуковой картой.

5 Установить звуковую карту в целевой компьютер.

6 Осуществить прослушивание звукового файла, используя WT- и FM-синтез. Оценить качество и реалистичность звучания.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Какие дополнительные разъёмы имеет звуковая карта?
2 Перечислить критерии выбора звуковой карты.
3 Пояснить технологию позиционирования звука?

ПРАКТИЧЕСКАЯ РАБОТА № 7

Тема учебной программы: Компоненты компьютерной системы

Тема работы: Сетевая карта

Цель работы: определять технические характеристики сетевой карты

Материально-техническое оснащение: конспект лекций; сетевые карты разных форм-факторов; видеоурок «Сетевая карта»

Количество часов: 2

КРАТКИЕ СВЕДЕНИЯ

Сетевая карта обеспечивает подключение компьютера к сети. Для настройки сетевой карты необходимо указать несколько адресов портов ввода-вывода и один канал прерывания, а также дополнительные 16 Кбайт свободной верхней памяти для создания буфера.

ЗАДАЧИ И УПРАЖНЕНИЯ

1 В целевом компьютере определить тип сетевого адаптера.

2 Определить характеристики сетевой карты.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Почему нельзя использовать ресурсы сетевой карты совместно с другими устройствами?
2 Что обеспечивает протокол согласования режимов Ethernet?

ПРАКТИЧЕСКАЯ РАБОТА № 8

Тема учебной программы: Конфигурирование компьютерной системы

Тема работы: Конфигурация компьютерной системы

Цель работы: определять базовую и расширенную конфигурацию компьютерной системы

Материально-техническое оснащение: конспект лекций; целевой компьютер; операционная система; программа WinCheckit

Количество часов: 2

КРАТКИЕ СВЕДЕНИЯ

Конфигурация компьютерной системы определяется визуально или с помощью средств операционной системы или специализированных программ.

Для обоих методов определения конфигурации определяют элементы:

- тип процессора;
- тактовая частота процессора;
- тип BIOS;
- количество подключённых накопителей;
- размер ОЗУ и кэш-памяти;
- параметры контроллера клавиатуры;
- наличие дополнительных контроллеров и адаптеров;
- тип системной и локальной шины и их характеристики.

ЗАДАЧИ И УПРАЖНЕНИЯ

1 Определить конфигурацию целевого компьютера визуально. Составить формуляр компьютерной системы.

2 Определить конфигурацию целевого компьютера с помощью средств операционной системы. Составить формуляр компьютерной системы.

3 Выполнить сравнение точности визуального и программного методов определения конфигурации компьютерной системы.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Перечислить устройства, входящие в минимальную конфигурацию компьютерной системы?

2 Какие факторы необходимо учитывать при правильном выборе конфигурации компьютерной системы?

ПРАКТИЧЕСКАЯ РАБОТА № 9

Тема учебной программы: Конфигурирование компьютерной системы

Тема работы: Компоновка компьютерной системы

Цель работы: устанавливать компоненты внутрь системного блока и обеспечивать правильность их подсоединения; подключать органы управления с лицевой панели системного блока

Материально-техническое оснащение: конспект лекций; компоненты системного блока разных форм-факторов; видеоуроки «Устройство корпуса», «Самостоятельная сборка ПК», «Подключение устройств передней панели корпуса»

Количество часов: 4

КРАТКИЕ СВЕДЕНИЯ

При сборке типичного компьютера используют следующие компоненты:

- корпус;
- блок питания;
- процессор с теплоотводным элементом;
- оперативная память;
- видеокарта;
- звуковая карта;
- сетевой адаптер;
- накопитель на жёстких дисках;
- накопитель на оптических дисках;
- клавиатура;
- устройство позиционирования;
- монитор;
- акустическая система.

Системная плата может иметь интегрированные элементы: звуковая карта; видеокарта; сетевой адаптер.

В качестве дополнительных элементов используют кабели и крепёжные компоненты.

Внимание! Не забудьте подсоединить кабель питания вентилятора процесса к специальному разъёму на системной плате. Если этого не сделать процессор может перегреться.

После компоновки системного блока, подключить периферийные устройства (монитор, клавиатура) и проверить систему на работоспособность.

ЗАДАЧИ И УПРАЖНЕНИЯ

1 Установить необходимые компоненты на системную плату вне корпуса.

2 Установить укомплектованную системную плату в корпус.

3 Установить блок питания внутрь корпуса.

4 Выполнить необходимые подключения.

5 Установить устройства хранения и выполнить их подключение.

6 Выполнить подключение лицевой панели системного корпуса к системной плате.

7 Проверить работоспособность укомплектованного системного блока.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Какие меры безопасности нужно предпринимать при подключении устройств?
2 Каково основное правило модернизации компьютерной системы?
3 Объяснить отличие «холодной» загрузки от «горячей» загрузки.
4 Какой эффект возможен при использовании модулей RAM с различной тактовой частотой?
5 При компоновке компьютерной системы, что понимают под «новым устройством»?

ПРАКТИЧЕСКАЯ РАБОТА № 10

Тема учебной программы: Конфигурирование компьютерной системы

Тема работы: Программа Setup BIOS

Цель работы: конфигурировать настройки CMOS Setup для получения полнофункциональной системы

Материально-техническое оснащение: конспект лекций; целевой компьютер; видеоурок «Основы работы с BIOS»

Количество часов: 2

КРАТКИЕ СВЕДЕНИЯ

В процессе загрузки компьютера BIOS сравнивает получаемые данные о конфигурации системы с информацией, хранящейся в специальной микросхеме памяти на системной плате. Если данные не совпадают, на экран выдаётся сообщение и необходимо с помощью утилиты CMOS Setup установить новые конфигурационные параметры.

В BIOS можно войти в процессе начальной загрузки, нажав клавишу Delete. Утилита имеет интерфейс в виде системы иерархического меню, перемещение по которому производится с помощью клавиш управления курсором.

После установки новых параметров конфигурации, информацию в BIOS необходимо сохранить. Все изменения, вносимые в BIOS, записываются в CMOS-память и используются при следующей загрузке компьютера.

ЗАДАЧИ И УПРАЖНЕНИЯ

1 Через параметры BIOS скорректировать показания системного календаря и часов.

2 В BIOS установить последовательность загрузки операционной системы: 1 – с оптического диска; 2 – с жёсткого диска. Отключить обращение к FDD.

3 Используя функции BIOS настроить метод обработки ошибок, позволяющий в случае обнаружения ошибки продолжить работу, нажав клавишу на клавиатуре.

4 В BIOS настроить параметры, отвечающие за работу энергосберегающих механизмов:
- использование ACPI;
- шаблон управления питанием;
- время перехода в спящий режим.

5 Используя функции BIOS определить прерывание и физический адрес портов: COM1, COM2, LPT.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Перечислить группы системных ресурсов.

2 Какие функции BIOS должны быть заблокированы при обновлении операционной системы?

3 Какое действие необходимо предпринять, если неизвестен пароль BIOS?

4 При каких условиях нужно обновлять BIOS?

5 Как можно загрузить стандартные заводские установки BIOS?

ПРАКТИЧЕСКАЯ РАБОТА № 11

Тема учебной программы: Модернизация компьютерной системы

Тема работы: Замена процессора и оперативной памяти

Цель работы: замена процессора на более мощный; замена оперативной памяти с увеличением объёма

Материально-техническое оснащение: конспект лекций; целевой компьютер; процессор

Количество часов: 2

КРАТКИЕ СВЕДЕНИЯ

1 Замена процессора. При замене процессора учитывают сокет, тактовую частоту системной шины и параметры BIOS. Перед установкой процессора в плоское гнездо (Socket) поднять его рычажок. Аккуратно вставить процессор (убедиться, что первый контакт процессора совпадает с первым контактом гнезда), плотно прижать его и опустить рычажок.

Внимание! Устанавливать процессор в разъём следует с учётом ключа.

2 Замена оперативной памяти. При замене оперативной памяти необходимо учитывать тип модуля и его тактовую частоту. Также при модернизации модулей RAM учитывают правила формирования банка данных для правильного определения общего объёма установленной памяти.

Вставлять модуль (или модули) памяти DIMM так, как указано либо в Руководстве пользователя на системную плату, либо на ней самой, причём начинают с разъёма, обозначенного как Bank 0 (нулевой банк).

Устанавливать следует только так: плавно поворачивать модули до тех пор, пока расположенные с каждой стороны разъёма зажимы плотно не зафиксируют их в нужном положении.

Внимание! На системную плату следует устанавливать однотипные модули оперативной памяти.

ЗАДАЧИ И УПРАЖНЕНИЯ

1 Выполнить замену процессора.

2 Проверить работоспособность системы.

3 Выполнить замену модулей оперативной памяти.

4 Проверить работоспособность системы.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Как определить, какой тип памяти можно использовать в системе?
2 Почему процессор может работать на неправильной тактовой частоте?

ПРАКТИЧЕСКАЯ РАБОТА № 12

Тема учебной программы: Модернизация компьютерной системы

Тема работы: Замена системной платы

Цель работы: замена системной платы с улучшенными техническими характеристиками

Материально-техническое оснащение: конспект лекций; целевой компьютер; системная плата

Количество часов: 2

КРАТКИЕ СВЕДЕНИЯ

Схема подготовки к работе по замене системной платы:

- записать существенные параметры CMOS Setup;
- открыть корпус;
- замаркировать старые платы расширения;
- вытащить системную плату и на столе: заменить процессор; заменить модули оперативной памяти;
- установить системную плату в корпус;
- установить и подключить накопители информации;
- установить карты расширения;
- подключить лицевую панель системного блока.

ЗАДАЧИ И УПРАЖНЕНИЯ

1 Выполнить замену системной платы, с учётом имеющихся компонентов.

2 Проверить работоспособность системы.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Какие компоненты на системной плате поддаются модернизации?
2 В каких случаях нужно рассмотреть необходимость обновления BIOS?

ПРАКТИЧЕСКАЯ РАБОТА № 13

Тема учебной программы: Модернизация компьютерной системы

Тема работы: Замена карт расширений

Цель работы: замена карты расширений с улучшенными техническими характеристиками

Материально-техническое оснащение: конспект лекций; целевой компьютер; карты расширений с разными техническими характеристиками

Количество часов: 2

КРАТКИЕ СВЕДЕНИЯ

Установка карт расширения осуществляется внутри системного корпуса. При модернизации карт расширения проверяют их поддержку со стороны системной платы и BIOS.

Внимание!

1 После установки карты расширения, её обязательно фиксируют винтом в корпусе.
2 При модернизации видеокарты учитывают интерфейс – AGP или PCI-X.

ЗАДАЧИ И УПРАЖНЕНИЯ

1 Выполнить замену видеокарты.

2 Выполнить замену звуковой карты.

3 Выполнить замену сетевой карты.

4 Проверить работоспособность системы.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Какое действие необходимо выполнять перед установкой плат расширения в систему?
2 Пояснить особенности интегрированных компонентов.

ПРАКТИЧЕСКАЯ РАБОТА № 14

Тема учебной программы: Модернизация компьютерной системы

Тема работы: Замена блока питания

Цель работы: замена блока питания с улучшенными техническими характеристиками

Материально-техническое оснащение: конспект лекций; кабели разных категорий

Количество часов: 2

КРАТКИЕ СВЕДЕНИЯ

Блок питания – самое важное устройство в компьютере. Срок службы блока питания составляет 3-7 лет. О приближающейся «кончине» блока питания свидетельствуют:

- внезапные перезагрузки или зависания компьютера во время обычной работы;
- ошибки оперативной памяти при начальном тестировании и при работе;
- прекращение работы устройств хранения данных;
- чрезмерное повышение температуры в блоке питания и корпусе;
- появление напряжения на корпусе;
- появление странных ошибок в работе операционной системы и программ.

ЗАДАЧИ И УПРАЖНЕНИЯ

1 Выполнить замену блока питания.

2 Проверить работоспособность системы.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Какую информацию необходимо знать о блоке питания для его замены?
2 Что означает запас по току?

ПРАКТИЧЕСКАЯ РАБОТА № 15

Тема учебной программы: Операционные системы

Тема работы: Установка операционной системы

Цель работы: определять семейство операционной системы; выполнять подготовку жёсткого диска к установке операционной системы; выполнять инициализацию установки операционной системы

Материально-техническое оснащение: конспект лекций; целевой компьютер; дистрибутивы операционных систем; видеоурок «Установка Windows XP»

Количество часов: 2

КРАТКИЕ СВЕДЕНИЯ

Главными составляющими операционной системы являются ядро, системные утилиты, драйверы и графическая оболочка. Каждый программный элемент выполняет свою функцию и следит, чтобы сохранялась стабильность работы операционной системы в целом.

Существует множество операционных систем, и каждая имеет свою степень распространённости. Некоторые более удобны для работу в сети, а другие – для автономной работы, поэтому совместить все, не теряя в быстродействии и стабильности, сложно. Выбор операционной системы во многом зависит от запросов и уровня подготовленности пользователя, наличия компьютерной сети, назначения компьютера и его конфигурации.

ЗАДАЧИ И УПРАЖНЕНИЯ

1 Выполнить подготовку жесткого диска к установке операционной системы.

2 Выполнить установку операционной системы.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Перечислить три преимущества файловой системы NTFS по сравнению с файловой системой FAT.

2 Для чего используется дескриптор процесса?

3 В чём различия понятий мультипрограммность и мультизадачность?

ПРАКТИЧЕСКАЯ РАБОТА № 16

Тема учебной программы: Операционные системы

Тема работы: Конфигурирование операционной системы

Цель работы: корректировка загрузки операционной системы

Материально-техническое оснащение: конспект лекций; целевой компьютер; операционная система MS Windows XP

Количество часов: 4

КРАТКИЕ СВЕДЕНИЯ

Увеличить скорость загрузки операционной системы можно двумя способами – сделать апгрейд аппаратной части компьютера или принудить операционную систему загружаться быстрее.

Существует несколько программных способов ускорения загрузки и работы операционной системы:

- настройка системных служб;
- настройка автозагрузки;
- очистка и настройка системного реестра.

ЗАДАЧИ И УПРАЖНЕНИЯ

1 Выполнить ускорение загрузки операционной системы:
- отключить неиспользуемые системные службы;
- очистить список автозагрузки.

2 Настроить файл подкачки.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Какой файл MS Windows формирует меню начальной загрузки?
2 Где в MS Windows можно оптимизировать управление виртуальной памятью?
3 Как можно оптимизировать операционную систему помощью файла подкачки?

ПРАКТИЧЕСКАЯ РАБОТА № 17

Тема учебной программы: Техническое обслуживание

Тема работы: Профилактическое обслуживание компьютерной системы

Цель работы: изучить способы чистки различных компонентов компьютерной системы

Материально-техническое оснащение: конспект лекций; целевой компьютер; инструменты для разборки и чистки

Количество часов: 2

КРАТКИЕ СВЕДЕНИЯ

Профилактическое обслуживание – комплекс мероприятий, направленных на поддержание исправного технического состояния компьютерной системы в течение определённого промежутка времени. Профилактическое обслуживание основано на календарном принципе – составляется график проведения работ с указанием сроков и объёмов мероприятий.

Существует два типа профилактических мероприятий:

- активное. Выполняется чистка отдельных компонентов или всей системы, проверка на наличие вирусов, резервное копирование информации, проверка поверхности жёстких дисков, дефрагментация файлов;
- пассивное. Выполняется защита компьютера от внешних неблагоприятных воздействий, установка защитных устройств в сети электропитания, поддержание микроклимата в помещении.

ЗАДАЧИ И УПРАЖНЕНИЯ

1 Обосновать положения:

- При заземлении компьютерной системы нельзя использовать батарею центрального отопления
- Компьютерная система подключается с источнику электропитания через сетевой фильтр или блок бесперебойного питания
- В приводах нельзя оставлять носители информации
- Оборудование, принесённое с мороза, нельзя включать сразу
- Слишком высокая или слишком низкая температура, а также слишком высокая или слишком низкая влажность отрицательно сказываются на качестве и продолжительности работы компьютерной системы
- Перед выполнением любых ремонтных работ следует всегда отключать компьютерную систему от источника электропитания
- Перед работой внутри системного блока необходимо снять с себя статическое электричество

2 Выполнить профилактическое обслуживание поверхности компонентов компьютерной системы.

3 Удалить пыль из системного блока и вентиляционных отверстий блока питания.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Чем лучше очищать окисленные контакты печатной платы?
2 Для каких целей используется антистатический браслет?
3 Что означает термин EFI?
4 Какой эффект возможен, если оставить системный блок открытым?
5 Какой тип огнетушителя необходимо использовать для тушения электрооборудования?

ПРАКТИЧЕСКАЯ РАБОТА № 18

Тема учебной программы: Техническое обслуживание

Тема работы: Диагностические программы

Цель работы: выполнять диагностику компьютерной системы с использованием стандартных и специализированных программных средств

Материально-техническое оснащение: конспект лекций; программы тестирования компьютера; видеоурок «Проводим диагностику компьютера – программа Everest»

Количество часов: 2

КРАТКИЕ СВЕДЕНИЯ

Производительность компьютерной системы зависит от технических характеристик составляющих её устройств. В случае возникновения сбоев в работе компьютера или перед его модернизацией необходимо проводить тестирование различных компонентов. Тестовые программы позволяют без вмешательства в устройство компьютера определить его отдельные компоненты, а также оценить производительность и сравнить результаты тестирования с эталоном.

ЗАДАЧИ И УПРАЖНЕНИЯ

1 Посмотреть информацию об установленных драйверах, используя возможности операционной системы

2 Выполнить тестирование системной платы. Определить частоты процессора, системной шины, шины периферийных устройств.

3 Выполнить тестирование оперативной памяти.

4 Определить основные параметры системы и указать их характеристики.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Как можно посмотреть информацию от датчиков мониторинга?

2 Работоспособность какого компонента компьютера напрямую зависит от качества его охлаждения?

3 Пояснить особенности технологии S.M.A.R.T.

4 Что такое микродиагностика?

ПРАКТИЧЕСКАЯ РАБОТА № 19

Тема учебной программы: Программные неисправности

Тема работы: Сбои в операционной системе

Цель работы: ознакомление со способами восстановления операционной системы; использование и создание точек отката; создание загрузочного диска

Материально-техническое оснащение: конспект лекций; целевой компьютер; видеоуроки «Контрольные точки восстановления», «Создание образа раздела», «Создание загрузочного диска»

Количество часов: 4

КРАТКИЕ СВЕДЕНИЯ

1 Проблемы при загрузке операционной системы. Возможны варианты:

- система не может найти ни одного загрузочного диска;
- система обнаружила критическую ошибку на загрузочном диске, продолжение загрузки невозможно. При этом на экран выводится соответствующее текстовое сообщение;
- компьютер зависает на определённом этапе загрузки или загрузка прекращается с выводом на экран сообщения об ошибке (или без него).

2 Проблемы при работе операционной системы. Возможны варианты:

- загружается слишком медленно;
- загружается, но сразу появляется сообщение о критической ошибке или компьютер зависает;
- загружается нормально, но при запуске разных программ выводится сообщение об ошибке или компьютер зависает;
- загружается нормально, но не работают некоторые программы и функции, оборудование.

ЗАДАЧИ И УПРАЖНЕНИЯ

1 Указать результаты нажатия клавиш при загрузке операционной системы:

□	
□	
□	
□	
□	

2 Создать точку отката на сегодняшнее число.

3 Загрузить систему в безопасном режиме. Определить, драйвера каких устройств работают в этом режиме.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Какие три фактора следует принимать во внимание при обновлении операционной системы?

2 Для чего предназначен безопасный режим запуска операционной системы?

3 В каких случаях следует активизировать запуск в пошаговом режиме с подтверждением?

4 Какие функции BIOS нужно отключить при проведении обновления операционной системы?

5 Как настроить автоматическое обновление для операционной системы?

ПРАКТИЧЕСКАЯ РАБОТА № 20

Тема учебной программы: Программные неисправности

Тема работы: Сбои в программном обеспечении

Цель работы: определять причину появления сбоя в работе программы; устранять неисправности в работе прикладных программ

Материально-техническое оснащение: конспект лекций; целевой компьютер; программное обеспечение

Количество часов: 2

КРАТКИЕ СВЕДЕНИЯ

1 Надёжность программного обеспечения – свойство сохранять заданные характеристики при определённых условиях эксплуатации. При анализе надёжности выполнения компьютерной системой заданных функций компьютер следует рассматривать как единый комплекс программных и аппаратных средств и учитывать, что его надёжность зависит также от надёжности программ. Надёжность ПО определяется его безотказностью и восстанавливаемостью.

Безотказность ПО – его свойство сохранять работоспособность в процессе обработки информации, которую можно определить вероятностью работы без отказов при определённых условиях внешней среды в течение заданного периода наблюдения.

Отказ программы – недопустимое отклонение характеристик процесса функционирования программы от требуемых. Определённые условия внешней среды – совокупность входных данных и состояния компьютерной системы.

Заданный период наблюдения обычно соответствует необходимому числу прогонов программы для решения задачи.

Безотказность ПО можно охарактеризовать также средним временем между двумя отказами в процессе выполнения программы `T` (при условии, что сбой аппаратных средств отсутствует). С точки зрения надёжности принципиальное отличие программных средств от аппаратных состоит в том, что программы не изнашиваются и не подвержены физическому старению в процессе работы.

Поэтому характеристики надёжности ПО зависят от тщательности разработки и отладки, а также от условий хранения носителей программ. Безотказность ПО определяется его корректностью, а значит целиком зависит от наличия в нём ошибок, внесённых на этапе создания и хранения, в то время как безотказность аппаратных средств зависит в основном от случайных отказов, связанных с физическими изменениями параметров элементов. Вид обрабатываемых данных не влияет на аппаратуру, но может привести к отказам ПО.

Интенсивность отказов ПО с течением времени уменьшается, так как в процессе эксплуатации обнаруживаются и устраняются его скрытые ошибки.

Восстанавливаемость программы может быть оценена сравнительной продолжительностью устранения ошибки в программе и восстановления ее работоспособности. Восстановление после отказа может заключаться в корректировке текста программы, исправлений данных, внесения изменений в организацию вычислительного процесса. Восстанавливаемость зависит от сложности структуры комплекса программ, от алгоритмического языка, от качества документации и т.д. Можно также говорить об устойчивости ПО, понимая под этим способность ограничивать последствия собственных ошибок и противостоять неблагоприятным условиям внешней среды. Устойчивость ПО может быть повышена с помощью разных форм структурной, информационной и временной избыточности, позволяющей иметь дублирующие модули программ, альтернативные

пути для решения одной задачи, позволяющих осуществлять контроль за процессом исполнения программ (зацикливание, блокировка и т.д.).

Причины отказов ПО:

- ошибки, скрытые в самой программе;
- искажение входной информации, подлежащей обработке;
- неверные действия пользователя (могут быть связаны с некорректной документацией);
- неисправность аппаратуры, на которой реализуется вычислительный процесс.

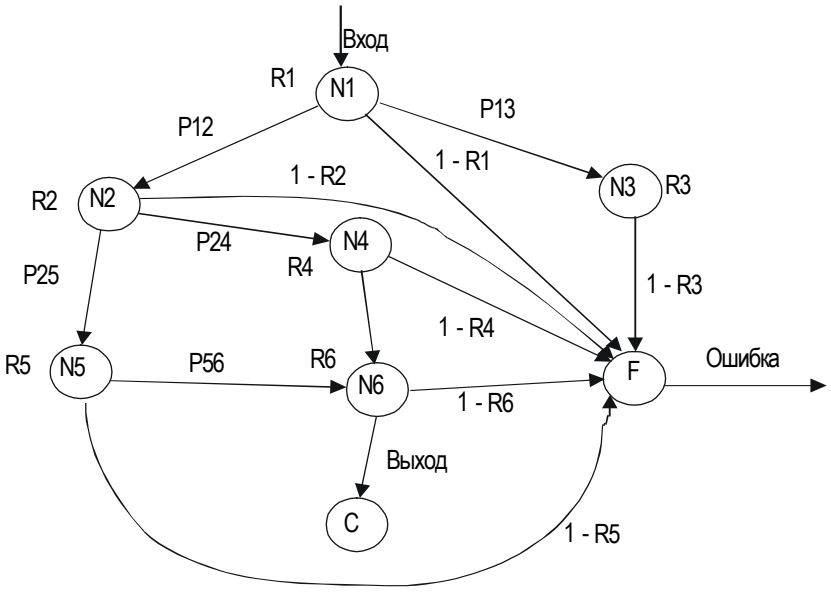
В зависимости от степени серьёзности последствий ошибок в программе, отклонения выполнения программой заданных функций можно разделить следующим образом: полное прекращение выполнения функций на длительное или неопределённое время или кратковременное прекращение хода вычислительного процесса.

Симптомы проявления ошибки в программе:

- преждевременное окончание программы;
- увеличение времени выполнения программы (зацикливание);
- потери или искажение накопленных данных;
- нарушение порядка вызова отдельных программ.

Для устранения ошибок программы необходимо предусмотреть специальные средства диагностики типа кодов завершения, вводить в ПО контрольные точки, обеспечить возможность рестарта с контрольных точек.

Для определения надёжности больших программных комплексов используют марковские модели. В марковском процессе выбор следующего модуля зависит только от модуля, выполняемого в данный момент и не зависит от предыстории. Структуру управления программой по марковской модели можно представить в виде направленного графа.



Каждое состояние программы может быть оценено вероятностью безотказной работы i-го модуля R_i. Вероятность перехода от i-го модуля к j-му показана величиной P_{ij}.

Вероятность отказа равна $1 - P_{ij}$. Таким образом можно составить матрицу переходных вероятностей:

$$P = \begin{Vmatrix} 0 & R_1P_{12} & R_1P_{13} & \dots & R_{ij} & 0 & 1-R_1 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & R_iP_{ij} & \dots & .. & 0 & 1-R_{n-1} \\ 0 & \dots & R_{n-1}P_{(n-1)j} & \dots & \dots & R_n & 1-R_n \\ 0 & \dots & \dots & \dots & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{Vmatrix}$$

Для каждого целого k>0 Pk(i, j) будет определять вероятность перехода из состояния i в состояние j за k шагов. Тогда матрица $T = I + P + P^2 + \ldots = \sum_{k=0}^{\infty} P^k$ будет определять вероятность перехода и одного состояния в другое за произвольное число шагов.

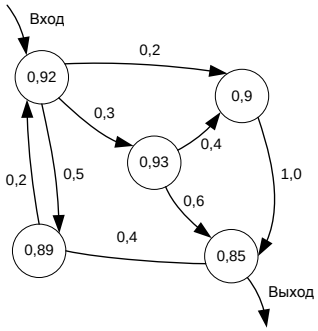
Вычеркнув из матрицы P две последние строки и два последних столбца, соответствующие успеху C и отказу F, получаем матрицу Q.

Пусть $S = I + Q + Q^2 + \ldots = \sum_{k=0}^{\infty} Q^k$. Положив W = I – Q, имеем: S = W-1 = (I – Q)-1, откуда надёжность программного комплекса: R = S (1,n) Rn.

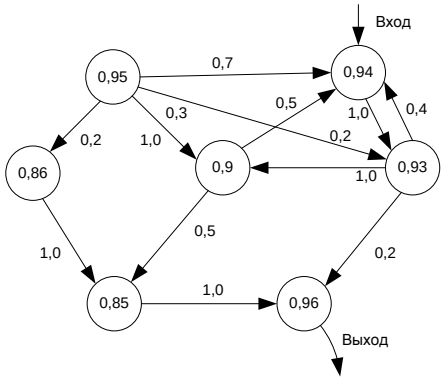
ЗАДАЧИ И УПРАЖНЕНИЯ

Определить надёжность комплекса программ, отображаемых графом:

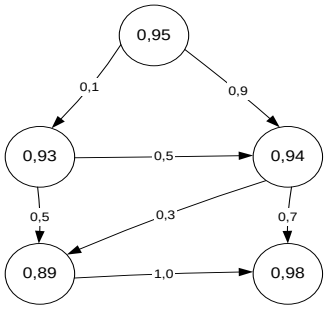
а)



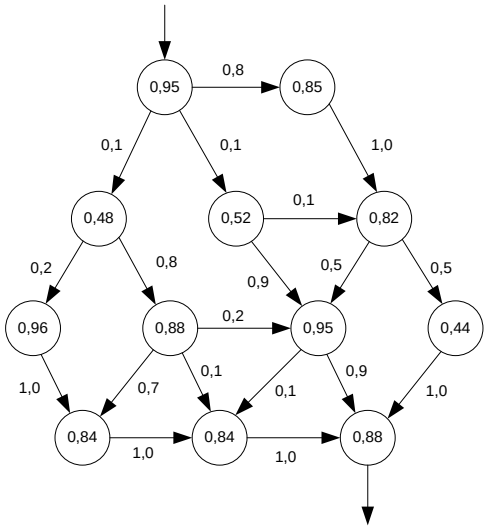
б)



в)



г)



КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Перечислить основные характеристики надёжности программного обеспечения.
2 В чём особенность динамических библиотек?
3 Указать основные классы скрытых ошибок программного обеспечения.
4 Как запустить Диспетчер задач с помощью клавиатуры?

ПРАКТИЧЕСКАЯ РАБОТА № 21

Тема учебной программы: Программные неисправности

Тема работы: Сбои системного реестра

Цель работы: выполнять редактирование и очистку содержимого системного реестра; производить восстановление файлов системного реестра

Материально-техническое оснащение: конспект лекций; целевой компьютер; программа Clean-Clear

Количество часов: 2

КРАТКИЕ СВЕДЕНИЯ

Системный реестр – основа операционной системы. Он содержит всю нужную для работы компьютера информацию: данные для первого запуска программ, параметры программ, константы и переменные и т.п.

При старте операционной системы реестр загружается в оперативную память, поэтому важным параметром является размер его файлов. Со временем объём хранящихся в реестре данных увеличивается, что требует дополнительного выделения памяти. Обычно это происходит из-за накапливания ненужной и давно забытой информации, остающейся после когда-то установленных (и уже удалённых) программ.

Бороться с захламлением реестра можно двумя способами. Первый заключается в переустановке операционной системы, что не всегда является лучшим вариантом, учитывая количество установленных программ. Второй способ подразумевает использование специальных программ очистки реестра. Такие программы анализируют ключи реестра, определяя, к каким из них система обращалась давно и какие являются ошибочными.

Существует много различных программ очистки реестра.

1 Структура реестра

- Раздел HKEY_CLASSES_ROOT. В этом разделе содержатся ключи двух главных типов. Первый тип ключей хранит информацию о расширении файла. Таким способом хранятся все трёхбуквенные расширения имён файлов (MP3, JPG), которые когда-либо использовались. Реестр также использует расширения для связи типа файла с определённым действием. Второй тип ключей представляет собой собственно ассоциацию. Расширение имени файла обычно указывает на файл данных определённого приложения. В разделе, описывающем ассоциацию, находятся ключи используемые в контекстном меню Проводник. Ассоциация также содержит информацию о том, какой значок выводить для данного типа файла.
- Раздел HKEY_CURRENT_USER содержит множество настроек программного обеспечения, в которых хранится информация о конфигурации рабочего стола и клавиатуры. Кроме того, в этом разделе есть информация о параметрах меню Пуск. Здесь – все настройки, специфичные для пользователя. Раздел HKEY_CURRENT_USER полностью посвящён настройкам текущего пользователя (пользователя, который в настоящий момент зарегистрирован в системе). Это отличается от пользовательской конфигурационной информации, которая хранится в других разделах реестра. Информация в этом разделе является динамической; информация в других разделах, которая относится к пользовательским настройкам, – статическая. Реестр копирует содержимое одного из подразделов из HKEY_USERS в данный раздел и по окончании работы обновляет раздел HKEY_USERS.

В HKEY_CURRENT_USER операционная система получает новую информацию о настройках системы, а также сюда помещаются любые изменения. Всё, что здесь хранится, служит только для настройки системы для нужд пользователя, но никогда не содержит системной информации.

· Раздел HKEY_LOCAL_MACHINE. В этом разделе содержатся основные сведения об аппаратных средствах компьютера, включая драйверы устройств и конфигурационную информацию. Если информации о каком-либо устройстве здесь нет, то операционная система не сможет использовать его.

Раздел HKEY_LOCAL_MACHINE – для очень подробной информации об аппаратном обеспечении (содержит всю информацию, необходимую для PnP; предоставляет полный список драйверов устройств и их уровни ревизии; содержит информацию о ревизии самого аппаратного обеспечения).

Кроме того, в этом разделе имеется некоторая информация о программном обеспечении (32-разрядное приложение будет здесь хранить таблицу установок и форматов). Эта информация используется приложением во время установки. Некоторые приложения используют её во время изменений параметров их установки. В этом разделе содержится только информация глобального характера.

· Раздел HKEY_USERS содержит список всех пользователей данного файла реестра. Необходимости в изменении информации этого раздела никогда не возникает, но его можно использовать в справочных целях. Причина, по которой нужно следовать этим указаниям, проста: ни одно изменение не войдет в силу до тех пор, пока пользователь не зарегистрируется в системе следующий раз. Кроме того, изменение настроек для текущего пользователя является пустой тратой времени, потому что операционная система заменяет все данные в соответствующем разделе данными, которые содержатся в разделе HKEY_CURRENT_USER, во время завершения сеанса или при завершении работы системы.

Существует еще одна проблема, связанная с использованием этого раздела в качестве единственного источника информации. На самом деле Windows поддерживает несколько копий реестра в многопользовательской среде, в некоторых случаях отдельную копию для каждого пользователя. По этой причине никогда нельзя быть уверенным, где именно можно найти информацию об определённом пользователе. Windows отслеживает эту информацию; необходимость поиска такой информации это головная боль для администратора.

Когда пользователь регистрируется в системе, Windows копирует информацию из его профиля в раздел реестра HKEY_CURRENT_USER. Когда пользователь выходит из системы или завершает работу, Windows заменяет информацию в разделе, соответствующему данному пользователю, информацией из раздела HKEY_CURRENT_USER.

· Раздел HKEY_CURRENT_CONFIG представляет собой самую простую часть реестра. Он содержит два главных раздела: Display и System. По существу эти разделы используются программным интерфейсом GDI API для конфигурации монитора и принтера.

Раздел Display имеет два подраздела: Fonts и Setting. Fonts определяет шрифты, которые Windows 98 использует для вывода на экран. Setting содержит текущее разрешение экрана и количество битов на пиксель. Количество битов на пиксель определяет доступное количество цветов (4 бита на пиксель – 16 цветов, 8 битов на пиксель – 256 цветов). Три шрифта, перечисленных в этом разделе, являются шрифтами по умолчанию, которые система использует для отображения значков и меню приложений. Эти настройки можно изменить в диалоговом окне свойств экрана.

Раздел System производит впечатление чего-то запутанного. Однако только один подраздел этого раздела имеет смысл для пользователя – Printers, который содержит список принтеров, подсоединённых к системе. Этот список не включает принтеры, которые используются по сети.

· Раздел HKEY_DYN_DATA содержит два подраздела: Config Manager и PerfStats. Просмотреть статус ключа Config Manager можно при помощи вкладки Устройства диалогового окна свойств системы. Значения ключей из раздела PerfStats отображаются в интерфейсе утилиты System Monitor.

2 Программа RegCleaner.

Главное окно программы содержит семь вкладок, каждая из которых отображает разную информацию: списки программ для деинсталляции, автозагрузки, зарегистрированных и добавленных расширений и т.д. Дополнительно можно работать с системными библиотеками, установленными устройства, реестром и т.д.

Программа позволяет очищать реестр в двух режимах: ручном и автоматическом. При первом можно самостоятельно выбирать, что нужно удалить, а что оставить. В автоматическом режиме программа сама анализирует записи в реестре и выводит все подозрительные данные на экран.

Все механизмы работы с реестром и другими данными доступны из меню Задачи.

ЗАДАЧИ И УПРАЖНЕНИЯ

1 Выполнить сохранение файлов реестра.

2 Используя программу CleanClear:
- просмотреть список программ, установленных в системе;
- выполнить очистку от ссылок на несуществующие файлы.

3 Выполнить восстановление файлов реестра.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 В каком режиме восстановления нельзя восстановить повреждённый системный реестр?
2 Можно ли редактировать файлы системного реестра по отдельности?

ПРАКТИЧЕСКАЯ РАБОТА № 22

Тема учебной программы: Аппаратные неисправности

Тема работы: Сигналы POST

Цель работы: определять неисправность компьютерной системы по сигналам POST

Материально-техническое оснащение: конспект лекций; целевой компьютер

Количество часов: 2

КРАТКИЕ СВЕДЕНИЯ

При каждом включении компьютера автоматически запускается подпрограмма BIOS – POST (Power On Self Test) – диагностическая программа самотестирования. POST проверяет важнейшие компоненты компьютера: работоспособность процессора, исправность чипсета, оперативной памяти (RAM), видеокарты и т.д.

На экране монитора последовательно появляются сообщения о нормальной загрузке.

В конце процесса тестирования выдаётся сводная таблица, содержащая основную информацию о компьютере, прочитать которую можно после нажатия клавиши ¯. После окончания процесса тестирования выводится стартовая надпись или же графическая заставка операционной системы.

POST является первым постом, позволяющим определить неисправность компьютера. Более того, порой это единственный способ определения неполадок, особенно если нет дополнительного оборудования.

POST может выдавать сообщения следующими способами:

- звуковой сигнал. Каждая конкретная неисправность имеет свою серию звуковых сигналов, которые выводятся в ходе тестирования устройств. Этот способ является основным;
- текстовое сообщение. На экране появляется сообщение, кратко описывающее неисправность и код ошибки, по которой неполадку можно изучить подробно, воспользовавшись документацией к системной плате или к BIOS. Этот способ используется как дополнение к звуковым сигналам при исправности видеосистемы компьютера. Главный плюс такого способа – неисправность компьютера незначительна;
- шестнадцатеричные коды в конкретный порт по определённому адресу. Данные способ используется независимо от того, выдаются ли звуковые или текстовые сообщения. Для чтения этих кодов, необходимо иметь POST-карту.

ЗАДАЧИ И УПРАЖНЕНИЯ

1 Определить причину возникновения сбоя по время начальной загрузки компьютера:

Звуковой сигнал	Описание неисправности	Рекомендации по устранению неисправности
Сигналов нет		
Непрерывный		
Постоянные короткие		
Один длинный и пустой экран		
Постоянные короткие		

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 О чём свидетельствуют неполадки во время самотестирования системы?

2 Какая последовательность звуковых сигналов свидетельствует о корректном завершении процедуры POST?

ПРАКТИЧЕСКАЯ РАБОТА № 23

Тема учебной программы: Аппаратные неисправности

Тема работы: Поиск и устранение неисправностей компонентов системного блока

Цель работы: определять характер возникновения неполадки в работе компьютера и степень её влияния; устранять неисправность в работе компонентов системного блока

Материально-техническое оснащение: конспект лекций; целевой компьютер; мультиметр

Количество часов: 2

КРАТКИЕ СВЕДЕНИЯ

Первое, что необходимо сделать в случае поломки компьютера, – определить характер и причину неисправности. При этом выделяют категории:

- влияние факторов внешнего окружения: воздействие прямых солнечных лучей; в помещении происходят резкие перепады температуры или повышенная влажность; в помещении работают устройства с циклической нагрузкой;
- компьютер не включается: обнулить CMOS-память; очистить от пыли блок питания; заменить предохранители и т.п.;
- проблемы при прохождении программы диагностики POST: отсутствуют звуковые сигналы и текстовые сообщения; звучит сигнал или выдаётся сообщение.

ЗАДАЧИ И УПРАЖНЕНИЯ

1 Используя мультиметр проверить:
- напряжение в блоке питания;
- целостность соединительных кабелей;
- напряжение батарейки.

2 Открыв системный блок, визуально определить неисправные компоненты.

3 Включив компьютер, определить возможную причину неисправности. Устранить неисправность и проверить работоспособность системы.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Что необходимо проверить в первую очередь, если система не включается?

2 Какие параметры, максимально зависящие от аппаратного обеспечения, проверяют в первую очередь?

3 Что предварительно нужно сделать для проверки выпрямителя блока питания, расположенного на диодной сборке?

ПРАКТИЧЕСКАЯ РАБОТА № 24

Тема учебной программы: Аппаратные неисправности

Тема работы: Поиск и устранение неисправностей устройств хранения данных

Цель работы: определять характер неисправности в работе накопителей на жёстких и оптических дисках; устранять неисправность в работе накопителей

Материально-техническое оснащение: конспект лекций; целевой компьютер; приводы CD/DVD; видеоурок «Дефрагментация диска»; видеоурок «Тестируем локальный диск на наличие ошибок – программа HDD Life»

Количество часов: 2

КРАТКИЕ СВЕДЕНИЯ

Устройства хранения оказывают значительное влияние на эффективность функционирования компьютерной системы в целом. Накопители на жёстких и оптических дисках обеспечивают постоянное и удобное хранение программного обеспечения и данных пользователя и высокую скорость доступа к ним.

Любое устройство хранения состоит из компонентов:

- накопитель;
- носитель;
- контроллер;
- обслуживающие программы – драйверы.

ЗАДАЧИ И УПРАЖНЕНИЯ

1 Ознакомиться ремонтом HDD методом перекомплектациии.

2 Ознакомиться с методом программного «ремонта» HDD.

3 Выполнить дефрагментацию диска.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Какие неисправности возникают при длительном перегреве жёсткого диска?
2 Как можно извлечь оптический диск из привода без подачи электропитания?

ПРАКТИЧЕСКАЯ РАБОТА № 25

Тема учебной программы: Аппаратные неисправности

Тема работы: Поиск и устранение неисправностей устройств отображения данных

Цель работы: определять характер неисправности в работе мониторов и проекционного аппарата; устранять неисправность в работе устройств отображения данных

Материально-техническое оснащение: конспект лекций; CRT- и LCD-монитор; цифровой проектор

Количество часов: 2

КРАТКИЕ СВЕДЕНИЯ

В настоящее время используют два основных типа мониторов: с электронно-лучевой трубкой и жидкокристаллический.

Монитор выходит из строя достаточно редко. Неисправности в мониторе возникают, как и в других изделиях электронной техники по следующим причинам:

- некачественное изготовление;
- нарушение правил эксплуатации;
- естественное старение электронных компонентов;
- ремонт неквалифицированным персоналом

ЗАДАЧИ И УПРАЖНЕНИЯ

1 Монитор подключён к видеокарте, установленной в порт AGP, но система изображение не выводит. Как можно устранить подобную неисправность?

2 Ознакомиться с критериями исправной работы тракта обработки видеосигнала.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Почему монитор может ограничивать количество выводимых цветов?
2 Какие меры безопасности необходимо соблюдать при ремонте и диагностике монитора?
3 Какие виды сигналов подаются на вход монитора и какова их характеристика?

ПРАКТИЧЕСКАЯ РАБОТА № 26

Тема учебной программы: Аппаратные неисправности

Тема работы: Поиск и устранение неисправностей устройств интерактивного взаимодействия

Цель работы: определять характер неисправности в работе клавиатуры, манипуляторных устройств, сканера; устранять неисправность в работе накопителей

Материально-техническое оснащение: конспект лекций; целевой компьютер; клавиатура; манипулятор «мышь»; измерительный переходник

Количество часов: 2

КРАТКИЕ СВЕДЕНИЯ

Компьютерная клавиатура – устройство неприхотливое, однако и у неё бывают разные неисправности: «залипание» клавиш, неисправность разъёма и т.п., таким образом, клавиатура является одним из факторов риска.

Манипулятор используется для управления курсором на экране компьютера. К числу основных отказов манипулятора мышь относятся: загрязнение, поломка кнопок, излом проводов кабеля. Перед проведением любых ремонтных работ мышь необходимо отключить от компьютера, иначе может «сгореть» порт, через который она подключена к компьютеру.

Сканер – высокоточное устройство, предназначенное для качественного сканирования любых изображений. Устранять неисправности следует осторожно, так как попытки устранить некоторые неисправности могут привести к полной разбалансировке оптики сканера, при этом восстановить её становится невозможно.

ЗАДАЧИ И УПРАЖНЕНИЯ

1 При перемещении курсора мыши, он двигается беспорядочно. Что можно сделать для исправления неполадки?

2 Объяснить принцип действия оптической «мыши».

3 Линейка сканера двигается, но когда приложение запрашивает просмотр, появляется сообщение об ошибке. В чём проблема?

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Какова роль контроллера i8048 в управлении клавиатурой?
2 В чём заключается назначение микросхемы LS138 в клавиатуре?
3 Какие порты служат для обращения к контроллеру клавиатуры?
4 Каким образом строится взаимодействия манипулятора «мышь» с системной платой?
5 Какие порты служат для обращения к контроллеру манипулятора?

ПРАКТИЧЕСКАЯ РАБОТА № 27

Тема учебной программы: Аппаратные неисправности

Тема работы: Поиск и устранение неисправностей устройств вывода на печать

Цель работы: определять характер неисправности в работе принтера; устранять неисправность в работе печатающего устройства

Материально-техническое оснащение: конспект лекций; целевой компьютер; клавиатура; манипулятор «мышь»; измерительный переходник

Материально-техническое оснащение: конспект лекций; целевой компьютер; принтеры разных технологий печати; инструмент для сборки и разборки; 3-5% раствор аммиака

Количество часов: 2

КРАТКИЕ СВЕДЕНИЯ

Принтеры выходят из строя часто. Неисправности могут быть как незначительными (засорение печатающей головки), так и весьма серьёзными (выход из строя фотобарабана). Впрочем, многих проблем можно легко избежать, если постоянно следить за состоянием принтера.

Ошибки в процессе печати могут возникать как из-за аппаратных, так и из-за программных неисправностей. В каждом конкретном случае нужно тщательно изучить ситуацию.

Очень часть пользователь устанавливает драйверы принтера, похожего на свой или совместимого с ним. Это также может вызывать различные программные ошибки.

ЗАДАЧИ И УПРАЖНЕНИЯ

1 Провести заправку картриджа лазерного принтера, выполнив предварительную чистку. Установить в принтер и проверить работоспособность.

2 Определить неисправность лазерного принтера. Устранить сбой в работе. Подключить к компьютеру и проверить работоспособность.

3 Определить неисправность струйного принтера. Устранить сбой в работе. Подключить к компьютеру и проверить работоспособность.

4 Описать решение ситуации:

- за месяц принтер ломается четвёртый раз, изнашивается механизм подачи бумаги;
- при печати лист не распечатывается до конца;
- на листе, отпечатанном на лазерном принтере, в некоторых местах тонер осыпается;
- при печати на лазерном принтере, изображение постепенно становится бледным. Картридж был недавно заменён;
- при печати струйный принтер одну строку печатает нормально, а следующую – белым (пустая).

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Назвать три проблемы, являющиеся общими для матричных, лазерных и струйных принтерах?

2 Где нужно проверять неисправность, если принтер производит ошибку подачи бумаги при попытке напечатать?

3 Что необходимо проверить в первую очередь, если лазерный принтер выдаёт пустые страницы?

4 Какие неполадки возникают при использовании некачественного тонера в лазерном принтере?

5 Почему нельзя использовать растворитель для прочистки сопел в струйном принтере?

6 Как можно увеличить скорость вывода на печать?

ПРАКТИЧЕСКАЯ РАБОТА № 28

Тема учебной программы: Обслуживание и ремонт портативных компьютеров

Тема работы: Внутренние компоненты портативного компьютера

Цель работы: определять конфигурацию портативного компьютера

Материально-техническое оснащение: конспект лекций; портативный компьютер

Количество часов: 2

КРАТКИЕ СВЕДЕНИЯ

Конфигурация компьютерной системы определяется визуально или с помощью средств операционной системы или специализированных программ.

Для обоих методов определения конфигурации определяют элементы:

- тип процессора;
- тактовая частота процессора;
- тип BIOS;
- количество подключённых накопителей;
- размер ОЗУ и кэш-памяти;
- параметры контроллера клавиатуры;
- наличие дополнительных контроллеров и адаптеров;
- тип системной и локальной шины и их характеристики.

ЗАДАЧИ И УПРАЖНЕНИЯ

1 Определить конфигурацию портативного компьютера визуально. Составить формуляр компьютерной системы.

2 Определить конфигурацию портативного компьютера с помощью средств операционной системы. Составить формуляр компьютерной системы.

3 Выполнить сравнение точности визуального и программного методов определения конфигурации портативного компьютера.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Какие компоненты влияют на дизайн портативных компьютеров?
2 Почему для портативного компьютера технологии теплоотвода являются актуальными?
3 Какие модули оперативной памяти устанавливают в портативный компьютер?
4 В чём заключаются сложности модернизации портативного компьютера?

ПРАКТИЧЕСКАЯ РАБОТА № 29

Тема учебной программы: Обслуживание и ремонт портативных компьютеров

Тема работы: Подключение внешних устройств к портативному компьютеру

Цель работы: проводить инсталляцию внешних устройств с портативными компьютерами

Материально-техническое оснащение: конспект лекций; портативный компьютер, принтер; сканер

Количество часов: 2

ЗАДАЧИ И УПРАЖНЕНИЯ

1 Описать типы разъёмов, установленные на портативном компьютере.

2 Подключить сканер и установить соответствующий драйвер устройства.

3 Выполнить сканирование документа.

4 Подключить принтер и установить соответствующий драйвер устройства.

5Произвести печать тестовой страницы.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Какие преимущества обеспечивает док-станции при подключении внешних устройств?

2 В чём особенность реализации проводных и беспроводных сетей в портативных компьютерах?

ПРАКТИЧЕСКАЯ РАБОТА № 30

Тема учебной программы: Обслуживание и ремонт портативных компьютеров

Тема работы: Профилактическое обслуживание портативного компьютера

Цель работы: изучить способы профилактического обслуживания портативного компьютера

Материально-техническое оснащение: конспект лекций; кабели разных категорий

Количество часов: 2

ЗАДАЧИ И УПРАЖНЕНИЯ

1 Определить условия пассивного профилактического обслуживания портативного компьютера.

2 Удалить пыль из корпуса и вентиляционных отверстий портативного компьютера.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Какие преимущества обеспечивает использование док-станции при профилактическом обслуживании?

2 Какой тип крепёжный деталей используется в портативном компьютере?

3 Перечислить правила эксплуатации портативного компьютера.

ПРАКТИЧЕСКАЯ РАБОТА № 31

Тема учебной программы: Обслуживание и ремонт портативных компьютеров

Тема работы: Диагностирование портативного компьютера

Цель работы: проведение диагностики портативного компьютера с использованием стандартных и специализированных программных средств

Материально-техническое оснащение: конспект лекций; программы тестирования

Количество часов: 2

ЗАДАЧИ И УПРАЖНЕНИЯ

1 Посмотреть информацию об установленных драйверах, используя возможности операционной системы

2 Выполнить тестирование системной платы. Определить частоты процессора, системной шины, шины периферийных устройств.

3 Выполнить тестирование оперативной памяти.

4 Определить основные параметры системы и указать их характеристики.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Пояснить особенность использования встроенных программ диагностики.
2 Какие параметры нельзя определить в процессе диагностирования?

ПРАКТИЧЕСКАЯ РАБОТА № 32

Тема учебной программы: Обслуживание и ремонт портативных компьютеров

Тема работы: Поиск и устранение неисправностей в работе портативного компьютера

Цель работы: определять неисправность в работе портативного компьютера с помощью аппаратных и программных средств

Материально-техническое оснащение: конспект лекций; кабели разных категорий

Количество часов: 6

ЗАДАЧИ И УПРАЖНЕНИЯ

1 Используя мультиметр проверить:
- напряжение в блоке питания;
- целостность соединительных кабелей;
- напряжение батарейки.

2 Открыв системный блок, визуально определить неисправные компоненты.

3 Включив компьютер, определить возможную причину неисправности. Устранить неисправность и проверить работоспособность системы.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Что проверяют в первую очередь, если компьютер не включается?
2 Пояснить особенности использования POST-карты.
3 Как определить неисправность во время самотестирования портативного компьютера?

ПРАКТИЧЕСКАЯ РАБОТА № 33

Тема учебной программы: Обслуживание компьютерных сетей

Тема работы: Профилактическое обслуживание сетевого оборудования

Цель работы: проводить профилактику сетевого оборудования

Материально-техническое оснащение: конспект лекций; локальная сеть; приборы для тестирования оборудования

Количество часов: 2

КРАТКИЕ СВЕДЕНИЯ

Целью выполнения любого профилактического мероприятия является продление срока безотказной работы компьютера и сетевого оборудования. Большинство мероприятий сводятся, главным образом, к периодической чистке как всей системы, так и отдельных её компонентов.

ЗАДАЧИ И УПРАЖНЕНИЯ

1 Проверить корректность подключения разъёмов электропитания.

1 Выполнить перестыковку разъёмов.

2 Произвести очистку активного сетевого оборудования.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Какие мероприятия обеспечивают защиту компьютерной сети от внешних неблагоприятных воздействий?

2 От чего зависит периодичность профилактического обслуживания?

ПРАКТИЧЕСКАЯ РАБОТА № 34

Тема учебной программы: Обслуживание компьютерных сетей

Тема работы: Контроль за работоспособностью компьютерной сети

Цель работы: определять сетевые протоколы

Материально-техническое оснащение: конспект лекций; кабели разных категорий

Количество часов: 2

КРАТКИЕ СВЕДЕНИЯ

Для мониторинга за работой в сетях используется пункт Сетевое подключение.

ЗАДАЧИ И УПРАЖНЕНИЯ

1 Используя свойства протокола Internet заполнить таблицы

IP адрес	
Маска подсети	
Основной шлюз	
Предпочитаемый DNS-сервер	
Альтернативный DNS-сервер	

Сетевая плата	
Используемые протоколы	
IP-адрес	
Маска подсети	
Доменное имя компьютера	
DNS-сервер(ы)	
Шлюз	

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Для чего используется маска подсети?
2 Пояснить назначение и принцип работы сервиса ARP.

ПРАКТИЧЕСКАЯ РАБОТА № 35

Тема учебной программы: Обслуживание компьютерных сетей

Тема работы: Диагностика компьютерной сети

Цель работы: проводить диагностику работоспособности компьютерной сети

Материально-техническое оснащение: конспект лекций; кабели разных категорий

Количество часов: 2

КРАТКИЕ СВЕДЕНИЯ

Для диагностики сети можно использовать штатные средства операционной системы – командную строку. Команды:

- `ipconfig` – проверяет состояние оборудования и наличие подключения кабеля;
- `ping` – проверяет сетевое оборудование;

Также общую информацию можно получить, если воспользоваться трассировкой – команда `tracert`.

Для диагностики сети обязательно проводят тестирование отдельных протоколов – программа `Telnet`.

ЗАДАЧИ И УПРАЖНЕНИЯ

1 Проверить состояние оборудования. Заполнить таблицу

Сетевой адаптер	
DHCP	
IP адрес	
Маска подсети	
Тип сети	
Основной шлюз	
DNS-серверы	

2 Проверить доступность своего оборудования и оборудования провайдера в последовательности:

- свой компьютер;
- шлюз Internet;
- свой IP у провайдера;
- DNS-сервер;
- IP любого рабочего хоста в сети;
- URL любого сайта.

3 Выполнить трассировку сайта yandex.ru.

4 Выполнить тестирование сетевых портов.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Какова суть тестирования с трассировкой?
2 Для чего проводят тестирование отдельных протоколов?

ПРАКТИЧЕСКАЯ РАБОТА № 36

Тема учебной программы: Обслуживание компьютерных сетей

Тема работы: Поиск и устранение неисправностей в локальной сети

Цель работы: определять неисправность в сети с помощью аппаратных и программных средств

Материально-техническое оснащение: конспект лекций; кабели разных категорий

Количество часов: 4

КРАТКИЕ СВЕДЕНИЯ

1 Поиск неисправностей в сети аппаратными средствами

Оборудование для диагностики и поиска неисправностей делят на основные группы:

- приборы для сертификации кабельных систем – определяют затухание, отношения сигнал-шум, импеданса, ёмкости и активного сопротивления;
- сетевые анализаторы – измеряют затухание в линии и её характеристики;
- кабельные сканеры – определяют длину кабеля, затухание, импеданс, схему разводки, уровень электрических шумов;
- тестеры (мультиметры) – определяют непрерывность кабеля.

Внимание! Тестер позволяет определить повреждение кабеля, но не обозначает, где произошёл сбой.

2 Поиск неисправностей в сети утилитами TCP/IP

Запуск утилит осуществляется из Командной строки меню Пуск.

- Ping – проверяет возможность соединения с удалённым компьютером: `Ping cn.dn.fio.ru`
- Pathping – отражает маршрут прохождения и предоставляет статистику потери пакетов на промежуточных маршрутизаторах: `Pathping sn.dn.fio.ru`

ЗАДАЧИ И УПРАЖНЕНИЯ

1 Последовательно подключая тестовые сетевые кабели к сетевой карте компьютера выполнить прозвонку кабеля, используя омметр. Результаты измерения занести в таблицу

Номер тестового кабеля	Результат измерения пар кабеля, Ом	Вывод о состоянии кабеля
1		
2		
3		

2 Используя сетевой тестер определить вариант разводки кабеля и проверить правильность. Результаты измерения занести в таблицу

Номер тестового кабеля	Тип разводки
1	
2	

3 Провести поиск места неисправности в кабеле сети с использованием сетевого радара.

4 Используя утилиту Ping проверить соединение рабочих станций с сервером.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 В каких случаях использование сетевого радара является предпочтительным?
2 Заполнить таблицу

Неисправность	Возможная причина	Последствия для сети	Решение проблемы	Меры предотвращения
Выход из строя коммутатора				
Выход из строя модема (точки доступа)				
Выход из строя сервера				
Выход из строя рабочей станции				
Выход из строя сетевого кабеля				

ПРАКТИЧЕСКАЯ РАБОТА № 37

Тема учебной программы: Обслуживание компьютерных сетей

Тема работы: Обслуживание серверов и рабочих станций

Цель работы: определять конфигурацию TCP/IP

Материально-техническое оснащение: конспект лекций; кабели разных категорий

Количество часов: 6

КРАТКИЕ СВЕДЕНИЯ

Утилиты TCP/IP

- Route – показывает и позволяет изменять конфигурацию локальной таблицу маршрутизации: `Route print`
- Tracert – отслеживает маршрут, по которому пакеты перемещаются к пункту назначения: `tracert sn.dn.fio.ru`
- Netstat - показывает текущую информацию сетевого соединения TCP/IP. Например, информацию о подключенном хосте и номера используемых портов. netstat Netstat -a Netstat -r Netstat -e Netstat -s
- Ipconfig - показывает текущую конфигурацию TCP/IP на локальном компьютере. Ключи утилиты:

/release – освобождает полученный от DHCP IP-адрес;
/renew – получает от DHCP новый IP-адрес;
/all – показывает всю информацию о TCP/IP конфигурации;
/flushdns – очищает кэш локального распознавателя DNS;
/regsiterdns – обновляет адрес в DHCP и перерегистрирует его в DNS;
/displaydns – показывает содержание кэша распознавателя DNS.

- Hostname – показывает локально настроенное имя узла TCP/IP;
- Arp – показывает и позволяет изменять кэш протокола ARP (Address Resolution Protocol).

ЗАДАЧИ И УПРАЖНЕНИЯ

1 Просмотреть таблицу маршрутизации сервера.

2 Отследить маршрут до сервера.

3 Получить информацию о подключённом сервере и используемых портах.

4 Определить информацию о подключённом хосте и номерах используемых портов.

5 Определить текущую конфигурацию TCP/IP.

6 Проверить IP-адрес рабочей станции и получить новый адрес от DHCP-сервера.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1 В чём особенность MAC-адреса?
2 Перечислить виды IP-адреса?
3 Для каких целей используется кэш DNS?

Романова Любовь Владимировна

Методические указания по выполнению практических работ
по междисциплинарному курсу
Техническое обслуживание и ремонт
компьютерных систем и комплексов

для обучающихся по специальности
09.02.01

Ответственный за выпуск: Романова Л.В.

Издательство Технического колледжа им.С.И.Мосина ТулГУ
Тула, п.Мясново, 18 проезд, д.94

Министерство науки и высшего образования Российской Федерации
ФГБОУ ВО «Тульский государственный университет»
Технический колледж им. С.И. Мосина

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к практическим и самостоятельным работам

по дисциплине «Дискретная математика»

специальность

09.02.01 «Компьютерные системы и комплексы»

Тула 2023

УТВЕРЖДЕНЫ

цикловой комиссией естественнонаучных дисциплин

Протокол от «14» 01 2023 № 6

Председатель цикловой комиссии _____ Е.А. Рейм

Тема 1.1 Множества. Подмножества
Занятие 1. Выполнение теоретико-множественных операций

1 Теоретическая часть

Над множествами выполняются операции: объединение, пересечение, вычитание и декартово произведение, обозначаемые соответственно $\cup$, $\cap$, \, x.

Определение 1. *Объединением множеств A, B* называется множество $A \cup B$, состоящее из всех тех элементов, каждый из которых принадлежит хотя бы одному из множества *A, B:*

$$A \cup B = \{m \mid m \in A \text{ или } m \in B\}.$$

Определение 2. *Пересечением множеств A, B* называется множество $A \cap B$, состоящее из всех тех элементов, которые содержатся в обоих множествах *A, B:*

$$A \cap B = \{m \mid m \in A \text{ и } m \in B\}.$$

Заметим, что пересечение двух множеств может оказаться пустым множеством. В этом случае исходные множества называются непересекающимися.

Определение 3. *Разностью множеств A, B* называется множество $A \setminus B$, состоящее из всех элементов множества *A,* не содержащихся в множестве *B:*

$$A \setminus B = \{m \mid m \in A,\ m \notin B\}.$$

Определение 4. *Декартовым произведением множеств A, B* называется множество $A \times B$, состоящее их всевозможных пар вида (a,b), где $a \in A$, $b \in B$

$$A \times B = \{(a,b) \mid a \in A,\ b \in B\}.$$

2 Практическая часть

Задача 1. Для множества $A=\{1,2,3\}$, $B=\{2,4\}$ имеем:

$$A \cup B = \{1,2,3,4\},\ A \cap B = \{2\},\ A \setminus B = \{1,3\},\ B \setminus A = \{4\},$$
$$A \times B = \{(1,2),(1,4),\{2,2\},\{2,4\},\{3,2\},\{3,4\}\},$$
$$B \times A = \{(2,1),(2,2),\{2,3\},\{4,1\},\{4,2\},\{4,3\}\}.$$

Задача 2. Пусть *A* есть отрезок [1, 3], *B* - отрезок [2, 4]; тогда объединением $A \cup B$ будет отрезок [1, 4], пересечением $A \cap B$ - отрезок [2, 3], разностью $A \setminus B$ - полуинтервал [1, 2), $B \setminus A$ - полуинтервал (3, 4].

Задача 3. Пусть *A* есть множество прямоугольников, *B* - множество всех ромбов на плоскости. Тогда $A \cap B$ есть множество всех квадратов, $A \setminus B$- множество прямоугольников с неравными сторонами, $B \setminus A$ - множество всех ромбов с неравными углами.

3 Задания для самостоятельной работы

Задание 1 Универсальное множество представляет собой множество натуральных чисел, множество А – множество делителей числа 15, В – множество нечетных чисел, меньших 9. Описать каждое множество перечислением. Найти: $A \cup B$, $A \cap B$, $A \setminus B$, $B \setminus A$, $A \times B$, $B \times A$.

Задание 2 Найдите А∩В, А∪В, А\В, В\А, А×В, если

а) A={-6,-3,0,3,7,9} и *B={-3,0,7,8}*

б) A={-2,-1,1,2,3,d,e} и *B={a,b,c,d}*

в) A={1,2,3,4,23,45} и *B={4,34,46,51}*

Задание 3 На множестве U всех букв русского алфавита заданы множества *A, B, C: A* = {у, г, р, о, з, а}; *B* = {р, и, с, к}; *C* = {о, п, е, р, а, ц, и, я}.

Найдите следующие множества и изобразите их кругами Эйлера:

а) A∩B ; *б) A∪B* *в) (A∩B) ∪C* *г) (A∪C) ∩B* *d) U\(A∪B∪C)*

е) *U\(A∩B∩C)*

Задание 4 Даны отрезки *A* = [-5,6], *B* = (3;7], *C* = (6,11]. Найдите следующие множества и изобразите их кругами Эйлера:

а) (A∪B)∩C *б) (A∩B) ∪C* *в) A∩B* *г) (A∪B)\ (A∩B)*

d) (C∪B)\ (A∩B) е) *(A∪C)\(A∩B)*;

Занятие 2 Подсчет количества элементов в конечных множествах

1 Теоретическая часть

Мощность множества – это число его элементов, которое может быть только целым. Это единственная количественная характеристика, которой в полной мере оперирует теория множеств. Для обозначения мощности символ множества заключают в прямые скобки или используют отдельный символ $|A| = n$. При небольших величинах мощности она определяется простым подсчетом или по подходящей формуле, но для бесконечных множеств оценка их мощности может быть сделана только путем сравнения с мощностью других, достаточно известных множеств. Иначе говоря, для того, чтобы охарактеризовать мощность бесконечного множества, необходимо отыскать известное равномощное множество.

Если объединять конечные множества, у которых ни один элемент не совпадает, то мощность объединения определяется суммой их мощностей. При $|A_1| = n_1$, $|A_2| = n_2$, ... получим

$$\left| \bigcup_{1}^{k} A_i \right| = \sum_{1}^{k} n_i$$

При объединении конечных множеств, которые могут пересекаться, результирующая мощность определяется более сложным путем. Для этого приходится использовать специальные прием, который в комбинаторике называется “методом включения и исключения”. Мощность объединения двух множеств определяется по формуле

$$|A_1 \cup A_2| = |A_1| + |A_2| - |A_1 \cap A_2|$$

Мощность пересечения вычитается, чтобы не учитывались дважды элементы, принадлежащие обоим объединяемым множествам. Аналогично, при объединении трех множеств:

$$|A_1 \cup A_2 \cup A_3| = |A_1| + |A_2| + |A_3| - |A_1 \cap A_2| - |A_1 \cap A_3| - |A_2 \cap A_3| + |A_1 \cap A_2 \cap A_3|$$

Последний член возвращает те элементы, которые из-за принадлежности ко всем трем множествам были исключены трижды. Количество включений элементов со знаком “плюс” или “минус” в результирующую сумму поясняет рисунок 3.

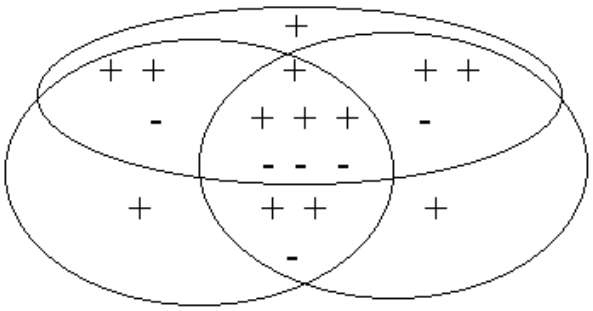


Рисунок 3

2 Практическая часть

Задача 1. Каждый студент группы «Информационная безопасность» занимается в свободное время либо в НСО, либо спортом. Сколько студентов в группе, если 23 увлекаются спортом, 12 занимаются в НСО, а 7 совмещают занятия в НСО и увлечения спортом?

Решение. Изобразим множества кругами Эйлера, обозначив множества «спортсменов» – *A* и «исследователей» – *B* (рисунок 4). Тогда $n(A \cap B)=7$, $n(A)=23$, $n(B)=12$; $n(AUB)=n(A)+n(B)-n(A \cap B)=23+12-7=28$.

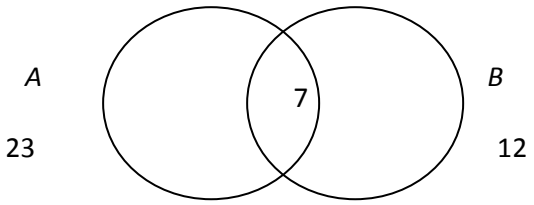


Рисунок 4 – Схематическое изображение условия Задачаа 1

Итак, в группе 28 студентов.

Задача 2. Пятьдесят лучших студентов колледжа наградили за успехи поездкой в Англию и Германию. Из них 5 не владели ни одним разговорным иностранным языком, 34 знали английский язык и 27 немецкий. Сколько студентов владели двумя разговорными иностранными языками?

Решение. Введем обозначения множеств:

E – множество студентов, не владеющих ни одним иностранным языком;

A – множество всех студентов, $|A|=50$;

B – множество студентов, владеющих английским языком, |*B*|=34;

C – множество студентов, владеющих немецким языком, |*C*|=27;

D – множество студентов, владеющих английским и немецким языками, |*D*|=*x*.

Представим множества графически с помощью кругов Эйлера (рисунок 5).

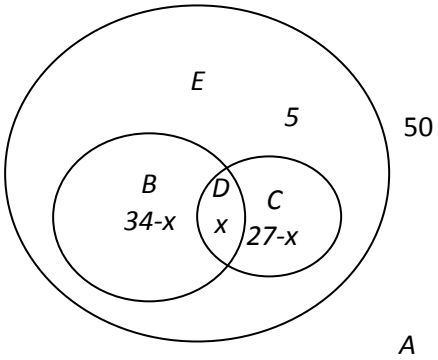


Рисунок 5 – Схематическое изображение условия Задача 2

Найдем |*D*| из уравнения

$$|B|+|C|-|D|=|A|-5$$

или

$$34+27-x=50-5,$$

отсюда

$$x = 16.$$

Следовательно, 16 студентов свободно общались на двух иностранных языков.

Задача 3. Из 35 студентов, побывавших на каникулах в Москве, все, кроме двоих, делились впечатлениями. О посещении Большого театра с восторгом вспоминали 12 человек, Кремля – 14, а 16 – о концерте, по три студента запомнили посещение театра и Кремля, а также театра и концерта, а четверо – концерта и пребывания в Кремле. Сколько студентов сохранили воспоминания одновременно о театре, концерте и Кремле?

Решение. Введем обозначения:

A – множество студентов, вспоминающих о театре *n(A)*=12;

B – о Кремле, *n(B)*=14;

C – о концерте, *n(C)*=16;

D – множество студентов, побывавших в поездке.

Изобразим, множества графически с помощью кругов Эйлера (рисунок 6).

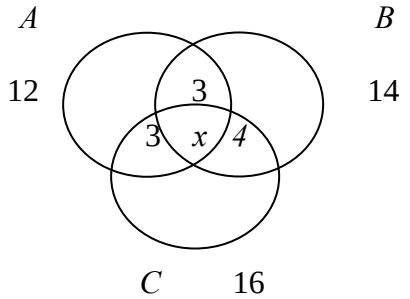


Рисунок 6 – Схематическое изображение условия Задача 3

$n(A \cup B \cup C) = n(A) + n(B) + n(C) - n(A \cap B) - n(A \cap C) - n(B \cap C) + n(A \cap B \cap C)$. Обозначим $n(A \cap B \cap C) = x$, тогда $35-2=12+14+16-3-3-4+x$, отсюда $x=1$.

Всего 1 студент рассказывал о трех культурных мероприятиях поездки.

3Задания для самостоятельной работы

Задание 1 В результате социологического опроса студентов факультета программирования о занятиях в свободное от уроков время выяснилось, что из 100 человек:

18 – любят только читать книги;

24 – читают книги, но не ходят в театр;

7 – читают книги и посещают театр;

10 – читают книги и посещают дискотеки;

47 – ходят на дискотеки;

9 – посещают театр и дискотеки;

13 – лежат на диване перед телевизором, занимаются только просмотром всех возможных каналов телевидения.

1. Сколько студентов любят ходить в театр?
2. Сколько студентов читают книги, посещают театр, но не дискотеки?
3. Сколько студентов посещают либо дискотеки, либо театр?
4. Сколько студентов, посещая дискотеки и театр, не любят читать книги?
5. Сколько студентов предпочитают только дискотеки?
6. Сколько студентов посещают либо дискотеки, либо театр, либо читают книги?

Задание 2 Из 100 студентов факультета программирования 42 посещают спортивные секции, 30 – занятия НСО, а 28 кружки художественной самодеятельности. На занятия НСО и спортом успевают ходить 5 студентов, спортом и художественной самодеятельностью занимаются 10, НСО и художественной самодеятельностью занимаются – 8, а сразу все три увлечения имеют три студента.

Сколько студентов:

а) не посещают ни одно из этих объединений по интересам;

б) занимаются только спортом;

в) занимаются либо в НСО, либо в художественной самодеятельности;

г) занимаются либо спортом, либо художественной самодеятельностью, но не в НСО;

д) занимаются или спортом, или художественной самодеятельностью, но не в НСО;

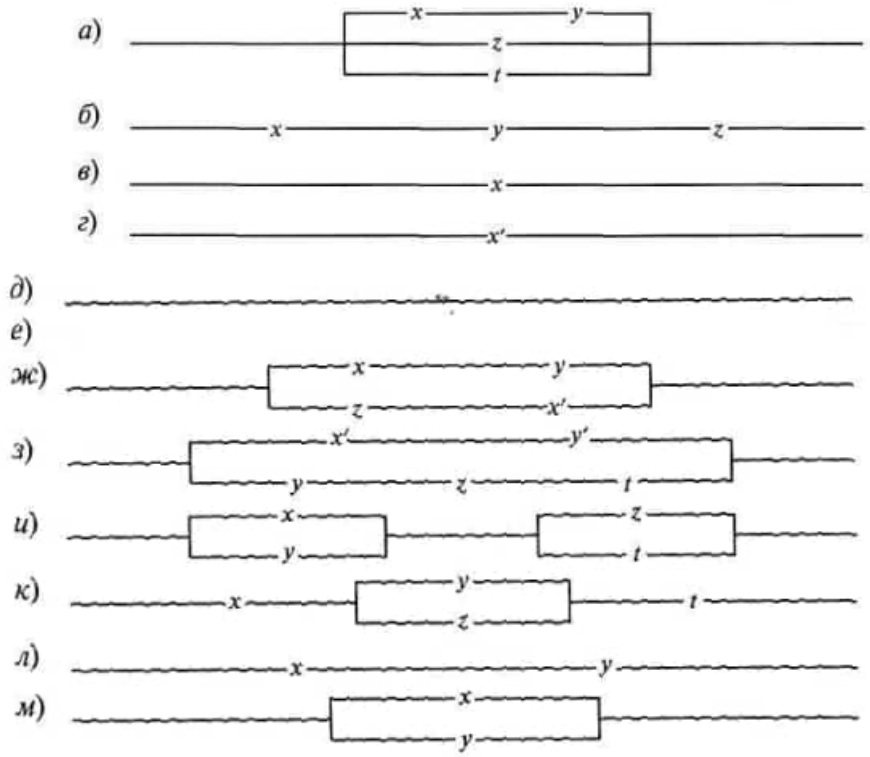
е) занимаются или в НСО, или художественной самодеятельностью, но не спортом?

Тема 2.2 Применение булевых функций к релейно- контактным схемам

Занятие 3. Анализ релейно-контактных схем

1.Теоретическая часть

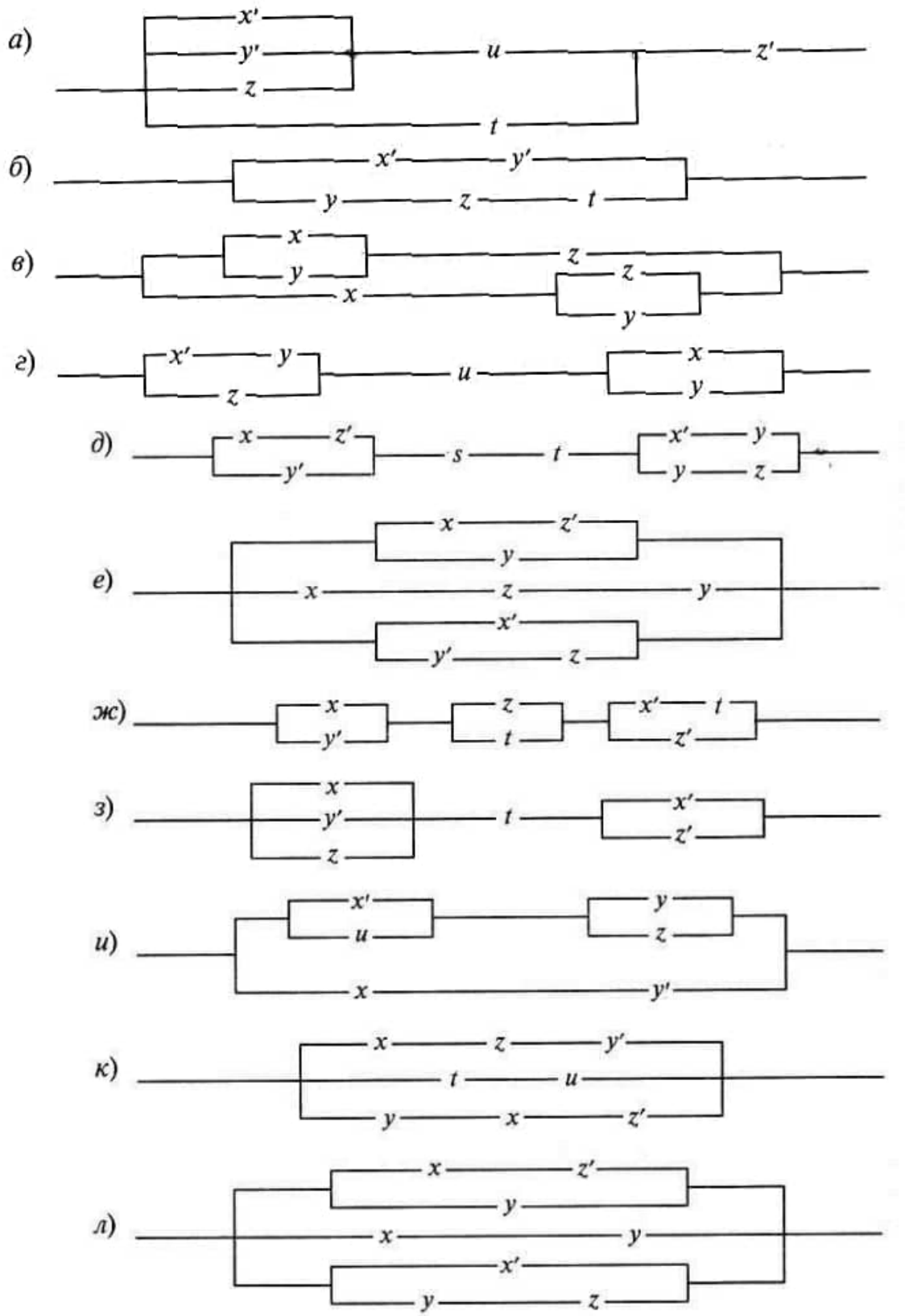
1. Найдите функции проводимости следующих релейно-контактных схем:



Решение, л) Ясно, что данная схема проводит электрический ток тогда и только тогда, когда оба независимых переключателя x и *y* замкнуты. Следовательно, функцией проводимости этой схемы будет такая булева функция от двух аргументов, которая принимает значение 1 в том и только в том случае, когда оба ее аргумента принимают значение 1. Как известно, такой функцией является конъюнкция $x \bullet y$. Итак, $\pi(x, y) = x \bullet y$.

м) Ясно, что данная схема проводит ток тогда и только тогда, когда по меньшей мере один из двух независимых переключателей *x* или *y* замкнут. Следовательно, функцией проводимости этой схемы будет такая булева функция от двух переменных, которая принимает значение 1 в том и только в том случае, когда хотя бы одна из переменных *x* или *y* принимает значение 1. Такой функцией является, как известно, дизъюнкция $x \vee y$. Итак, $\pi(x, y) = x \vee y$.

2. По данной релейно-контактной схеме найдите ее функцию проводимости и условия работы:



Решение л) Схема состоит из трех параллельных ветвей. Первая ветвь, в свою очередь, состоит из двух параллельных ветвей, в одной из которых последовательно соединены два контакта x и z', а в другой есть лишь один контакт y. Поэтому первая из трех параллельных ветвей имеет следующую функцию проводимости: xz' v y.

Вторая параллельная ветвь состоит из двух последовательно соединенных контактов *x* и *y* и поэтому имеет следующую функцию проводимости: *xy.*

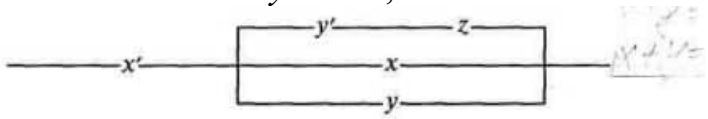
Наконец третья параллельная ветвь схемы состоит из двух параллельных ветвей, в одной из которых единственный контакт *x',* а в другой последовательно соединены контакты *y и Z-* Поэтому функция проводимости третьей ветви схемы есть *x′* v *yz.*

Итак, поскольку данная схема состоит из трех параллельно соединенных ветвей, функции проводимости которых мы нашли, то для нахождения функции проводимости всей схемы нужно рассмотреть дизъюнкцию найденных функций: *n(x, y, z) = (xz′* vy)v *vxyv(x'vyz).*

3. Постройте релейно-контактную схему с заданной функцией проводимости:

а) $(xy \vee z' \vee x')(x' \vee y)$;
б) $(x' \vee y)(yz \vee x) \vee uz$;
в) $x(yz \vee y'z') \vee x'(y'z \vee yz')$;
г) $(x' \vee y)ty'x'(y \vee z)$;
д) $x(yz \vee t) \vee xyz' \vee z(y \vee x')$;
е) $x'(yz' \vee x(ty \vee z(x \vee y')))$;
ж) $(x \vee yz)(xt \vee z(x' \vee y))$;
з) $xy' \vee u(v \vee z)x' \vee x'uv$;
и) $((z \vee x)y'u \vee x'v)xz$;
к) $x(y \vee z') \vee x' \vee (y \vee xz')x$;
л) $x'(y'z \vee x \vee y)$.

Р е ш е н и е. л) Схема представляет собой последовательное соединение контакта и схемы с функцией проводимости *y'*zvxvy. Последняя схема, в свою очередь, состоит из трех параллельных ветвей. В первой ветви последовательно соединены контакты *y′* и *z,* вторая содержит контакт x, третья — лишь контакт *y.* Итак, искомая схема имеет вид

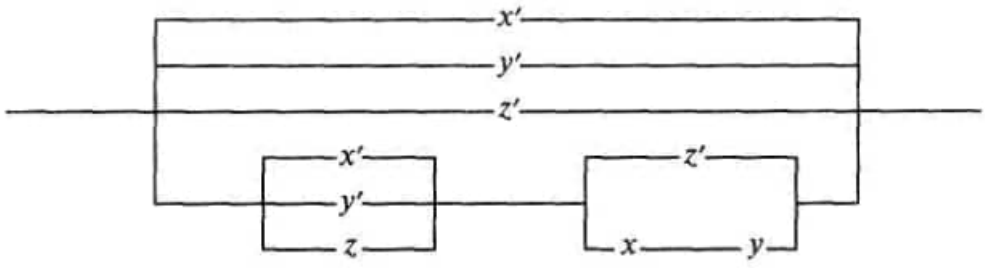


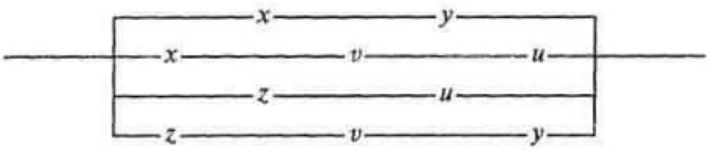
4. Постройте релейно-контактную схему с заданной функцией проводимости:

а) *{xy→x'y){xv zy)*
б) (x→*y)→ x'(y* v *z);*
в) (x → (y→ *z))→*(y→ x');
г) *(x | y')→* ((x v *y) |* (x v z*));*
д) *(z ↓xy)((x* v z)↓yz);
е) (x | (x↓y') | (x'↓(yvz'));
ж) *(x + y') v (x + z)(y'+ z');*
з) *(xy + z) → x'z;*
и) *(x′ + y′)(x ↔ y);*
к) *xy | (x′→ x(y* v z));
л) (x → (y→ z')) v(xyo↔ *z)*

Решение, л) Выразим сначала данную функцию через функции ', и, v, причем так, чтобы знак ' стоял бы лишь на переменных и не стоял на скобках: (x → (*y→ z'))* v (xy ↔z) = (x' v (y →z')) v *(xy→*z)(z → *xy)* =x'vy'vz'v (x' v y' v z)(z' v *xy).*

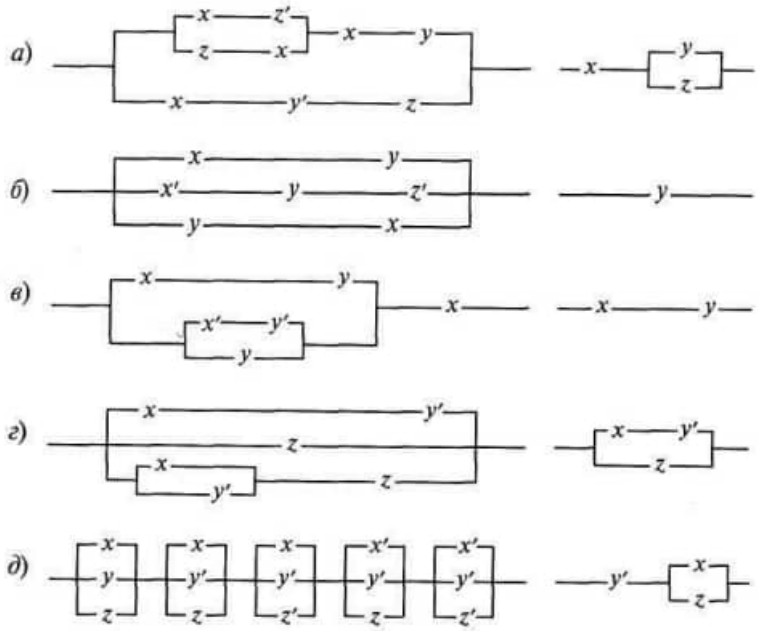
Соответствующая схема имеет вид

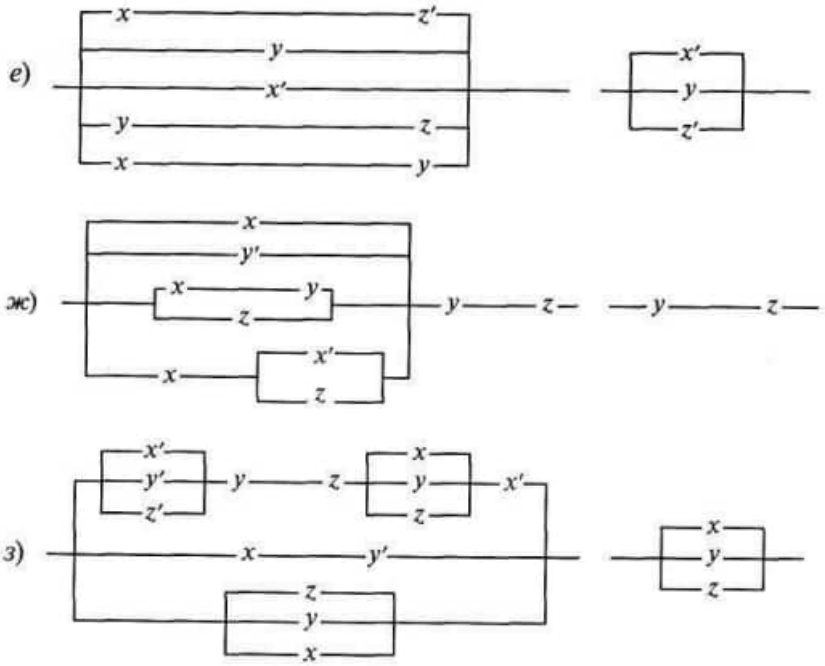


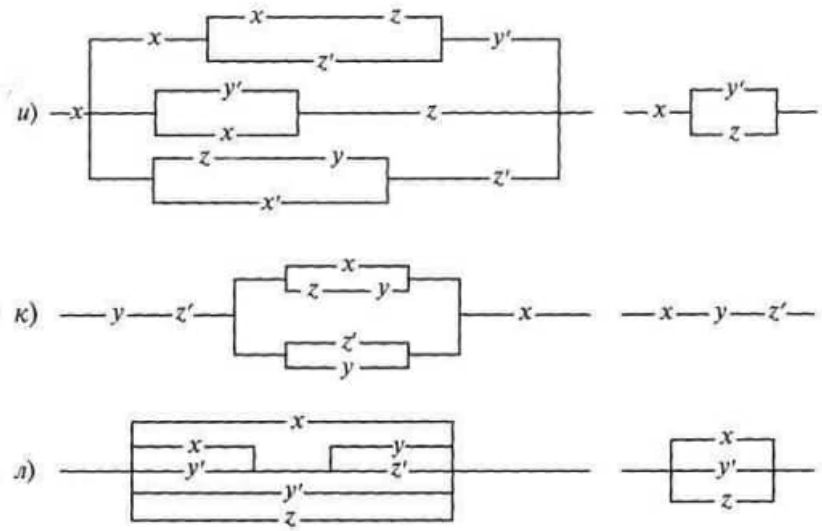


2.Практическая часть

1. Проверьте равносильность следующих релейно-контактных схем:





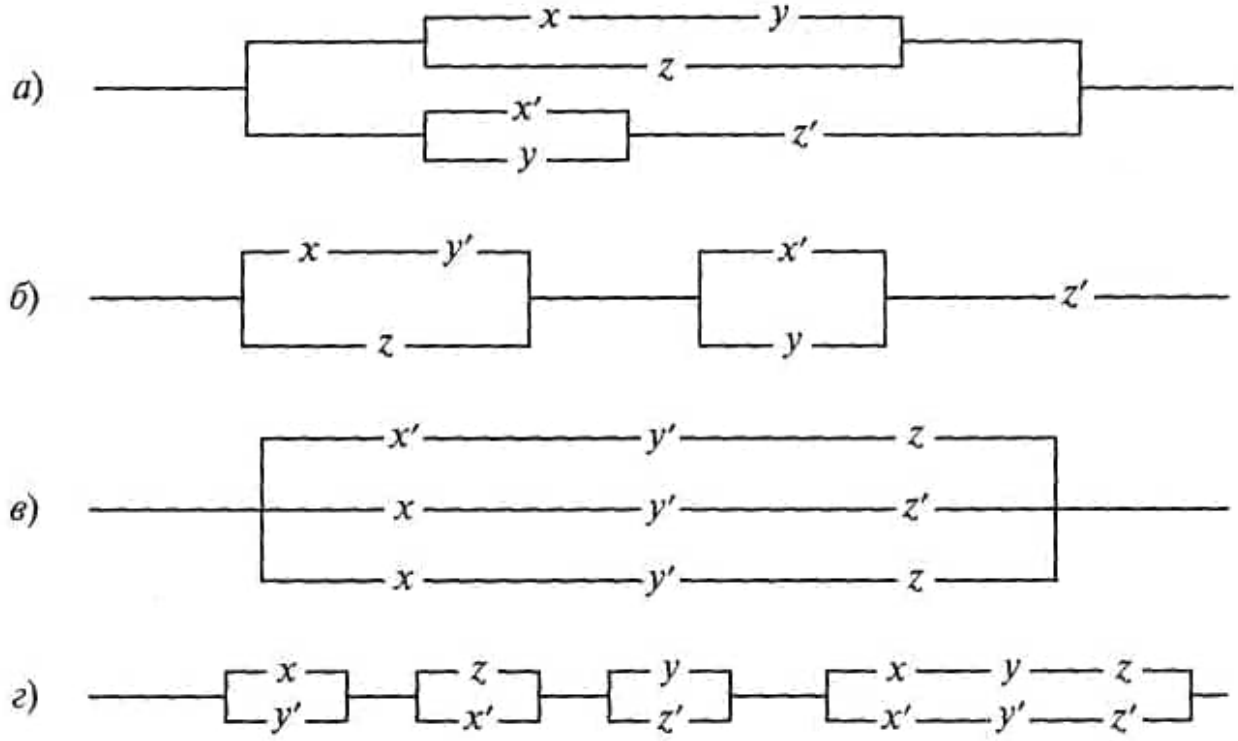


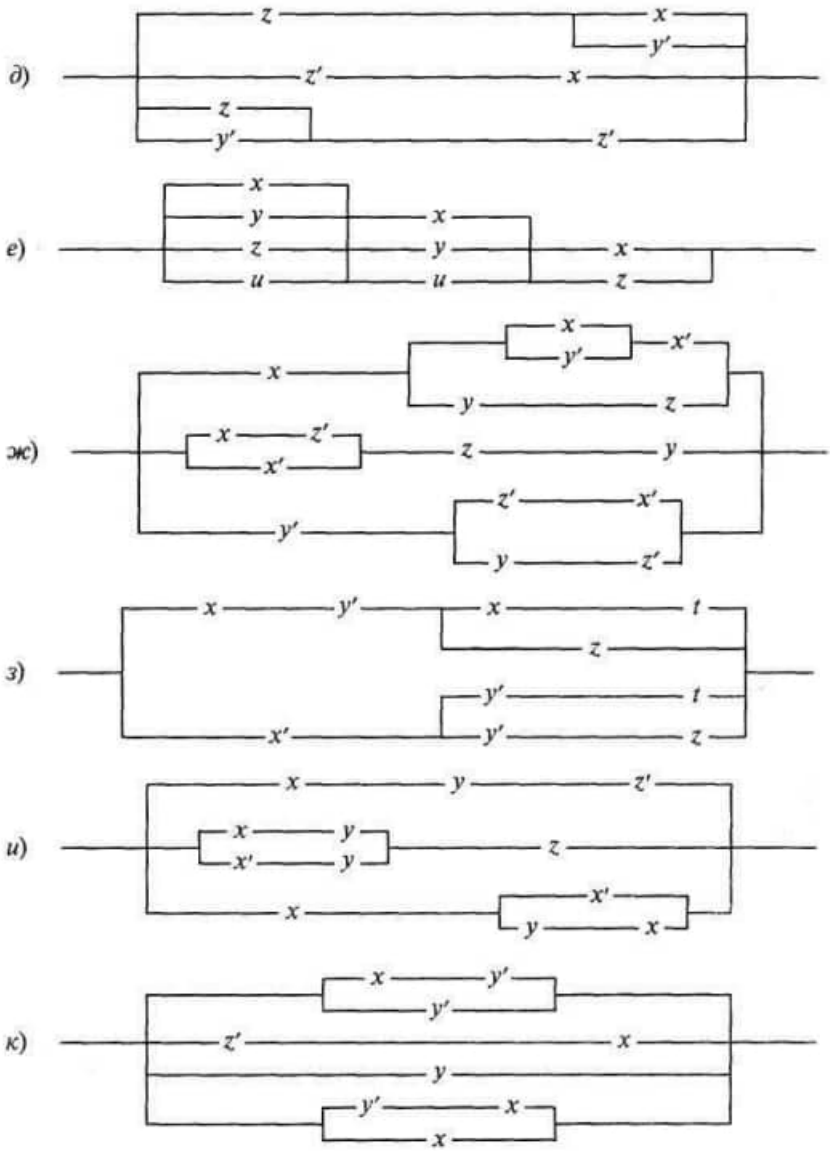
Р е ш е н и е. л) Сначала составим функцию проводимости первой из двух данных схем, а затем преобразуем ее:

$$\pi(x, y, z) = x \vee (x \vee y')(y \vee z') \vee y' \vee z = x \vee [(x \vee y')(y \vee z') \vee y'] \vee z =$$
$$= x \vee (x \vee y' \vee y')(y \vee z' \vee y') \vee z = x \vee (x \vee y')(y \vee y' \vee z') \vee z = x \vee (x \vee$$
$$\vee y')(1 \vee z') \vee z = x \vee (x \vee y')1 \vee z = x \vee (x \vee y') \vee z = x \vee y' \vee z.$$

Ясно, что полученная функция (булева функция, конечно, осталась той же самой, а изменилась лишь форма ее аналитической записи) является функцией проводимости второй из двух данных схем.

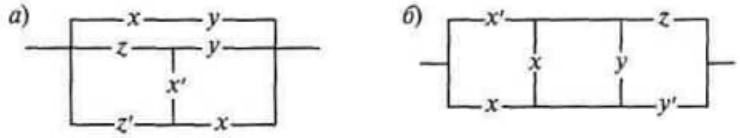
2. Упростите следующие релейно-контактные схемы:

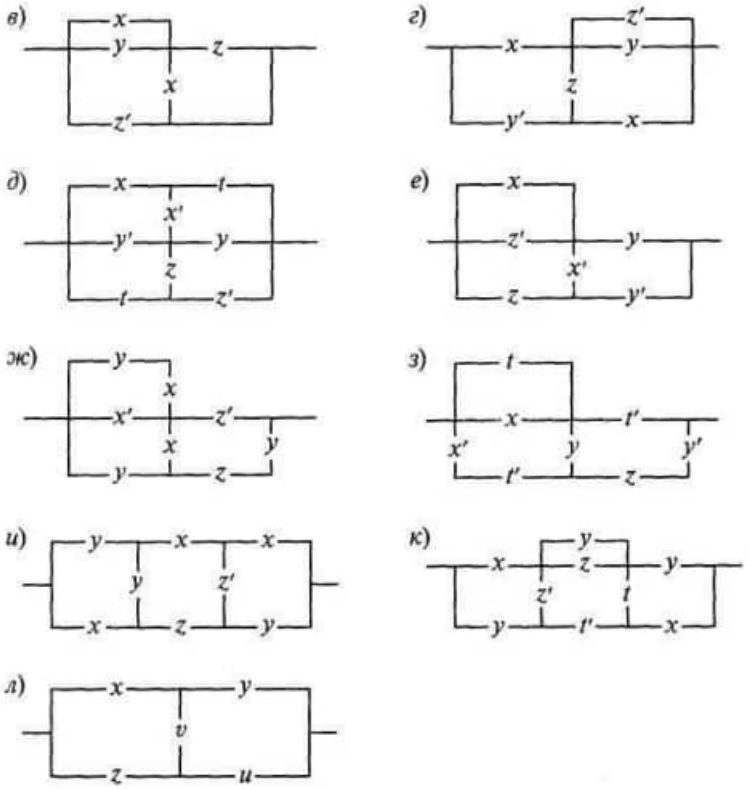




Задания для самостоятельной работы

1. Упростите следующие релейно-контактные схемы (называемые мостиковыми):





Решение, л) Ток может протекать через контакты *x* и *y,* если они замкнуты, т.е. если конъюнкция *xy* равна 1. Далее ток может пройти через контакты x, *v, u,* если они замкнуты, т. е. если конъюнкция *xvu* равна 1. Аналогично, ток может пройти через контакты *z, u* или *z, v, y,* т.е. ток проходит через схему, если одна из конъюнкций *zu* или *zvy* равна 1. Итак, ток пройдет через схему в том и только в том случае, когда хотя бы одна из конъюнкций *xy, xvu, zu* или *xvy* равна 1, т. е. когда равна 1 дизъюнкция всех этих конъюнкций. Следовательно, функция проводимости данной схемы такова: *n(x, y, z, u, v) = xy* v *xvu* v *zu* v *zvy,* а сама схема может быть реализована обычной (не мостиковой) схемой, но с большим числом контактов, чем исходная схема:

Занятие4. Анализ релейно-контактных схем
1.Теоретическая часть

1. Постройте наиболее простые релейно-контактные схемы по заданным условиям работы

$$\pi(x, y, z, t) = (x \vee y \vee z \vee t)(x \vee y \vee z \vee t')(x \vee y' \vee z' \vee t')(x' \vee y' \vee z' \vee t)(x' \vee y' \vee z' \vee t') = x(x \vee y \vee z \vee tt')(x \vee y' \vee zz' \vee t')(x' \vee y' \vee z' \vee tt') = (x \vee y \vee z)(x \vee y' \vee t')(x' \vee y' \vee z') = [x \vee (y \vee z)(y' \vee t')](x' \vee y' \vee z') = (x \vee y'z \vee yt' \vee zt')(x' \vee y' \vee z') = x'y'z \vee x'yt' \vee x'zt' \vee xy' \vee y'z \vee y'zt' \vee xz' \vee yz't' = x'yt' \vee x'(y \vee y')zt' \vee xy' \vee y'z \vee x(y \vee y')z' \vee (x \vee x')yz't' =$$

$$\pi(x, y, z, t) = (x \vee y \vee z \vee t)(x \vee y \vee z \vee t')(x \vee y' \vee z' \vee t')(x' \vee y' \vee z' \vee t)(x' \vee y' \vee z' \vee t') = x(x \vee y \vee z \vee tt')(x \vee y' \vee zz' \vee t')(x' \vee y' \vee z' \vee tt') = (x \vee y \vee z)(x \vee y' \vee t')(x' \vee y' \vee z') = [x \vee (y \vee z)(y' \vee t')](x' \vee y' \vee z') = (x \vee y'z \vee yt' \vee zt')(x' \vee y' \vee z') = x'y'z \vee x'yt' \vee x'zt' \vee xy' \vee y'z \vee y'zt' \vee xz' \vee yz't' = x'yt' \vee x'(y \vee y')zt' \vee xy' \vee y'z \vee x(y \vee y')z' \vee (x \vee x')yz't' =$$

$$= x'yt' \vee x'yzt' \vee x'y'zt' \vee xy' \vee y'z \vee xyz' \vee xy'z' \vee xyz't' \vee x'yz't'.$$

Указание. Используя СДН-форму (или СКН-форму), найдите сначала аналитическое выражение для функции π проводимости исходной схемы. Максимально упростите полученное выражение. После этого начертите релейно-контактную схему, для которой полученное выражение представляет собой функцию проводимости.

2 Постройте наиболее простую релейно-контактную схему с четырьмя переключателями, которая должна проводить электрический ток тогда и только тогда, когда выполнено по меньшей мере одно из следующих трех условий: а) переключатель x замкнут и только один из переключателей y или z замкнут; б) t разомкнут и только два из остальных переключателей разомкнуты; в) только два переключателя, но не переключатели y и t, замкнуты.

Решение. Используя условия, которым должна удовлетворять искомая схема, составим сначала таблицу значений функции проводимости π этой схемы (в последнем столбце таблицы указано то условие, по которому функция π принимает значение 1 на данном наборе значений аргументов).

Зная теперь все наборы значений аргументов, на которых функция π обращается в 0, запишем выражение для нее, используя совершенную конъюнктивную нормальную форму (потому что наборов значений аргументов, на которых функция обращается в 0, значительно меньше, чем наборов, на которых функция обращается в 1, и, значит, СКН-форма будет более простой, нежели СДН-форма), и затем упростим его:

$$\pi(x, y, z, t) = (x \vee y \vee z \vee t)(x \vee y \vee z \vee t')(x \vee y' \vee z' \vee t')(x' \vee y' \vee z' \vee t)(x' \vee y' \vee z' \vee t') = x(x \vee y \vee z \vee tt')(x \vee y' \vee zz' \vee t')(x' \vee y' \vee z' \vee tt') = (x \vee y \vee z)(x \vee y' \vee t')(x' \vee y' \vee z') = [x \vee (y \vee z)(y' \vee t')](x' \vee y' \vee z') = (x \vee y'z \vee yt' \vee zt')(x' \vee y' \vee z') = x'y'z \vee x'yt' \vee x'zt' \vee xy' \vee y'z \vee y'zt' \vee xz' \vee yz't' = x'yt' \vee x'(y \vee y')zt' \vee xy' \vee y'z \vee x(y \vee y')z' \vee (x \vee x')yz't' = x'yt' \vee x'yzt' \vee x'y'zt' \vee xy' \vee y'z \vee xyz' \vee xy'z' \vee xyz't' \vee x'yz't'.$$

x	y	z	t	π	Условие
0	0	0	0	0	
0	0	0	1	0	
0	0	1	0	1	*б*
0	0	1	1	1	*в*
0	1	0	0	1	*б*
0	1	0	1	0	
0	1	1	0	1	*в*
0	1	1	1	0	
1	0	0	0	1	*б*
1	0	0	1	1	*в*
1	0	1	0	1	*а*
1	0	1	1	1	*а*
1	1	0	0	1	*а*
1	1	0	1	1	*а*
1	1	1	0	0	
1	1	1	1	0	

3 Каждый из трех членов комитета, голосуя «за», нажимает на кнопку. Построить по возможности более простую схему, через которую проходил бы ток и включал электрическую лампочку тогда и только тогда, когда не менее двух членов комитета голосуют «за».

4.Комитет состоит из пяти человек. Решение выносится большинством голосов. Если председатель «против», то решение не принимается. Построить такую схему, чтобы, голосуя «за», каждый из пяти человек нажимал бы на кнопку, и в случае принятия решения зажигалась бы сигнальная лампочка.

5. Постройте релейно-контактную схему с четырьмя переключателями, которая проводит ток тогда и только тогда, когда замыкаются некоторые, но не все переключатели.

6 Постройте релейно-контактную схему с пятью переключателями, которая проводит ток тогда и только тогда, когда замкнуты все ее переключатели или когда не *замкнут ни один* из них.

7. Постройте схему с тремя переключателями, которая замыкается тогда и только тогда, когда замкнуты либо ровно один, либо ровно два переключателя. Используйте не более шести контактов.

8. Начертите схему с пятью переключателями, которая замыкается, если и только если замкнуты ровно четыре из этих переключателей.

9. Требуется составить схему с четырьмя переключателями *x, y, z, t.* Схема должна проводить ток тогда и только тогда, когда будут замкнуты переключатели *x* и *y или z* и t

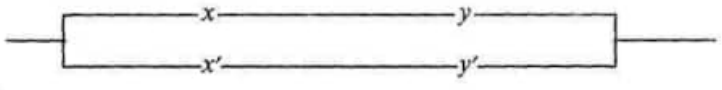
10.Начертите схему, которая замыкается тогда и только тогда, когда либо переключатель *x* замкнут, либо переключатель *y* замкнут, либо переключатель *z* разомкнут.

2.Практическая часть

1. Постройте релейно-контактную схему с пятью переключателями, которая проводит ток тогда и только тогда, когда меньшая часть переключателей замкнута, но если последний переключатель замкнут, то схема проводит ток, независимо от положения остальных переключателей.

2.Имеется одна лампочка в лестничном пролете двухэтажного дома. Постройте схему так, чтобы на каждом этаже своим выключателем можно было бы включать и выключать лампочку, независимо от положения другого выключателя

Решение. Функция проводимости $\pi(x, y)$ такой схемы должна обладать тем свойством, что ее значение меняется всякий раз, когда меняется значение одного ее аргумента. Следовательно, например: $\pi(1, 1) = \pi(0, 0) = 1$; $\pi(0, 1) = \pi(1, 0) = 0$. Используя СДН-форму, отсюда легко получаем: $\pi(x, y) = xyv\ x'y'$, а соответствующая схема имеет вид



3 Спроектируйте релейно-контактную схему, позволяющую включать и выключать электрическую лампочку с помощью трех независимых переключателей. Сколькими способами можно сконструировать такую схему?

4 Спроектируйте релейно-контактную схему, которая позволяла бы включать и выключать электрическую лампочку с помощью четырех независимых переключателей.

5. Постройте релейно-контактную схему с пятью переключателями, каждый из которых позволял бы включать и выключать в любой момент одну и ту же лампочку.

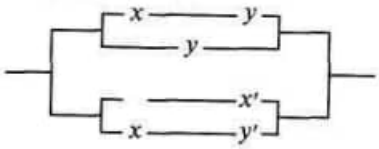
6. Опишите функцию проводимости релейно-контактной схемы с *n* переключателями, позволяющей включать и выключать лампочку любым из них .. Сколько существует таких булевых функций от *n* аргументов?

3 Задания для самостоятельной работы

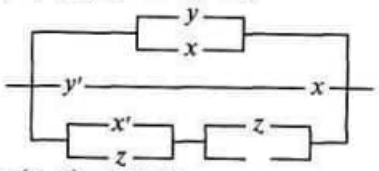
1. Схема должна быть замкнута, если переключатель *x* разомкнут, но без того, чтобы *y* и *z* (но не оба вместе) были замкнуты: в последнем случае цепь разомкнута. Кроме того, цепь должна быть замкнута, когда *x* замкнут, но при этом *y* и *z* не должны быть одновременно замкнуты или одновременно разомкнуты; в последнем случае цепь также размыкается. Постройте удовлетворяющую этим условиям релейно-контактную схему с наименьшим возможным числом контактов.

2. Какой контакт, x, y, x' или y', необходимо вставить в вакантное место релейно-контактной схемы, чтобы функция проводимости полученной схемы стала бы равна данной булевой функции:

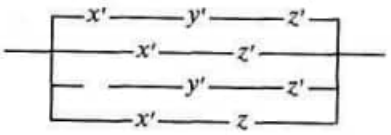
ж) $\pi(x, y) = x \vee y$;



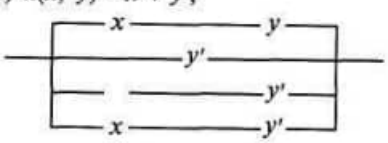
з) $\pi(x, y, z) = x \vee y \vee z$;



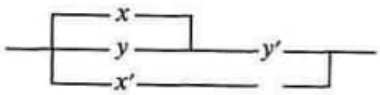
и) $\pi(x, y, z) = x'$;



к) $\pi(x, y) = x \vee y'$;



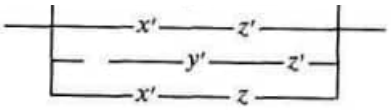
л)

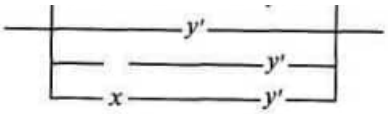


Р е ш е н и е. л) Обозначим искомый контакт @ и составим функцию проводимости данной схемы: $\pi(x, y) = (x \vee y)\, y' \vee x'@$.

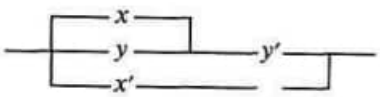
Упростим ее, используя свойства булевых функций: $\pi(x, y) = xy' \vee yy' \vee x'@ = xy' \vee 0 \vee x'@ = xy' \vee x'@$.

По условию должно быть: $xy' \vee x'@ = x' \vee y'$. Сравнивая выражения в левой и правой частях последнего равенства, заключаем, что $@ = x'$.





л)



Тема 2.4 Исчисление предикатов

Занятие 5 Аксиоматическая теория исчисления предикатов

Подобно аксиоматической теории высказываний (L) может быть построена и аксиоматическая теория предикатов - формализованное исчисление предикатов (PL). Она начинается с выбора системы аксиом и правил вывода .

Под аксиоматической теорией, построенной на основе системы аксиом Σ, понимается совокупность всех теорем, доказываемых, исходя из этой системы аксиом.

Далее в формализованном исчислении предикатов может быть доказана теорема о дедукции, используемая для построения доказательств. Затем устанавливается полнота формализованной теории PL (теорема Геделя о полноте - 1930 г.): теоремами теории PL являются все те и только те формулы, которые логически общезначимы, т.е. являются тавтологиями алгебры предикатов.

Определение. Пусть A, B, C - произвольные формулы, x — произвольная переменная, A(x) - произвольная формула, c - произвольная переменная, не обязательно отличная от x, A(c) - результат подстановки c вместо всех свободных вхождений x в A(x) и c свободна для x в A(x). Тогда следующие записи есть аксиомы:

1. $A\to(B\to A)$.
2. $(A\to B)\to(A\to(B\to C))\to(A\to C)$.
3. $A\to(B\to A\wedge B)$.
4. $A\wedge B\to A$.
5. $A\wedge B\to B$.
6. $A\to A\vee B$.
7. $B\to A\vee B$.
8. $(A\to C)\to((B\to C)\to(A\vee B\to C))$.
9. $(A\to B)\to((A\to\neg B)\to\neg A)$.
10. $\neg\neg A\to A$.
11. $(A\to B)\to((B\to A)\to(A\leftrightarrow B))$.
12. $(A\leftrightarrow B)\to(A\to B)$.
13. $(A\leftrightarrow B)\to(B\to A)$.
14. $\forall\, xA(x)\to A(c)$.
15. $A(c)\to\exists\, xA(x)$.

Аксиомы (1)-(13) те же , что и в теории высказываний (L) для схемы 1. В качестве аксиом теории L возможно использование схемы 2. Аксиома (14) - единственная аксиома для квантора $\forall$, который удаляется ею, а (15) - для квантора $\exists$, который вводится ею.

Правилами вывода в теории предикатов (PL) являются следующие три процедуры перехода:

а) от формул вида A и A→B к формуле вида B для любых A и B (правило modus ponens, MP);

б) от формулы вида B→A(x) к формуле вида B→∀ xA(x), для любых формул B и A(x), если B не содержит свободных вхождений x (правило обобщения O);

в) от формулы вида A(x)→B к формуле вида ∃ xA(x)→B для любых формул A(x) и B, если B не содержит свободных вхождений x (правило конкретизации $К_Н$).

Определение формального доказательства, доказуемой формулы (теоремы), формального вывода, выводимой формулы делается также, как и в теории L с учетом аксиомных схем (14), (15) и правил вывода O и $К_Н$.

Правила введения и удаления кванторов.

Теорема дедукции для теории PL.

Пусть Г, A |—B и в выводе формулы B из множества формул Г∪ {A} правила O и $К_Н$ не применяются ни к какой переменной, входящей свободно в формулы множества Г∪ {A}, кроме тех случаев, когда заключения правил O и $К_Н$ находятся раньше первого вхождения любой из формул множества Г∪ {A} в качестве посылок. Тогда Г|—A→ B.

К имеющимся правилам введения и удаления для операторов ¬, ∧,∨,→,↔ из теории L добавим четыре правила для ∀ и ∃ .

Пусть x - произвольная переменная, A(x) - произвольная формула, c - произвольная переменная, A(c) - результат подстановки c вместо всех свободных вхождений x в A(x), Г — любое множество формул, B — произвольная формула. Тогда справедливы следующие правила введения и удаления, представленные в таблице 3.2.

Таблица 3.2.

Квантор	Введение			Удаление		
∀	15	Если Г|—A(x), то Г|—∀xA(x), x не входит свободно в Г	ВКО Уоб	16	∀xA(x) |—A(c), c- свободная для x в A(x)	УКО УКо
∃	17	A(c) |—∃xA(x), c – свободна для x в A(x)	ВКС ЭО	18	Если Г,A(x) |—B, то Г,∃xA(x) |—B, x не входит свободно в Г и B	УКС

где: Уоб - универсальное обобщение;
Уко - универсальная конкретизация;
ЭО - экзистенциальное обобщение;
ВКО - введение квантора общности;
УКО – удаление квантора общности;
ВКС – введение квантора существования;
УКС – удаление квантора существования.

Таким образом, формализованное исчисление предикатов, как и исчисление высказываний, непротиворечиво, а его аксиомы независимы. В отличие от теории L, теория PL есть неразрешимая аксиоматическая теория, т.е. не существует единого алгоритма , позволяющего для любой формулы алгебры предикатов определить, является ли она теоремой формализованного исчисления предикатов (из-за бесконечности предметной области, которая приводит в общем случае к бесконечным таблицам истинности, за исключением одноместных предикатов).

2. Практическая часть

1.Какие из следующих выражений являются предикатам

А) «x делится на 5» ($x \in N$);

Б) «Река x впадает в озеро Байкал» (x пробегает множество названий всевозможных рек);

В) «x^2+2x+4» ($x \in R$);

Г) «$(x+y)^2=x^2+2xy+y^2$» ($x,y \in R$)

Д) «x есть брат y» (x,y пробегают множества всех людей);

Е) «x и y лежат по разные стороны от z»(x,y пробегают множества всех точек, а z – всех прямых одной плоскости»;

Ж) «ctg45°=1»;

З) «x перпендикулярна y»(x,y пробегают множество всех прямых одной плоскости);

И) «$x^2+x-6=0$» ($x \in R$);

К) «Для всех вещественных чисел x выполняется равенство $x^2+x-6=0$»

2. Для каждого из следующих высказываний найдите предикат (одноместный или многоместный), который обращается в данное высказывание при замене предметных переменных подходящими значениями из соответствующих областей:

А) «3+4=7»

Б) «Вера и Надежда – сестры»

В) «Сегодня вторник»

Г) «Город Саратов находится на берегу Волги»

Д) «sin30°=0.5»

Е) «А.С Пушкин – великий русский поэт»

Ж) «$3^2+4^2=5^2$»

З) «Река Индигирка впадает в озеро Байкал»

И) «Если число делится на 3, то оно делится на 9»

К) «Луна – спутник Марса»

Л) «$tg(\pi/4)=1$»

Построив такой предикат постарайтесь или точно указать его область истинности, или как её обрисовать.

Решение: л) Можно указать три предиката, каждый из которых обращается в данное высказывание при соответствующей подстановке. Первый предикат одноместный «tgx=1» ($x \in R \setminus (\pi|2+\pi n: n \in N$). Он превращается в

данное высказывание при подстановке $x=\pi/4$. Получающееся высказывание истинно. Указанным значением не исчерпывается множество истинности построенного предиката. Как нетрудно установить это множество следующее; $(\pi/4+\pi n: n\in N)$. Второй предикат также одноместный «$tg(\pi/4)=y$». Он превращается в данное высказывание при подстановке $y=1$. Ясно, что этим значением и исчерпывается множество истинности этого предиката. Наконец можно построить третий предикат, двухместный «$tgx=y$» $(x\in R\setminus(\pi|2+\pi n: n\in N, y\in R)$. Он превращается в данное высказывание при подстановке $x=\pi/4$, $y=1$. Его область истинности представляет собой множество упорядоченных пар, совокупность которых изображается в виде бесконечного семейства кривых, называемых тангенсоидами.

3. Прочитайте следующие высказывания и определите, какие из них истинные, а какие ложные, считая, что все переменные пробегают множество действительных чисел:

А) $(\forall x)(\exists y)(x + y = 7)$

Б) $(\forall y)(\exists x)(x + y = 7)$

В) $(\exists x)(\forall y)(x + y = 7)$

Г) $(\forall x)(\forall y)(x + y = 7)$

Д) $[(\forall x)(\forall y)(x + y = 7)] \rightarrow (3 = 4)$

Е) $(\forall x)[(x^2 > x) \leftrightarrow (x > 1) v (x < 0))]$;

Ж) $(\forall a)\{[(\exists x)(ax = 6)] \leftrightarrow (a \neq 0)\}$;

З) $(\forall b)(\exists a)(\forall x)(x^2 + ax + b > 0\}$;

И) $(\forall x)[((x > 1) v (x < 2)) \leftrightarrow (x = x)]$;

К) $(\exists b)(\exists a)(\exists x)(x^2 + ax + b) = 0)$;

Л) $(\exists a)(\forall b)(\exists x)(x^2 + ax + b) = 0)$;

Решение: а) Двухместный предикат «x+y=7» задан над множеством действительных чисел R. Это означает, что вместо каждой из двух его предметных переменных x и y могут быть подставлены действительные числа. Если такая подстановка сделана вместо обеих переменных, например «6+3=7», то предикат превращается в высказывание (в данном случае ложное). Но данный двухместный предикат «x+y=7» может быть превращен в высказывание и другим путем: именно путем применения к нему операций квантификации (взятия квантора общности или квантора существования). Применим сначала к двухместному предикату «x+y=7» операцию взятия квантора существования по переменной y. Получим уже одноместный предикат «$((\exists y))$(x+y=7)» относительно переменной x, которая пробегает множество R. Говорят, что в полученном выражении переменная y связана, а переменная x свободна. Вместо переменной y мы уже ничего не можем подставлять, в то время как вместо x могут быть поставлены действительные числа, в результате чего одноместный предикат будет превращаться в высказывания. Например, высказывание «$((\exists y))$(10+y=7)» можно прочитать так: «Существует действительное

число у такое, что 10+у=7». Ясно, что это высказывание истинно. (В качестве такого у, существование которого утверждает это высказывание , нужно взять действительное число -3). Легко далее понять, что какое бы действительное число x_0 мы ни подставили вместо переменной х в предикат «$(\exists y)$(10+у=7)» , предикат превращается в истинное высказывание. Действительно, в качестве такого числа у, существование которого утверждает высказывание, нужно взять разность 7-х. Это обстоятельство, согласно определению операции взятия квантора общности означает, что получающееся высказывание $(\forall x)(\exists y)(x + y = 7)$ истинно. Его можно прочитать следующим образом: «Для любого действительного числа существует такое действительное число, сумма которого с первым равна 7». В выражении $(\forall x)(\exists y)(x + y = 7)$ уже нет свободных переменных. Обе переменные х и у стоят под знаками кванторов и поэтому являются связанными. Само же выражение не является предикатом, оно есть высказывание истинное, как было установлено ранее.

Б) Высказывание $(\forall y)(\exists x)(x + y = 7)$ можно прочитать так: «Существует такое действительное число, которое после прибавления к любому действительному числу в сумме дает 7 ». Нетрудно понять, что это утверждение ложно. В самом деле, рассмотрим одноместный предикат $(\exists x)(x + y = 7)$ относительно переменной у, применением к нему квантора существования получается данное высказывание. Ясно, что какое бы действительное число ни подставить вместо предметной переменной у, например $(\exists x)(x + 4 = 7)$ предикат будет превращаться в ложное высказывание. (Высказывание $(\exists x)(x + 4 = 7)$ ложно, так как одноместный предикат $(x + 4 = 7)$ превращается в ложное высказывание, например при подстановке вместо переменной х числа 5). Поэтому высказывание $(\forall y)(\exists x)(x + y = 7)$, получающееся из одноместного предиката $(\exists x)(x + y = 7)$ применением операции взятия квантора существования по у ложно.

И) Это высказывание читается так: «Любое действительное число равно самому себе тогда и только тогда, когда оно больше 1 или меньше 2». Чтобы выяснить, истинно или ложно данное высказывание, будем, например, искать такое действительное число х, которое превратило бы одноместный предикат $(\forall x)[((x > 1) \vee (x < 2)) \leftrightarrow (x = x)]$ в ложное высказывание. Если нам удастся найти такое число, то данное высказывание, получающееся из этого предиката, «навешиванием», т.е. применением операции) взятия квантора общности, ложно. Если же мы придем к противоречию, предположив, что такое х существует, то данное высказывание истинно.

Ясно, что предикат «х=х» превращается в истинное высказывание при подстановке вместо х любого действительного числа, т.е. является тождественно истинным. Спрашивается, можно ли указать действительное число, которое превратило бы предикат «$(x > 1) \vee (x < 2)$» в ложное

высказывание? Нет, потому, что какое бы действительное число мы не взяли, оно либо больше 1, либо меньше 2(либо одновременно и больше 1 и меньше 2, что вовсе не возбраняется в нашем случае).Следовательно, предикат «$(x>1) \vee (x<2)$» тождественно истинен. Тогда тождественно истинным будет и предикат $[((x>1) \vee (x<2)) \leftrightarrow (x=x)]$, а значит, данное высказывание $(\forall x)[((x>1) \vee (x<2)) \leftrightarrow (x=x)]$ по определению операции взятия квантора общности истинно.

4. Из следующих предикатов с помощью кванторов постройте всевозможные высказывания и определите, какие из них истинны, а какие ложны ($x \in R$):

А) $x^2+2x+1=(x+1)^2$;

Б) $(x-3)(x+3)<x^2$;

В) $e^{|x|} < \ln|x| (x \neq 0)$;

Г) $(x^2+1=0) \rightarrow ((x=1) \vee ((x=2))$;

Д) $(x<0) \vee (x=0) \vee (x>0)$;

е) $|x-y| \geq ||x|-|y||$;

ж) $\sin x=\sin y$;

з) $x^2=y^2 \rightarrow x=y$;

и) $(x+y)^2=x^2+2xy+y^2$;

к) $|x-y| \leq 3$:

л) $x^2=25$;

м) $x^2+y^2=16$;

Решение: л) Из этого одноместного предиката с помощью кванторов можно построить два высказывания «$(\forall x)x^2=25$» и «$(\exists x)x^2=25$». Первое высказывание читается так: «Квадрат любого действительного числа равен 25». Оно ложно и сточки зрения здравого смысла, и согласно определению взятия квантора общности; предикат «$x^2=25$» не является тождественно истинным, и поэтому высказывание «$(\forall x)x^2=25$» тождественно ложно. Второе высказывание читается так «Существует действительное число, квадрат которого равен 25». Это высказывание истинно, т.к. предикат «$x^2=25$» не является тождественно ложным.

М) Предварительно посмотреть решение задачи 9.3.а. Из данного двухместного предиката $x^2+y^2=16$ можно построить четыре высказывания с помощью двух комбинаций кванторов «$(\forall x)(\forall y)x^2+y^2=16$», «$(\exists x)(\forall y)x^2+y^2=16$», «$(\forall y)(\exists x)x^2+y^2=16$», «$(\exists x)(\exists y)x^2+y^2=16$». Чтение этих высказываний сделать самостоятельно. Исследуем вопрос истинности предикатов. Посмотрим первые два высказывания. Они получены из одноместного предиката $(\forall y)x^2+y^2=16$ с помощью кванторов общности и существования. Этот предикат превращается в ложное высказывание при подстановке вместо него в предметной переменной x любого действительного числа. Другими словами этот предикат тождественно ложен. Следовательно, применение к этому предикату, как квантора общности, так и квантора существования приводит к ложным высказываниям. Итак, оба высказывания ложны.

Обратимся теперь к двум последним высказываниям. Они получены из одноместного предиката $(\exists x)x^2+y^2=16$ с помощью кванторов общности и существования. Этот предикат не является тождественно истинным (т.к. например при y=5 получаем ложное высказывание $(\exists x)x^2+25=16$). Поэтому применение к нему квантора общности приводит к ложному высказыванию. Таково третье высказывание. Этот же предикат $(\exists x)x^2+y^2=16$ не является тождественно ложным (т.к. ,например, при y=0 он превращается в истинное высказывание $(\exists x)x^2+0=16$). Поэтому применение к нему квантора существования приводит к истинному высказыванию. Таково последнее высказывание. Итак, из четырех высказываний последнее истинно, а остальные ложны.

Тема 4.1 Основные элементы комбинаторики

Занятие 6 Решение комбинаторных задач

1 Теоретическая часть

Согласно классическому определению подсчет вероятности события *A* сводится к подсчету числа благоприятствующих ему исходов. Делают это обычно комбинаторными методами.

Комбинаторика — раздел математики, в котором изучаются задачи выбора элементов из заданного множества и расположения их в группы по заданным правилам, в частности задачи о подсчете числа комбинаций (выборок), получаемых из элементов заданного конечного множества. В каждой из них требуется подсчитать число возможных вариантов осуществления некоторого действия, ответить на вопрос «сколькими способами?».

Многие комбинаторные задачи могут быть решены с помощью следующих двух важных правил, называемых соответственно *правилами умножения и сложения.*

Правило умножения (основной принцип): если из некоторого конечного множества первый объект (элемент *x)* можно выбрать n_1 способами и после каждого такого выбора второй объект (элемент *y)* можно выбрать n_2 способами, то оба объекта *(x* и *y)* в указанном порядке можно выбрать $n_1 * n_2$ способами.

Этот принцип, очевидно, распространяется на случай трех и более объектов.

2 Практическая часть

Примеры вычисления вероятностей

Пример 1. В урне находятся 12 белых и 8 черных шаров. Найти вероятность того, что среди наугад вынутых 5 шаров 3 будут черными?

Решение: Выбрать 5 шаров из 20 можно C_{20}^{5} различными способами (все выборки — неупорядоченные подмножества, состоящие из 5 элементов), т. е. $n = C_{20}^{5}$ Определим число случаев, благоприятствующих событию *B* — «среди 5 вынутых шаров 3 будут черными». Число способов выбрать 3 черных шара из 8, находящихся в урне, равно C_8^3. Каждому такому выбору соответствует C_{12}^{2} способов выбора 2-х белых шаров из 12 белых в урне. Следовательно, по основному правилу комбинаторики (правилу умножения), имеем: $m = C_8^3 * C_{12}^{2}$. По формуле (1.3) находим, что

$$P\{B\} = \frac{C_8^3 * C_{12}^{2}}{C_{20}^{5}}$$

Пример 2. В коробке 5 синих, 4 красных и 3 зеленых карандаша. Наудачу вынимают 3 карандаша. Какова вероятность того, что: а) все они одного цвета; б) все они разных цветов; в) среди них 2 синих и 1 зеленый карандаш.

Решение: Сначала заметим, что число способов выбрать 3 карандаша из 12 имеющихся в наличии равно $n = C_{12}^{3} = 220$.

а) Выбрать 3 синих карандаша из 5 можно C_5^3 способами 3 красных из имеющихся 4 можно выбрать C_4^3 способами; 3 зеленых из 3 зеленых — C_3^3 способами.

По правилу сложения общее число m случаев, благоприятствующих событию A = {три карандаша, вынутых из коробки, одного цвета}, равно $m = C_5^3 + C_4^3 + C_3^3 = 15$. Отсюда $P(A) = \frac{m}{n} = \frac{15}{220} = \frac{3}{44}$

б) Пусть событие B — {три вынутых карандаша разных цветов}. Число m исходов, благоприятствующих наступлению события B, по правилу умножения равно $m. = C_5^1 * C_4^1 * C_3^1 = 5 \cdot 4 \cdot 3 = 60$. Поэтому $P(B) = \frac{m}{n} = \frac{60}{220} = \frac{3}{11}$

в) Пусть событие C = {из трех выбранных карандашей 2 синих и 1 зеленый}. Выбрать 2 синих карандаша из имеющихся 5 синих можно C_5^2 способами, а 1 зеленый из имеющихся 3 зеленых — C_3^1 способами. Отсюда по правилу умножения имеем: $m = C_5^2 \cdot C_3^1 = 30$. Поэтому $P(C) = \frac{m}{n} = \frac{30}{220} = \frac{3}{22}$

Пример 3. Дано шесть карточек с буквами Н, М, И, Я, Л, О. Найти вероятность того, что: а) получится слово ЛОМ, если наугад одна за другой выбираются три карточки; б) получится слово МОЛНИЯ, если наугад одна за другой выбираются шесть карточек и располагаются в ряд в порядке появления.

Решение: а) Из шести данных букв можно составить $n = A_6^3 = 120$ трехбуквенных «слов» (НИЛ, ОЛЯ, ОНИ, ЛЯМ, МИЛ и т.д.). Слово ЛОМ при этом появится лишь один раз, т.е. n = 1. Поэтому вероятность появления слова ЛОМ (событие A) равна $P(A) = \frac{m}{n} = \frac{1}{120}$

б) Шестибуквенные «слова» отличаются друг от друга лишь порядком расположения букв (НОЛМИЯ, ЯНОЛИМ. ОЛНИЯМ и т.д.). Их число равно числу перестановок из 6 букв, т. е. $n = P_6 = 6!$. Очевидно, что $m = 1$. Тогда вероятность появления слова МОЛНИЯ (событие B) равна $P(B) = \frac{m}{n} = \frac{1}{6!} = \frac{1}{120}$

Пример 4 В почтовом отделении имеются открытки 6 видов. Какова вероятность того, что среди *4* проданных открыток все открытки: а) одинаковы, б) различны?

Решение: Выбрать 4 открытки 6 видов можно $\bar{C}_6^4 = 126$ способами, т. е n=126.

а)Пусть событие A = {продано 4 одинаковые' открытки}. Число m исходов, благоприятствующих наступлению события А, равно числу видов открыток, т. е. $m = 6$. Поэтому $P(A) = \frac{6}{126} = \frac{1}{21}$

б) Пусть событие B — {проданы 4 различные открытки}. Выбрать 4 открытки из 6 можно $C_6^4 = 15$ способами, т. е. $m = 15$. Следовательно, $P(B) = \frac{`5}{126} = \frac{5}{42}$

3. Задачи для самостоятельного решения

1. В лифт 9-этажного дома вошли 4 человека. Каждый из них независимо друг от друга может выйти на любом (начиная со второго) этаже. Какова вероятность того, что все вышли: а) на_разных этажах; б) на одном этаже: в) на 5 этаже?
2. Из колоды карт (их 36) вытаскивают наудачу 5 карт. Какова вероятность того, что будут вытащены 2 туза и 3 шестерки
3. Семь человек рассаживаются наудачу на скамейке. Какова вероятность того, что два определенных человека будут сидеть рядом?
4. На 5 карточках разрезной азбуки изображены буквы Е, Е, Л, П, П. Ребенок случайным образом выкладывает их в ряд. Какова вероятность того, что у него получится слово ПЕПЕЛ?

Занятие 7. Размещения с повторениями

1 Теоретическая часть

Если при выборке m элементов из n элементы возвращаются обратно и упорядочиваются, то говорят, что это *размещения с повторениями.*

Размещения с повторениями могут отличаться друг от друга элементами, их порядком и количеством повторений элементов. Число всех размещений из n элементов по m с повторениями обозначается символом $\bar{A}_n^m$ и вычисляется по формуле

$$\bar{A}_n^m = n^m \qquad (1.12)$$

Пример 1. Из 3 элементов а, 6, *с* составить все размещения по два элемента с повторениями.

Решение: По формуле (1.12) число размещений по два с повторениями равно $\bar{A}_3^2 = 3^2 = 9$. Это: (a,a), (a,b), *(a,c),* (b,b), (b,a), *(b*,c), (c,c), (c,a), (c.b).

Пример 2. Сколько пятизначных чисел можно составить, используя цифры: а) 2, 5, 7, 8; б) 0, 1, 9?

Решение: а) Все пятизначные числа, составленные из цифр 2, 5, 7, 8, отличаются друг от друга либо порядком их следования (например, 25558 и 52855), либо самими цифрами (например, 52788 и 78888). Следовательно, они являются размещениями из 4 элементов по 5 с повторениями, т.е. $\bar{A}_n^m$. Таким образом, искомое число пятизначных

чисел равно $\overline{A}_4^5 = 4^5 = 1024$. Этот же результат можно получить, используя правило умножения: первую цифру слева в пятизначном числе можно выбрать четырьмя способами, вторую — тоже четырьмя способами, третью — четырьмя, четвертую — четырьмя, пятую — четырьмя. Всего получается 4 • 4 • 4 • 4 • 4 = 1024 пятизначных чисел.

б) Если пятизначные числа состоят из цифр 0, 1,9, то первую цифру слева можно выбрать двумя способами (0 не может занимать первую позицию), каждую из оставшихся четырех цифр можно выбрать тремя способами. Согласно правилу умножения, таких чисел будет 2 • 3 • 3 • 3 • 3 = 162.

2 Практическая часть

1.Из 60 вопросов, входящих в экзаменационные билеты, студент знает 50. Найти вероятность того, что среди 3-х наугад выбранных вопросов студент знает: а) все вопросы; б) два вопроса.

2. В барабане револьвера 7 гнёзд, из них в 5 заложены патроны. Барабан приводится во вращение, потом нажимается спусковой курок. Какова вероятность того, что, повторив такой опыт 2 раза подряд: а) оба раза не выстрелит; б) оба раза револьвер выстрелит?

3. Для проведения соревнования 10 команд, среди которых 3 лидера, путем жеребьевки распределяются на 2 группы по 5 команд в каждой. Какова вероятность того, что 2 лидера попадут в одну группу, 1 лидер — в другую?

4.Из колоды карт (их 36) наугад вынимают 2 карты. Найти вероятность, что среди них окажется хотя бы одна «дама».

3. Задачи для самостоятельного решения

1. Сколько различных «слов», состоящих из трех букв, можно образовать из букв слова БУРАН? А если «слова» содержат не менее трех букв?

2. Сколькими способами можно выбрать один цветок из корзины, в которой имеется 12 гвоздик, 15 роз и 7 хризантем?

3. Группа студентов изучает 10 различных дисциплин. Сколькими способами можно составить расписание занятий в понедельник, если в этот день должно быть 4 разных занятия?

4. Из 10 мальчиков и 10 девочек спортивного класса для участия в эстафете надо составить три команды, каждая из которых состоит из мальчика и девочки. Сколькими способами это можно сделать?

5. Сколько можно составить четырехзначных чисел так, чтобы любые две соседние цифры были различны?

Занятие 8. Сочетанияс повторениями

1 Теоретическая часть

Если при выборке *m* элементов из *n* элементы возвращаются обратно без последующего упорядочивания, то говорят, что это *сочетания с повторениями.*

Число всех сочетаний из *n* элементов по m с повторениями обозначается символом $\bar{C}_n^m$ и вычисляется по формуле

$$\bar{C}_n^m = C_{n+m-1}^m. \qquad (1.13)$$

Пример 1. Из трех элементов *a,* 6, с составить все сочетания по два элемента с

По формуле (1.13) число сочетаний по два с повторениями равно $\bar{C}_3^2 = C_{3+2-1}^2 = C_4^2 = \frac{4 \cdot 3}{1 \cdot 2} = 6$. Составляем эти сочетания с повторениями: (a,a), (a,b), (a,c), (b,b), (b,c), (c,c). ●

Пример 2. Сколькими способами можно составить букет из 5 цветов, если в наличии есть цветы трех сортов?

Решение: Рассматриваемое множество состоит из трех различных элементов, а выборки имеют объем, равный 5. Поскольку порядок расположения цветов в букете не играет роли, то искомое число букетов равно числу сочетаний с повторениями из трех элементов по 5 в каждом. По формуле **(1.13)** имеем $\bar{C}_3^5 = C_7^5 = C_7^{7-5} = C_7^2 = \frac{7 * 6}{1 * 2} = 21$

2 Практическая часть

1. В электричке 12 вагонов. Сколько существует способов размещения 4 пассажиров, если в одном вагоне .должно быть не более одного пассажира?
2. Сколькими способами 3 награды могут быть распределены между 10 участниками соревнования?
3. Из 4 первокурсников, 5 второкурсников и 6 третьекурсников надо выбрать 3 студента на конференцию. Сколькими способами можно осуществить этот выбор, если среди выбранных должны быть студенты разных курсов?
4. Сколькими способами можно расставить на полке 7 различных книг, чтобы определенные три книги стояли рядом? Не рядом?

3. Задачи для самостоятельного решения

1. Сколькими способами можно рассадить 5 человек за круглым столом? (Рассматривается только расположение сидящих относительно друг друга.)
2. 10 студентов, среди которых С. Федин и А. Шилов, случайным образом занимают очередь в библиотеку. Сколько имеется вариантов расстановки студентов, когда между Фединым и Шиловым окажутся 6 студентов?
3. У одного школьника имеется 7 различных книг для обмена, а у другого — 16. Сколькими способами они могут осуществить обмен: книга на книгу? Две книги на две книги?

Занятие 9. Перестановки с повторениями

1 Теоретическая часть

Пусть в множестве с n элементами есть k различных элементов, при этом 1-й элемент повторяется n_1 раз, 2-й элемент — n_2 раз .... *к-й* элемент — n_k-раз, причем $n_1 + n_2 + ... + n_k = n$.

Перестановки из n элементов данного множества называют *перестановками с повторениями из n элементов.*

Число перестановок с повторениями из n элементов обозначается символом $P_n(n_1,n_2, \bullet - -,n_k)$ и вычисляется по формуле

$$P_n(n_1, n_2, \ldots, n_k) = \frac{n!}{n_1!n_2! \ldots n_k!}. \quad (1.14)$$

Пример 1. Сколько различных пятизначных чисел можно составить из цифр 3, 5, 5, 8?

Решение: Применим формулу (1.14). Здесь n = 5, n_1 =2, $n2$=2, $n_з$ = 1- Число различных пятизначных чисел, содержащих' цифры 3.5 и 8 равно

$$P_5(2,2,1) = \frac{5!}{2!*2!*1!} = 30$$

2 Практическая часть

1. В урне 12 белых и 8 черных шаров. Сколькими способами можно выбрать 5 шаров, чтобы среди них было: а) 5 черных; б) 3 белых и 2 черных?
2. Сколькими способами можно распределить 15 выпускников по трем районам, если в одном из них имеется 8, в другом - 5 и в третьем -2 вакантных места?
3. Известно, что 7 студентов сдали экзамен по теории вероятностей на хорошо и отлично. Сколькими способами могли быть поставлены им оценки?
4. Игральная кость (на ее 6 гранях нанесены цифры от 1 до 6) бросается 3 раза. Сколько существует вариантов выпадения очков в данном опыте? Напишите некоторые из них.
5. Сколькими способами можно распределить 6 различных подарков между четырьмя ребятишками

3. Задачи для самостоятельного решения

1. Сколькими способами можно составить набор из 6 пирожных, если имеется 4 сорта пирожных?
2. Группа учащихся из 8 человек отправляется в путешествие по Крыму. Сколькими способами можно составить группу из учащихся 5-7 классов?
3. Сколькими способами можно распределить 4 книги на трех полках книжного шкафа? Найти число способов расстановки книг на полках, если порядок их расположения на полке имеет значение.
4. Сколько «слов» можно получить, переставляя буквы в слове: а) ГОРА: б) ИНСТИТУТ?
5. Сколько существует способов размещения 9 человек в двухместный, трехместный и четырехместный номера гостиницы?
6. Сколькими способами можно распределить 16 видов товаров_по трем

магазинам, если в 1-й магазин надо доставить 9, во 2-й — 4, а в третий — 3 вида товаров?

Тема 5.1 Основы теории графов

Занятие 10 Запись матриц инцидентности, смежности, достижимости

1 Теоретическая часть

Матрица смежности. Если вершины неориентированного графа *G* помечены метками v_1, v_2, ..., v_n, то элементы матрицы смежности *A(G)* размера *N*x*N* определяются следующим образом: ***A(i,j)* = 1**, если v_i смежна с v_j; ***A(i,j)* = 0** в противном случае.

Для неориентированного графа, изображенного на рисунке 8, матрица инцидентности имеет вид, представленный на рисунке 9 а.

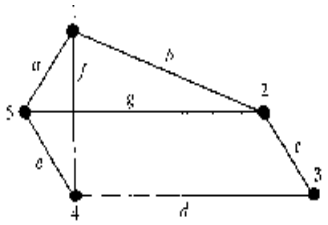


Рисунок 8 – Неориентированный граф

A(G):

	1	2	3	4	5
1	0	1	0	1	1
2	1	0	1	0	1
3	0	1	0	1	0
4	1	0	1	0	1
5	1	1	0	1	0

I(G):

	a	b	c	d	e	f	g
1	1	1	0	0	0	1	0
2	0	1	1	0	0	0	1
3	0	0	1	1	0	0	0
4	0	0	0	1	1	1	0
5	1	0	0	0	1	0	1

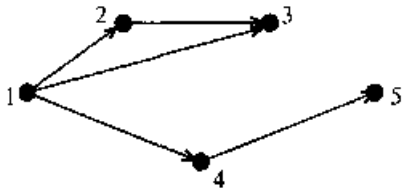
Рисунок 9, а - Матрица смежности *Рисунок 4, б* - Матрица инцидентности

Для ориентированного графа *G* - ***A(i,j)* = 1**, если дуга направлена от вершины *vi* к вершине *vj;* ***A(i,j)* = 0** в противном случае.

Матрица инцидентности. Если вершины неориентированного графа G помечены метками v*1, v2, ... vm,* а ребра - метками e1, e2, ..., *en,* то элементы **матрицы инцидентности I(G)** размера *M x N* определяются правилом: **I*(i,j)* = 1,** если *vi* инцидентна *ej;* ***I(i,j)* = 0** в противном случае (см. рис. 4. *б).*

Для ориентированного графа *G* -; ***I(i,j)* = 1**, если *vi* является *начальной* вершиной ребра ej ; ***I(i,j)* = -1**, если *vi* является *конечной* вершиной ребра ej; ***I(i,j)* = 0** в противном случае.

Для ориентированного графа *G,* имеющего *N* вершин, можно рассмотреть **матрицу достижимости** *C(G)* размера *N* x *N,* элементы которой определяются так: **C*(i,j)* = 1**, если вершина *vj* достижима из *vi;* ***C(i, j)* = 0** в противном случае. Ниже приведен пример ориентированного графа и его матрицы достижимости, рис. 5.



C(G):

	1	2	3	4	5
1	0	1	1	1	1
2	0	0	1	0	0
3	0	0	0	0	0
4	0	0	0	0	1
5	0	0	0	0	0

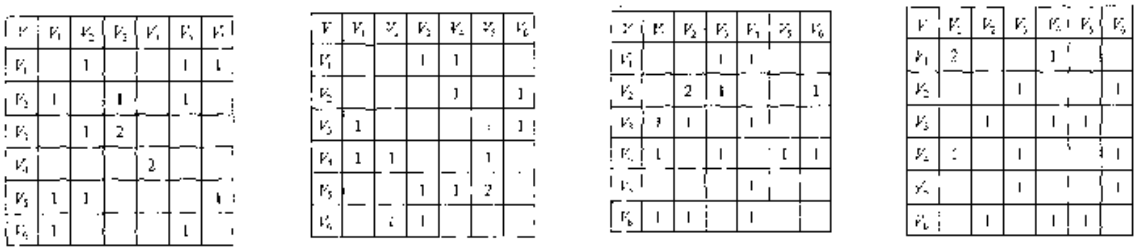
Рисунок 5 - Матрица достижимости ориентированного графа

2 Задания для самостоятельной работы

Задание 1

Орграф задан матрицей смежности.

a) Постройте изображение этого графа.
b) Укажите степени вершин графа.
c) По матрице смежности постройте матрицу инцидентности этого графа.
d) Постройте матрицу достижимости графа.
e) Содержит ли граф цикл? Простой цикл? Укажите его.
f) Укажите набольшую цепь графа.
g) Определите вид заданного графа.

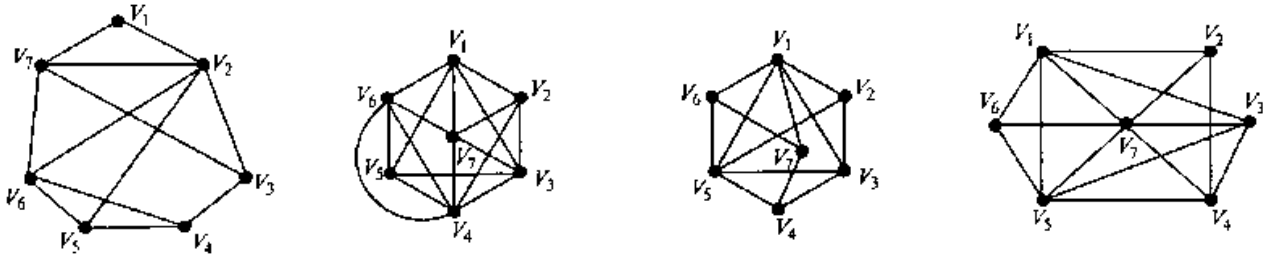


а) б) в) г)

Задание 2

Граф G задан диаграммой

a) Составьте для него матрицу смежности.
b) Постройте матрицу инцидентности.
c) Укажите степени вершин графа.
d) Постройте простой цикл, содержащий вершину V4.
e) Определите вид заданного графа.



а) б) в) г)

Занятие 11 Определение кратчайшего пути между вершинами графа методом Дейкстры графах

1 Теоретическая часть

Пусть дан граф $G = (X, \Gamma)$, дугам которого приписаны веса (стоимости), задаваемые матрицей $C = \| c_{ij} \|$. Задача о кратчайшем пути состоит в нахождении кратчайшего пути от заданной вершины $s \in X$ до заданной конечной

вершины $t \in X$, при условии, что такой путь существует, т.е. при условии $t \in R(s)$. Здесь $R(s)$ – множество вершин, достижимых из вершины s.

Элементы c_{ij} могут быть положительными, отрицательными или нулями. Единственное ограничение состоит в том, чтобы в G не было циклов с суммарным отрицательным весом.

Из общей задачи о кратчайшем пути вытекают две подзадачи:

а) для заданной начальной вершины s найти кратчайшие пути между s и всеми другими вершинами $x_i \in X$;

б) найти кратчайшие пути между всеми парами вершин.

Рассмотрим задачу определения кратчайшего пути между заданными вершинами s и t для случая $c_{ij} > 0, \forall\ i, j$. Наиболее эффективный алгоритм решения задачи о кратчайшем $(s - t)$ пути первоначально дал Дейкстра. В общем случае этот метод основан на приписывании вершинам временных пометок, причем пометка вершины дает верхнюю границу длины пути от s к этой вершине. Эти пометки (их величины) постепенно уменьшаются с помощью итерационной процедуры, и на каждом шаге итерации только одна из временных пометок становится постоянной. Последнее указывает на то, что пометка уже не является верхней границей, а дает точную длину кратчайшего пути от s к рассматриваемой вершине.

Алгоритм Дейкстры включает следующие основные шаги:

Пусть $l(x_i)$ – пометка вершины x_i.

Присвоение начальных значений

Шаг 1$_x$. Положить $l(s) = 0$ и считать эту пометку постоянной. Положить $l(x_i) = \infty$ для всех $x_i \neq s$ и считать эти пометки временными. Положить $p = s$.

Обновление пометок

Шаг 2. Для всех $x_i \in \Gamma(p)$, пометки которых временные, изменить пометки в соответствии со следующем выражением:

$$l(x_i) = \min \{ l(x_i);\ \ l(p) + c(p, x_i). \} \qquad (3.7)$$

Превращение пометки в постоянную

Шаг 3. Среди всех вершин с временными пометками найти такую, для которой:

$$l(x_i^*) = \min \{ l(x_i) \} .$$

Шаг 4. Считать пометку вершины x_i^* постоянной и положить $p = x_i^*$.

Шаг 5.

а) если надо найти лишь путь от s к t. Если $p = t$, то $l(p)$ является длиной кратчайшего пути. Останов. Если $p \neq t$, то переход к шагу 2 .

б) если требуется найти пути от s ко всем остальным вершинам. Если все вершины отмечены как постоянные, то эти пометки дают длины кратчайших путей. Останов. Если некоторые пометки являются временными, то переход к шагу 2.

Как только длины кратчайших путей от s будут найдены, сами пути можно получить при помощи рекурсивной процедуры с помощью соотношения

$$l\left(x_i'\right) + c\left(x_i', x_i\right) = l(x_i), \qquad (3.8)$$

где вершина x_i' непосредственно предшествует вершине x_i в кратчайшем пути от s к x_i.

Если кратчайший путь от s до любой вершины x_i является единственным, то дуги $\left(x_i', x_i\right)$ этого пути образуют ориентированное дерево с корнем s.

Если существует несколько кратчайших путей от s к какой либо другой вершине, то при некоторой фиксированной вершине x_i' соотношение (3.8) будет выполняться для более чем одной вершины x_i. В этом случае выбор может быть либо произвольным (если нужен хотя бы один кратчайший путь), либо таким, что рассматриваются все дуги $\left(x_i', x_i\right)$, входящие в какой-либо из кратчайших путей, и при этом совокупность всех таких дуг образует не ориентированное дерево, а общий граф, называемый базой относительно s или просто – s – базой.

Рассмотрим действие алгоритма на примере.

2. Практическая часть

Методика решения задач поиска кратчайшего пути алгоритмом Дейкстры

Пример 3.17. Рассмотрим граф G, представленный на рис. 3.12, где каждое неориентированное ребро рассматривается как пара противоположно ориентированных дуг равного веса. Матрицы смежностей весов C приведены ниже. Требуется найти все кратчайшие пути от вершины x_1 ко всем остальным вершинам. Для решения задачи воспользуемся алгоритмом Дейкстры.

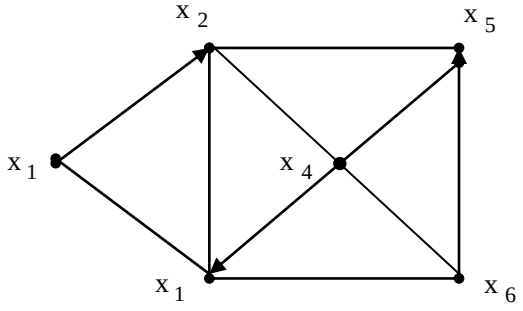


Рисунок 3.12 - Смешанный граф

i / j	x_1	x_2	x_3	x_4	x_5	x_6
x_1		10	6			
x_2			14	8	16	
x_3	6	14				9
x_4		8	7		4	13

x_5		16		4		
x_6			9	13	3	

Решение.

Постоянные пометки будем снабжать знаком $+$, остальные пометки рассматриваются как временные.

Шаг 1.

$$l(x_1) = 0^+, \quad l(x_i) = \infty, \forall x_i \neq x_1, p = x_1.$$

Первая итерация

Шаг 2.

$\Gamma(p) = \Gamma(x_1) = \{x_2, x_3\}$ – все пометки временные.

Применяя (3.7), получим

$$l(x_2) = \min\{\infty; 0+10\} = 10,$$
$$l(x_3) = \min\{\infty; 0+6\} = 6.$$

Шаг 3.

$$\min\{l(x_2); l(x_3)\} = \min\{10, 6\} = 6 = l(x_3^*).$$

Шаг 4.

$$l(x_3) = 6^+; \quad p = x_3.$$

Шаг 5.

Не все вершины имеют постоянные пометки.

Переход к следующей итерации.

Вторая итерация.

Шаг 2.

$$\Gamma(p) = \Gamma(x_3) = \{x_1, x_2, x_6\}.$$
$$l(x_2) = \min\{10; 6+14\} = 10.$$
$$l(x_6) = \min\{\infty; 6+9\} = 15.$$

Шаг 3.

$$\min\{l(x_2); l(x_6)\} = \min\{10, 15\} = 10.$$

Шаг 4.

$$l(x_2^*) = 10; \quad p = x_2.$$

Шаг 5. Переход к следующей итерации.

Третья итерация

Шаг 2.

$$\Gamma(p) = \Gamma(x_2) = \{x_3, x_4, x_5\}.$$
$$l(x_4) = \min\{\infty; 10+8\} = 18.$$
$$l(x_5) = \min\{\infty; 10+16\} = 26.$$

Шаг 3.

$$\min\{l(x_4); l(x_5)\} = \min\{18, 26\} = 18.$$

Шаг 4.

$$l(x_4^*) = 18; \quad p = x_4.$$

Шаг 5. Переход к следующей итерации.

Четвертая итерация

Шаг 2.

$$\Gamma(p) = \Gamma(x_4) = \{x_2, x_3, x_5, x_6\}.$$

Временные пометки имеют x_5 и x_6.

$$l(x_5) = \min\{26; 18 + 4\} = 22,$$

$$l(x_6) = \min\{15; 18 + 13\} = 15.$$

Шаг 3.

$$\min\{l(x_5); l(x_6)\} = \min\{22, 15\} = 15.$$

Шаг 4.

$$l(x_6^*) = 15; \quad p = x_6.$$

Шаг 5.

Переход к следующей итерации.

Пятая итерация

Шаг 2.

$$\Gamma(p) = \Gamma(x_6) = \{x_3, x_4, x_5\},$$

$$l(x_5) = \min\{22; 15 + 3\} = 18.$$

Шаг 3.

$$\min l(x_5) = 18.$$

Шаг 4.

$$l(x_5^*) = 18; \quad p = x_5.$$

Шаг 5. Все вершины имеют постоянные пометки. Останов. Окончательная расстановка пометок приведена на рис. 3.13.

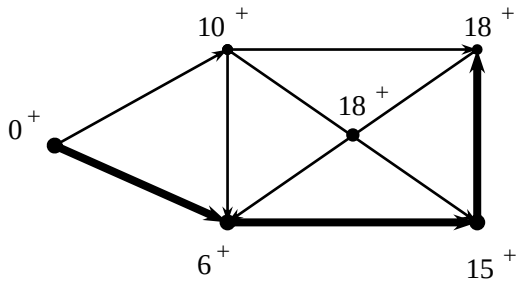


Рисунок 3.13 - Окончательные пометки вершин графа

Для нахождения кратчайшего пути между вершиной x_5 (например) и начальной вершиной последовательно применяем (3.8). Таким образом, полагая $x_i = x_5$, находим вершину x_5', непосредственно предшествующую x_5 в кратчайшем пути от x_1 к x_5; вершина x_5' должна удовлетворять соотношению

$$l(x_5') + C(x_5', x_5) = l(x_5) = 18.$$

Единственной такой вершиной является $x_6 = x_5'$.

Продолжая аналогичные рассуждения получим:

$$l\left(x_6'\right)+c\left(x_6', x_6\right)=l\left(x_6\right)=15 \Rightarrow x_6'=x_3.$$

$$l\left(x_3'\right)+c\left(x_3', x_3\right)=l\left(x_3\right)=6 \Rightarrow x_3'=x_1.$$

Таким образом, кратчайший путь от вершины x_7 к вершине x_5 проходит через вершины x_3 и x_6. То есть $\mu\left(x_1, x_5\right)=\left(x_7, x_3, x_6, x_5\right)$. Путь на рисунке 3.13 выделен чёрным цветом.

3 Задания для самостоятельной работы

1.. Методом Дейкстры определить кратчайшие пути между всеми парами вершин в нагруженном орграфе, заданном матрицей длин дуг.

$$\begin{bmatrix} 0 & 7 & 2 & 3 & - \\ - & 0 & - & 2 & - \\ 3 & - & 0 & 5 & 4 \\ - & - & 2 & 0 & 1 \\ - & 3 & 4 & - & 0 \end{bmatrix}$$

2. Методом Дейкстры определить кратчайшие пути между всеми парами вершин в нагруженном орграфе, заданном матрицей длин дуг.

$$\begin{bmatrix} 0 & 4 & - & - & 5 & 1 & - \\ 3 & 0 & 2 & 1 & - & - & - \\ 1 & 1 & 0 & - & - & - & 3 \\ - & 3 & 1 & 0 & 1 & - & - \\ - & - & 2 & - & 0 & 1 & 5 \\ - & 3 & - & 2 & 2 & 0 & - \\ - & - & 2 & - & - & 2 & 0 \end{bmatrix}$$

3. Методом Дейкстры определить кратчайшие пути между всеми парами вершин в нагруженном орграфе, заданном матрицей длин дуг.

$$\begin{bmatrix} 0 & 5 & - & 3 & 4 \\ 5 & 0 & 2 & - & 3 \\ - & 2 & 0 & 4 & 1 \\ 3 & - & 4 & 0 & 2 \\ 4 & 3 & 1 & 2 & 0 \end{bmatrix}$$

Тема 6.1 Абстрактные автоматы

Занятие 12 Алгебра событий

1 Теоретическая часть

Теория автоматов нуждается в специальном, формальном аппарате для описания наиболее абстрактных моделей автоматов, каковыми являются алфавитные операторы. Таким формальным инструментом является *алгебра событий* или *алгебра Клини*. Она была создана в 50-х годах как часть теории цифровых

машин, а затем развивалась применительно к контекстно-свободным языкам в общей теории алгоритмов. Алгебра событий основывается на представлении алфавитного оператора как распознавателя входных слов.

Представим себе множество слов во входном алфавите X, содержащее все без исключения слова этого вида. В него входит пустое слово, не содержащее ни одной буквы, бесконечные слова и даже запрещенные для данного автомата слова (если он частичный). Очевидно, множество входных слов бесконечно. Разобьем это множество на классы E_1, E_2, ..., число которых равно числу букв выходного алфавита Y. Условимся, что прием любого слова из класса E_1 приводит к появлению на выходе распознавателя буквы y_1, прием слова из E_2 - к появлению y_2 и т.д. Классы E_1, E_2, ... называются *событиями* (обозначение идет от английского event <iv'ent>). Используется выражение: "E_i - событие, представленное в автомате буквой y_i".

Если имеется некоторый автомат Мура, описанный отмеченным графом переходов, то выделив на графе все пути от начальной вершины до вершины, отмеченной буквой y_k. мы тем самым определяем событие E_k. Если одной и той же выходной буквой отмечено несколько вершин, то все пути до этих вершин должны войти в событие E_k. Для автомата Мили событие E_k тоже можно представить в виде множества путей на графе переходов, но теперь уже не до определенных состояний, а до дуг, отмеченных буквой y_k. Таким образом, смысл термина "событие" не соответствует его интуитивному пониманию: “событием” называется множество (возможно бесконечное) различных процессов, объединенных общим результатом.

Когда задается некоторый автомат, то тем самым однозначно задается совокупность событий *представимых в данном автомате*. И наоборот, событие называется *представимым в автомате*, если существует подходящий для этого автомат. Понятия автомата и представимого в нем события помогают выяснению проблемы алгоритмической разрешимости множеств. К настоящему времени доказано, что в автомате представимо любое событие, состоящее из конечного множества входных слов или даже из их бесконечного множества, если входные слова состоят из циклически повторяющегося начального отрезка. Доказано также, что существуют события, не представимые в автомате. Например, непериодическая бесконечная последовательность входных букв не представима, а значит и не распознаваема автоматом. Доказано также, что существуют события, не представимые в автомате, но которые могут быть алгоритмически разрешены другим путем, например с помощью машины Тьюринга. Парадоксально выглядит факт алгоритмической неразрешимости определения в общем виде свойства событий быть представимыми в автоматах.

Продолжим определение основных понятий алгебры событий, имея в виду только конечные события. Такие события можно задавать списками, например
$E_i = \{x_1, x_2x_1, x_1x_3\}$.

Событие, содержащее только одно однобуквенное слово, называется *элементарным событием*. К элементарным событиям относится также пустое событие, не содержащее ни одной буквы. Пустое событие будем обозначать буквой e. Списки, определяющие конечные события, не принято заключать в

фигурные скобки. Упрощая, таким образом, запись, не будем забывать, что расположение слов в списке остается произвольным

Если выходной алфавит $Y = \{y_1, y_2,...y_k\}$ имеет k букв, то все входные слова нужно распределить по k событиям, и каждое событие будет представлено своей буквой y. Если автомат является частичным, то все запрещенные слова должны составить запрещенное событие $E_{запр}$. Результат представим в табл.5.1.

Таблица 5.1

E	y
E_1	y_1
E_2	y_2
...	...
$E_{запр}$	-

Имея для каждого события определяющий его список входных слов, можно для каждого входного слова буква за буквой построить выходное слово, вовсе не зная функций перехода и выхода автомата, не имея понятия о числе его состояний. Зададимся входным словом

$S = x_2x_3x_4x_1x_3x_4$.

Начнем искать среди всех событий такое, которое содержит однобуквенное слово x_2. Найдя такое событие, мы сразу определим первую букву выходного слова. Затем ищем событие, содержащее двухбуквенное слово x_2x_3. Проведя аналогичные действия до конца входного слова, получим всё выходное слово.

Сказанное означает, что автомат-распознаватель легко превращается в автомат-преобразователь. Кроме того, можно заметить, что на этом уровне описания нет разницы между цифровым автоматом и некоторым контекстно-свободным языком программирования. Разумеется, поиск входных слов в списках практически невозможен, а если события бесконечны, то и сами списки невозможны. Алгебра событий как раз и предназначена для оперирования событиями без использования списков. В ней определены три операции:

-*объединение:* $E_1 \quad E_2$;

-*конкатенация* или *сцепление*: E_1E_2;

-*итерация* E*.

Запятая, разделяющая слова в списке, заменяется знаком дизъюнкции Ъ , скобки используются только для изменения порядка действий. Обычный порядок действий: вначале итерация, затем конкатенация и, наконец, объединение. Кроме трех основных операций можно использовать еще теоретико-множественное дополнение. Обычно используемый символ дополнения мы будем заменять на !*E,* во избежание неверной передачи его броузером.

Объединение ассоциативно и коммутативно. Имеется нейтральный элемент: пустое слово *e*. Противоположных элементов нет, следовательно, по объединению множество событий образует полугруппу (абелеву аддитивную полугруппу с нулем). Если

$$E_1=x_1x_2 \quad x_2\,x_3;\; E_2=x_1 \quad x_1\,x_3,$$

то объединение выразится так

$$E_1 \quad E_2 = x_1 \quad x_1x_2 \quad x_1x_3 \qquad x_2x_3.$$

Конкатенация ассоциативна, но не коммутативна. Конкатенация дистрибутивна относительно объединения, но пользоваться этим свойством нужно осторожно, помня о некоммутативности конкатенации:

$$E_1E_2 \quad E_1E_3 = E_1(E_2 \quad E_3), \text{ но}$$

$$E_1E_2 \quad E_3E_1 \quad E_1(E_2 \quad E_3).$$

Итерация E^* представляет объединение пустого слова e, события E и всех произведений вида EE, EEE, ... включая бесконечную конкатенацию событий E.

$$E^* = \qquad ...$$

Объединение, конкатенация и итерация позволяют компактно описывать огромное количество слов и событий, хотя не всякое событие поддается описанию этим путем.

Регулярным событием называется такое событие, которое может быть описано композицией конечного числа объединений, конкатенаций и итераций.

Любое конечное событие (т.е. конечное множество конечных слов) регулярно, так как его можно описать объединением слов, а слова - объединением букв. События, содержащие бесконечные слова или бесконечное множество слов, регулярны, если поддаются описанию с помощью итерации. Клини доказал, что в конечном автомате, как и в любой формальной алгоритмической системе, могут быть представлены только регулярные события.

Рассмотрим примеры использования алгебры событий для описания алфавитных операторов

Пример 1.

$X = \{x_1,x_2\}$; $Y = \{y_1,y_2,y_3\}$.

Словесное описание событий: серия входных сигналов, состоящая из трех букв x_1, приводит к y_1. Если после трех или большего числа букв x_1 подряд поступает x_2, то выдается y_2. В остальных случаях выдается y_3.

Определим событие E_1. Будем искать регулярное выражение в виде произведения двух событий: первого, содержащего все слова *без трех букв* x_1 *подряд и не оканчивающихся на* x_1, и второго, состоящего из одного слова $x_1x_1x_1$. Следовательно

$$E_1 = (x_2 \quad x_1x_2 \quad x_1x_1x_2)^* x_1x_1x_1.$$

В итерационных скобках - всего три члена, но множество описанных слов бесконечно. В частности итерация содержит и пустое слово, поэтому событию E_1 принадлежит и трехбуквенное слово $x_1x_1x_1$.

В словесном задании алфавитного оператора было сказано, что если после трех или большего числа букв x_1 появится x_2, то должна быть выдана буква y_2.

Соответствующее регулярное выражение получим из предыдущего, умножив его справа на x_1* и на x_2:

$$E_2 = (x_2 \quad x_1x_2 \quad x_1x_1x_2)^* x_1x_1x_1x_1^*x_2.$$

Так как итерация содержит e, то простейшим элементом E_2 будет $x_1x_1x_1x_2$.
Все слова, не вошедшие в E_1 и E_2, составят E_3:

$$E_3 = !(E_1 \quad E_2).$$

Пример 2. Опишем регулярным выражением работу последовательного двоичного сумматора с внутренней памятью переноса. Входной абстрактный алфавит должен состоять из четырех букв, в соответствии с четырьмя возможными комбинациями двоичных цифр слагаемых
$X = \{x_1, x_2, x_3, x_4\} = \{00, 01, 10, 11\}$.
Выходной алфавит должен состоять всего из двух букв
$Y = \{y_1, y_2\} = \{0, 1\}$.
Внутренний алфавит как таковой не существует, хотя при рассуждениях мы должны учитывать состояние памяти переноса.
Определим событие E_2, приводящие к y_2. Это событие включает два случая. В первом случае переноса нет, а на вход поступает x_2 или x_3. Во втором случае имеется единица переноса, а на вход поступает x_1 или x_4. Опишем первый случай. К запоминанию единицы переноса не приводят однобуквенные слова x_1, x_2 и x_3. Событиями, не приводящими к записи единицы переноса, будут также

$$x_1 \quad x_2 \quad x_3 \quad \text{и} \quad (x_1 \quad x_2 \quad x_3)^*.$$

Последнее из них представляет интерес как наиболее общее. Кроме того, перенос равен нулю, если после любой последовательности слов на вход автомата поступает x_1. Это выражается формулой

$$(x_1 \quad x_2 \quad x_3 \quad x_4)^*x_1$$

Множество всех представимых в автомате входных слов, описанное в данной формуле с помощью итерации над списком всех букв алфавита, называется *всеобщим событием.* Объединяя полученные выражения и умножая их справа на x_2 или x_3 получим:
$E_2 \quad x_1 \quad x_2 \quad x_3 \quad x_1 \quad x_2 \quad x_3 \quad x_4)^*x_1)\,(x_2 \quad x_3)$ (5.1)

E_2, описываются формулой (5.1). Второй случай подразумевает наличие единицы в памяти переносов. Перенос должен остаться после получения любого слова из события
$(x_1 \quad x_2 \quad x_3 \quad x_4)^*x_4\,(x_2 \quad x_3 \quad x_4)^*$
Умножая это выражение справа на $x_1 \quad x_4$ получим
E_2 [redacted] $x_2 \quad x_3 \quad x_4)^*x_4\,(x_2 \quad x_3 \quad x_4)^*\,(x_1\,x_2)..$ (5.2)

Объединяя (5.1) и (5.2) получим регулярное выражение для события E_2:

$$E_2 = ((x_1 \quad x_2 \quad x_3 \qquad x_1 \quad x_2 \quad x_3 \quad x_4)^* x_1)\,(x_2 \quad x_3$$

$$\blacksquare \quad x_2 \quad x_3 \quad x_4)^* x_4\,(x_2 \quad x_3 \quad x_4)^*(x_1 \quad x_4).$$

Приведенные примеры поясняют путь составления регулярных выражений на основе произвольных словесных определений событий.

Следующей задачей синтеза является построение функций перехода и выхода абстрактного автомата. Так как функции автомата Мура проще, можно синтезировать сперва автомат Мура, а затем по уже известному нам алгоритму преобразовать его (при необходимости) в автомат Мили. В любом случае в конце синтеза должна быть решена задача минимизации числа состояний.

Рассмотрим методику синтеза сперва в очень упрощенном виде. Допустим, что задано конечное множество событий, представленных в автомате соответствующими выходными буквами. Возьмем первое слово события E_1 и условимся, что первая буква этого слова переводит автомат из состояния a_0 в a_1. Вторая буква того же слова переводит автомат из a_1 в a_2 и т.д. до конца первого слова.

Затем берем второе входное слово из E_1. Если оно начинается с той же буквы x, что и первое, то и здесь первый переход определится как переход из a_0 в a_1. Вообще все одинаковые начальные сегменты слов должны преобразовываться в одинаковые цепочки переходов. С момента появления несовпадающих букв x определяются переходы в новые, ранее не занятые состояния.

Например: первое слово x_1x_3, второе - $x_1x_2x_3$. Первому слову соответствует цепочка $a_0\,a_1\,a_2$, а второму - $a_0\,a_1\,a_3\,a_4$.

Аналогично устанавливаются последовательности переходов для всех слов события E_1. Последние состояния каждой из последовательностей отмечаются буквой y_1 (это соответствует подходу к автомату, как к распознавателю - распознавание совершается в момент получения последней буквы события).

Аналогично обрабатываются все слова события E_2 и т.д. Одинаковым начальным сегментам слов, даже принадлежащих разным событиям, должны соответствовать одинаковые цепочки переходов. Нарушение этого условия привело бы к недетерминированности некоторых переходов.

Когда все слова всех событий обработаны указанным способом, т.е. установлена связь между входными сигналами, состояниями автомата и выходными сигналами, остается только выразить эту связь в виде отмеченной таблицы переходов автомата Мура.

2 Практическая часть

Пример 3. $X = \{x_1, x_2, x_3\}$; $Y = \{y_1, y_2, y_3\}$;

$$E_1 = x_2\,x_1 \quad x_1x_3;\ E_2 = x_2x_3 \quad x_3x_1 \quad x_3x_3;\ E_{\text{запр}} = x_2x_2x_2;\ E_3 = !(E_1 \quad E_2 \quad E_{\text{запр}})$$

Цепочки переходов будем определять только для E_1 и E_2. Событие E_3 определено как дополнение, поэтому соответствующие цепочки определятся сами собой.

Что касается запрещенного события, то поведение автомата на этом событии может быть доопределено на одном из последующих этапов синтеза, например при минимизации числа состояний. Можно подойти к этому вопросу иначе, упростив автомат с самого начала. Для этого достаточно включить запрещенное событие в какое-нибудь другое в качестве его подмножества. В данном примере примем $E_{\text{запр}}$ МE_3.

Чтобы облегчить составление цепочек переходов проделаем *разметку мест* регулярных выражений E_1 и E_2.

Местом регулярного выражения называется промежуток между буквами, между буквой и скобкой, между буквой и знаком Ъ , а также в начале и в конце регулярного выражения, т.е. перед первой и после последней буквы. Места обозначаются вертикальными штрихами:

$$E_1 = \underset{0}{|}x_2\underset{1}{|}x_1\underset{2}{|}\vee\underset{0}{|}x_1\underset{3}{|}x_3\underset{4}{|};$$

$$E_2 = \underset{0}{|}x_2\underset{1}{|}x_3\underset{5}{|}\vee\underset{0}{|}x_3\underset{6}{|}x_1\underset{7}{|}\vee\underset{0}{|}x_3\underset{6}{|}x_3\underset{8}{|}$$

Под штрихами записываем индексы состояний автомата в соответствии с вышеописанными правилами. Закончив разметку мест приступаем к составлению таблицы переходов автомата.

Всего должно получиться девять состояний - от a_0 до a_8. Готовим бланк таблицы автомата Мура с девятью состояниями и тремя входными буквами (табл. 5.2). Через косую черту делаем отметки выходных букв: пишем y_1, если состояние соответствует концу слова из E_1, y_2 - если это конец слова из E_2, и y_3 - во всех остальных случаях.

Таблица 5.2

	x_1	x_2	x_3
a_0.-	a_3	a_1	a_6
a_1/y_2	a_2		a_5
a_2/y_1			
a_3/y_3			a_4
a_4/y_1			
a_5/y_2			
a_6/y_3	a_7		a_8
a_7/y_2			
a_8/y_2			

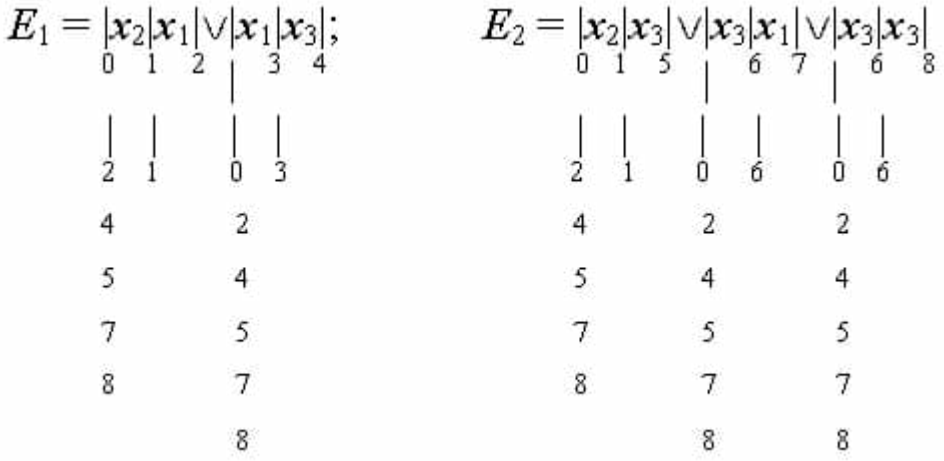
Клетки таблицы заполняем в соответствии с разметкой мест. Сделав все записи, вытекающие из разметки мест, мы видим, что большинство клеток таблицы пусто. Это произошло по двум причинам. Первая состоит в том, что мы не задали переходы, соответствующие событию E_3. Чтобы исправить это упущение, было бы достаточно просто вписать в свободные клетки состояния a_1, a_3 или a_6, отмеченные буквой y_3. Вторая, более серьезная причина, состоит в том, что мы синтезировали автомат однократного действия. Перед началом работы он

находится в состоянии a_0, верно обрабатывает первое поступившее в него слово, после чего в a_0 не возвращается. Реакция автомата на все последующие слова не определена.

Чтобы получить автомат многократного действия, необходимо усовершенствовать разметку мест. Будем различать две категории мест: основные и предосновные места.

Основные места, это - начальное место регулярного выражения и места справа от буквы. *Предосновные места*, это - *все места слева от буквы.* По этому определению некоторые места будут одновременно основными и предосновными.

Основные места размечаются как было показано раньше, а предосновные - по особым правилам, называемым *правилами подчинения мест*. Формулировку этих правил дадим позже, а сейчас покажем, как это делается, взяв тот же пример



Предосновные места выделяем, удлинив соответствующие штрихи. Эти места отмечаются индексами состояний, в которых автомат находится *перед* приемом данной буквы.

Например, автомат находится в состоянии a_0. Какие буквы могут поступать при этом на вход автомата, чтобы привести к событию E_1? Это могут быть первые буквы слов из события E_1, т.е. x_2 в первом слове или x_1 - во втором. Стало быть, отметка 0 должна быть сделана перед первым словом (основное место) и перед вторым (предосновное место). Аналогично располагаются отметки 0 в E_2. В итоге получился ряд предосновных мест, подчиненных основному месту с отметкой 0.

Какие предосновные места должны быть подчинены основному месту с индексом 1? В данном примере это соответствует срединам первых слов событий E_1 и E_2, т.е. здесь предосновные места одновременно являются и основными и подчинены сами себе.

Основному месту с индексом 2 подчинены все те предосновные места, которые раньше получили отметку 0, так как закончив обработку слова x_2x_1 автомат должен быть готов к приему новых слов, как если бы он находился в состоянии a_0. По этой же причине подчиним все предосновные места, ранее получившие метку 0, основным местам 4, 5, 7 и 8, соответствующим концам слов обоих событий.

Подправив, таким образом, методику синтеза, повторим попытку составления таблицы переходов автомата Мура. На основании новой разметки мест получается таблица 5.3.

Таблица 5.3

	x_1	x_2	x_3
a_0/-	a_3	a_1	a_6
a_1/y_3	a_2		a_5
a_2/y_1	a_3	a_1	a_6
a_3y_3			a_4
a_4/y_1	a_3	a_1	a_6
a_5/y_2	a_3	a_1	a_6
a_6/y_3	a_7		a_8
a_7/y_2	a_3	a_1	a_6
a_8/y_2	a_3	a_1	a_6

Четыре клетки таблицы остались свободными. Они соответствуют словам события E_3. Чтобы обеспечить правильную многократную работу автомата, в эти клетки нужно вписать какое-нибудь состояние, отмеченное буквой y_3 и определить переходы из него точно так же, как из a_0. Это условие проще всего обеспечивается добавлением ещё одного состояния a_9 и расширением таблицы на одну строку. Результат представлен в табл. 5.4.

Таблица 5.4

	x_1	x_2	x_3
a_0/-	a_3	a_1	a_6
a_1/y_3	a_2	a_9	a_5
a_2/y_1	a_3	a_1	a_6
a_3y_3	a_9	a_9	a_4
a_4/y_1	a_3	a_1	a_6
a_5/y_2	a_3	a_1	a_6
a_6/y_3	a_7	a_9	a_8
a_7/y_2	a_3	a_1	a_6
a_8/y_2	a_3	a_1	a_6
a_9/y_3	a_3	a_1	a_6

Основная часть задачи синтеза в данном примере завершена. Остается провести минимизацию числа состояний полученного автомата, как полностью определенного автомата Мура. От неопределенности значения выхода при начальном состоянии автомата нужно избавиться, доопределив функцию выхода наиболее целесообразным способом. В данном примере следует принять l (a_0) = y_3.

Проведя минимизацию, получим автомат с шестью состояниями (табл.5.5).

Таблица 5.5

	x_1	x_2	x_3
a_0/y_3	a_2	a_1	a_3

a_1/y_3	a_4	a_0	a_5
a_2/y_3	a_0	a_0	a_4
a_3/y_3	a_5	a_0	a_5
a_4/y_1	a_2	a_1	a_3
a_5/y_2	a_2	a_1	a_3

С помощью этой таблицы можно убедиться, что при поступлении на вход автомата произвольных слов он распознает их в соответствии с заданием, или, иначе говоря, правильно преобразует в выходные.

Допустим, что принято слово $x_1x_3x_1$ и преобразовано (см. табл. 5.5) в слово $(y_3)y_3y_1y_3$, первая буква которого не является откликом на входное слово, и поэтому записана в скобках. Буква y_1 является откликом на слово x_1x_3. Вторая и третья буквы входного слова совпадают с одним из слов события E_2. Но событие E_2 не состоится, так как автомат воспринимает букву x_3 не как начало нового слова, а как конец предыдущего. Всё это полностью соответствует требуемому закону функционирования автомата.

Пример 4. Возьмем регулярное выражение общего вида, со скобками и с использованием итераций.

$X = \{x_1, x_2\}$; $Y = \{y_1, y_2\}$; $E_1 = (x_1 \quad {}_2x_1)^*x_2x_2(x_1)^*x_2$; $E_2 = !E_1$.

Разметим места регулярного выражения E_1. при этом будем иметь в виду, что в начальном состоянии автомат может находиться в следующих случаях:

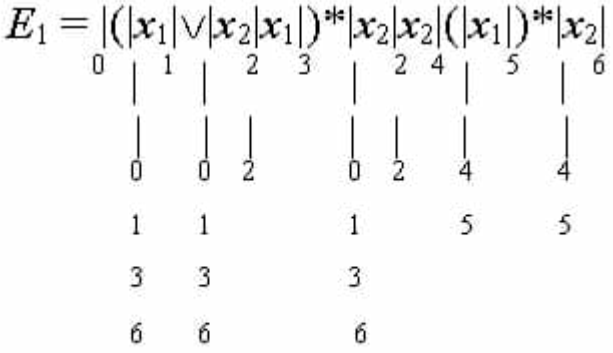
-перед началом работы,

-перед приемом однобуквенного слова x_1 ,

-перед приемом первой буквы слова x_2x_1,

-перед приемом буквы x_2 после итерационных скобок (в случае пустой итерации).

Во всех остальных случаях автомат уже не будет находиться в состоянии a_0



В состоянии a_1 автомат находится после получения слова x_1 в итерационных скобках и находясь в этом состоянии должен далее работать так же, как если бы он начинал работу из состояния a_0. Поэтому отметку 1 делаем у всех предосновных мест, ранее отмеченных отметкой 0.

Отметка 2 делается только в двух местах, которые оказываются одновременно основными и предосновными - ведь начав отработку слова, начинающегося с x_2,

автомат уже не может находиться ни в каком другом состоянии, кроме a_2. Отметки 3 и 6 присваиваются предосновным местам, которые уже получили отметку 0 и т.д.

Закончив разметку мест, составляем таблицу переходов и выходов (табл.5.6).

Таблица 5.6

	x_1	x_2
a_0/-	a_1	a_2
a_1/y_2	a_1	a_2
a_2/y_2	a_3	a_4
a_3/y_2	a_1	a_2
a_4/y_2	a_5	a_6
a_5/y_2	a_5	a_6
a_6/y_1	a_1	a_2

Минимизация дает автомат с четырьмя состояниями (табл. 5.7).

Таблица 5.7

	x_1	x_2
a_0/y_2	a_0	a_1
a_1/y_2	a_0	a_2
a_2/y_2	a_2	a_3
a_3/y_1	a_0	a_1

В заключение этой темы запишем формальные правила подчинения мест в регулярных выражениях общего вида:

-индексы мест, находящихся между буквами, на другие места не распространяются,

-индекс места перед любыми скобками распространяется на места перед словами в скобках. Если скобки - итерационные, то индекс места перед ними распространяется и на место после них,

-индексы мест после слов в скобках распространяются на место после этих скобок. Если скобки - итерационные, то эти места распространяются на места перед всеми словами в этих скобках,

-индекс конечного места распространяется на все те места, на которые распространяется индекс 0.

Занятие 13 Абстрактные схемы алгоритмов

1 Теоретическая часть

Абстрактные схемы алгоритмов

Абстрактная схема алгоритма или, как её часто именуют, граф-схема, является самым удобным и самым обычным входным документом при проектировании управляющих автоматов. Она строится на основе микропрограмм тех процессов, которыми должен управлять автомат. В то же время для синтеза автомата конкретный смысл этих процессов, а следовательно и конкретное содержание микропрограмм, не имеет значения. По этой причине содержательное описание микропрограмм заменяется структурными обозначениями управляющих

сигналов или даже абстрактными буквами выходного алфавита *Y*, как показано на рис. 6.1.

Рис.6.1

Синтез автомата по схеме алгоритма (СА) является рутинной прикладной задачей, одной из задач логического проектирования цифровой электронной аппаратуры. Как правило, исходные данные для синтеза представлены содержательным описанием работы аппаратуры или схемой алгоритма со структурными обозначениями (входной алфавит и выходной).

Тем не менее, вести синтез в рамках структурной теории было бы трудно, так как еще не определены ни число состояний автомата, ни их коды. Поэтому при синтезе автомата по СА используется смешанное представление: в качестве входного алфавита используется структурный алфавит, в качестве выходного - структурный или абстрактный, а для внутренних состояний - непременно абстрактный алфавит. Лишь по окончании абстрактного синтеза принимается тот или иной вариант кодирования состояний и делается переход к структурному внутреннему алфавиту .

Здесь мы будем пользоваться абстрактными обозначениями выходных сигналов и называть буквы y_1, y_2, ... *операторами*, как это принято в специальной литературе. Смысл этого названия состоит в том, что каждая такая буква обозначает одну определенную *микрокоманду*, содержащую один или несколько управляющих сигналов *z*, и задает определенную совокупность одновременно выполняемых *микроопераций*. Входные структурные переменные u_1, u_2, ... также в соответствии с традицией будем называть *логическими условиями* (ЛУ). Переход от содержательных обозначений логических условий (например, на языке функционального микропрограммирования) к структурным переменным поясняет рис. 6.2

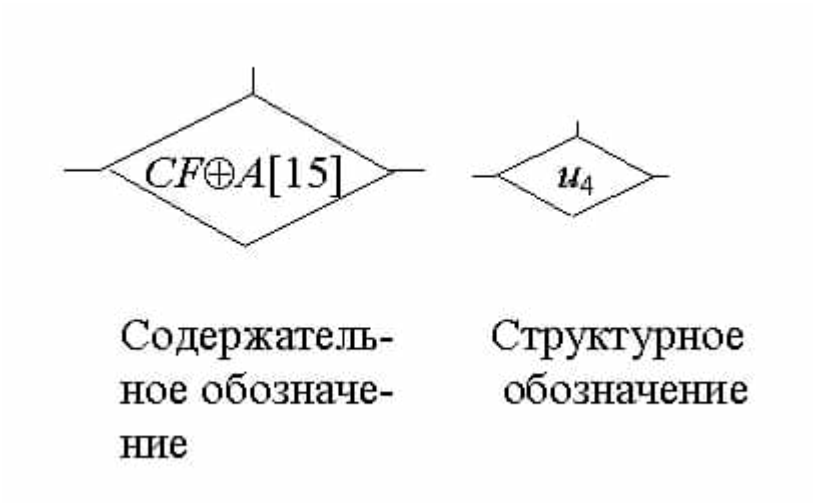


Рис. 6.2

Содержательный смысл операторов и ЛУ устанавливается на подготовительном этапе синтеза на основе входных данных об используемых функциональных модулях управляемой аппаратуры и её функциях. Смысл обозначений закрепляется таблицами.

Как правило, управляющий автомат должен реализовать некоторое множество микропрограмм, которые необходимо объединить в одну СА так, чтобы осталось одно общее начало и один конец. Ветвлением на отдельные микропрограммы управляют *внешние* по отношению к данному аппаратному блоку ЛУ, которые определяются кодом макрооперации. Число внешних ЛУ n определяется очевидным соотношением

$$n \geq log_2 m,$$

где m - число объединяемых микропрограмм. Одно значение вектора u (обычно нулевое) используется для задания режима ожидания. На рис. 6.3 показано ветвление по логическим условиям u_1, u_2, u_3 на семь путей, каждый из которых соответствует отдельной микропрограмме и условно заменен штриховой линией.

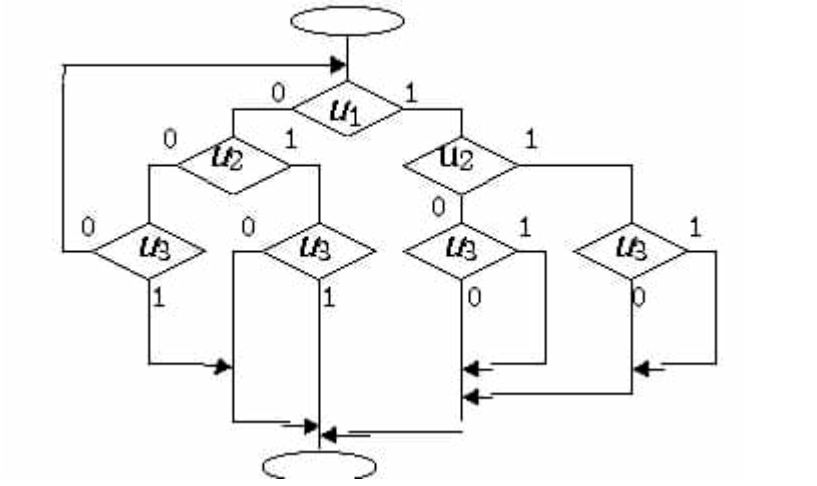


Рис. 6.3

Отдельные микропрограммы могут содержать не только одинаковые операторы, но и цепочки одинаково расположенных операторов или одинаковые подграфы с ветвлением на логических условиях. Эти совпадения можно использовать для упрощения СА. Для этого нужно объединить совпадающие участки отдельных

путей на СА, а если их продолжения не совпадают, то вновь организовать ветвление с помощью внешних ЛУ.

Кроме граф-схем для описания алгоритмов используются и другие средства, из которых в данной лекции мы рассмотрим два.

Матричная схема алгоритма (МСА) содержит ту же информацию, что и граф-схема, но в большей степени приспособлена для формализации задач по исследованию и преобразованию алгоритмов, в частности для объединения алгоритмов.

Формулы перехода являются средством описания алгоритма по частям с помощью булевой алгебры и в основном представляют интерес для преобразования матричных и других специальных описаний алгоритмов в граф-схемы.

Рассмотрим все названные средства на конкретном примере.

Пример 1. Требуется объединить алгоритмы, заданные на рис. 6.4 в виде двух граф-схем: СА1 и СА2.

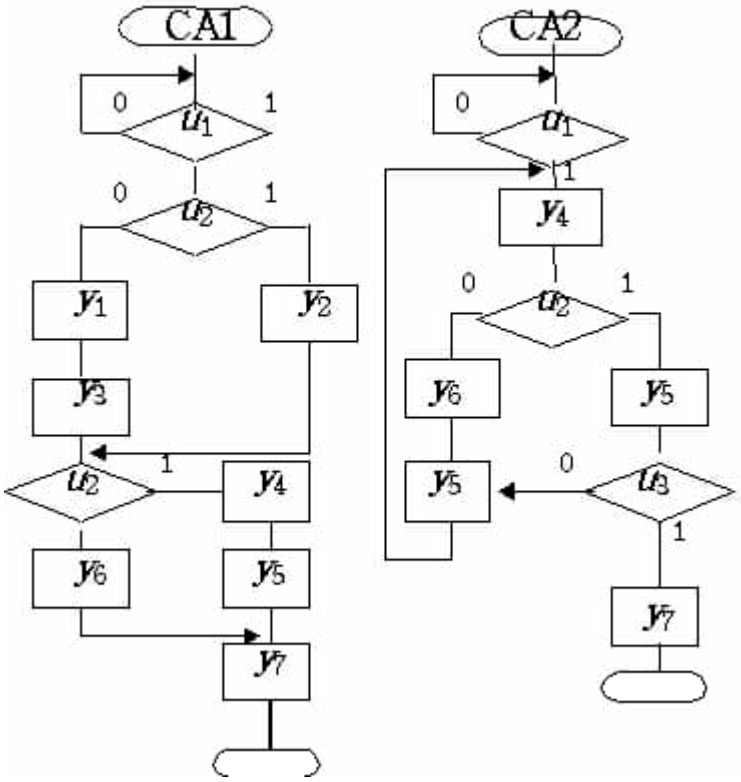


Рис.6.4

Самое очевидное и простое решение сводится к тому, что начальные терминаторы СА1 и СА2 заменяются одним, после которого делается проверка u_1. Далее следует проверка внешнего условия u_4 и далее, вплоть до оператора y_7 идут две раздельные ветви алгоритма. Перед оператором y_7 эти ветви сливаются, а после него находится общий конечный терминатор. Такое простое решение приводит к СА с одиннадцатью операторами и шестью проверками ЛУ.

Приступая к объединению алгоритмов с помощью МСА, необходимо устранить повторения одинаковых операторов в заданных СА. В данном примере имеется повторение оператора y_5 в СА2, что допустимо в граф-схеме, но не допустимо в других видах схем. Ведь в них переход от одного оператора к другому должен

определяться однозначно без помощи соединительных линий. Чтобы избежать неоднозначности переходов, один из операторов y_5 СА2 переименовываем в $y_{8(5)}$. Если впоследствии обнаружится возможность объединить операторы y_5 и $y_{8(5)}$, то переименование будет отменено.

МСА представляет квадратную таблицу с горизонтальными входами от $y_{нач}$ до y со старшим из имеющихся порядковых индексов и вертикальными - от y_1 до $y_{конечн}$. В клетки таблицы записываются логические условия соответствующего перехода. МСА для первого из алгоритмов представлена в табл. 6.1.

Таблица 6.1

	y_1	y_2	y_3	y_4	y_5	y_6	y_7	$y_к$
y_0	$u_1\bar{u}_2$	u_1u_2						
y_1			1					
y_2				u_2		$\bar{u}_2$		
y_3				u_2		$\bar{u}_2$		
y_4					1			
y_5							1	
y_6							1	
y_7								1

Отметим одно важное свойство МСА: дизъюнкция содержимого всех клеток одной строки равна единице. Исключением является равенство дизъюнкции булевой переменной ждущей вершины СА (как это получилось в верхней строке табл. 6.1).

Логическим условием безусловного перехода является константа единица. Если по граф-схеме прослеживается несколько параллельных путей от одного оператора к другому, то соответствующие логические условия должны записываться в клетки МСА через знак дизъюнкции.

МСА для второго алгоритма представлена в табл. 6.2.

Таблица 6.2

	y_4	y_5	$y_{8(5)}$	y_6	y_7	$y_к$
y_0	u_1					
y_4		u_2		$\bar{u}_2$		
y_5			$\bar{u}_3$		u_3	
$y_{8(5)}$	1					
y_6			1			
y_7						1

Объединенная МСА содержит строки и столбцы для всех операторов, встречающихся хотя бы в одном из объединяемых алгоритмов. В её строках взятые из первой МСА условия дополнительно умножаем на u_4, а взятые из второй - на u_4

Объединенная МСА представлена в табл. 6.3.

Таблица 6.3

	y_1	y_2	y_3	y_4	y_5	$y_{8(5)}$	y_6	y_7	$y_к$
$y_н$	$u_1\overline{u}_2\overline{u}_4$	$u_1u_2\overline{u}_4$		u_1u_4					
y_1			1						
y_2				u_2			$\overline{u}_2$		
y_3				u_2			$\overline{u}_2$		
y_4					$\overline{u}_4 \vee u_2u_4$		$\overline{u}_2u_4$		
y_5						$\overline{u}_3u_4$		$\overline{u}_4 \vee u_3u_4$	
$y_{8(5)}$				1					
y_6						u_4		$\overline{u}_4$	
y_7									1

На основании объединенной МСА , прослеживая от строки к столбцу все условия всех переходов, составим формулы перехода:

$y_н \to u_1\overline{u}_2\overline{u}_4y_1 \vee u_1u_2\overline{u}_4y_2 \vee u_1u_4y_4;$

$y_1 \to y_3;$

$y_2 \to u_2y_4 \vee \overline{u}_2y_6;$

$y_3 \to u_2y_4 \vee \overline{u}_2y_6;$

$y_4 \to \overline{u}_4y_5 \vee u_2u_4y_5 \vee \overline{u}_2u_4y_6;$

$y_5 \to \overline{u}_3u_4y_{8(5)} \vee \overline{u}_4y_7 \vee u_3u_4y_7;$

$y_{8(5)} \to y_4;$

$y_6 \to u_4y_{8(5)} \vee \overline{u}_4y_7;$

$y_7 \to y_к.$

Для построения объединенной граф-схемы эти формулы нужно привести к виду так называемых *элементарных выражений,* т.е. произвести разложение по переменным *u*. Не происходит подобное разложение только по переменным, задающим режим ожидания (в данном примере буква u_1).

После несложных преобразований формул перехода получим элементарные выражения:

$$y_н \to u_1(\bar{u}_4(\bar{u}_2 y_1 \vee u_2 y_2) \vee u_4 y_4);$$
$$y_1 \to y_3;$$
$$y_2 \to u_2 y_4 \vee \bar{u}_2 y_6;$$
$$y_3 \to u_2 y_4 \vee \bar{u}_2 y_6;$$
$$y_4 \to \bar{u}_4 y_5 \vee u_4(u_2 y_5 \vee \bar{u}_2 y_6);$$
$$y_5 \to \bar{u}_4 y_7 \vee u_4(\bar{u}_3 y_{8(5)} \vee u_3 y_7);$$
$$y_{8(5)} \to y_4;$$
$$y_6 \to u_4 y_{8(5)} \vee \bar{u}_4 y_7;$$
$$y_7 \to y_к.$$

При составлении объединенной граф - схемы (рис. 6.5) будем учитывать следующие обстоятельства. Присутствие в регулярном выражении какого-нибудь ЛУ только с инверсией или только без инверсии (как u_1 в регулярном выражении $y_н$) однозначно указывает на наличие в данном алгоритме ждущего условия. Это должно быть отражено в граф-схеме. Сравнение формул перехода для y_5 и $y_{8(5)}$ показывает их несовпадение. Следовательно, раздельное обозначение для этих операторов должно быть сохранено.

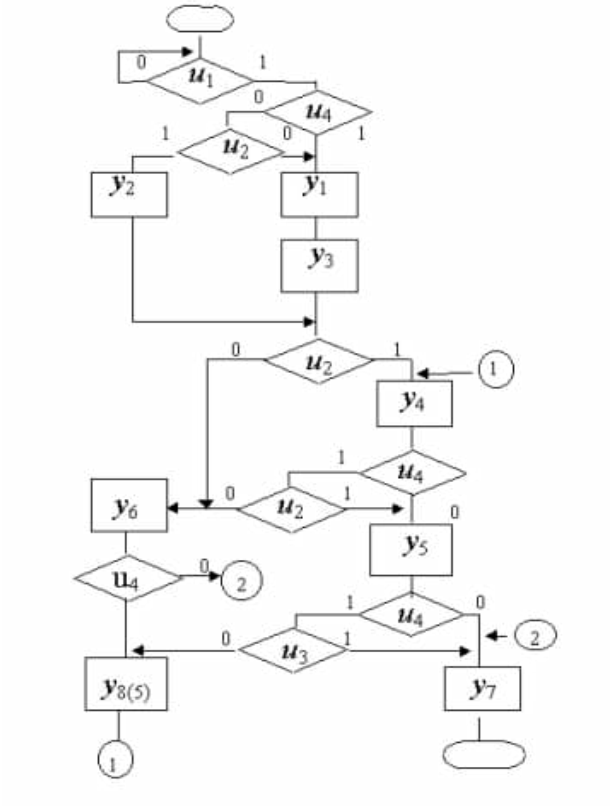


Рис. 6.5

Новая СА имеет восемь операторов и девять проверок логических условий. Это - характерный результат формальной методики: число операторов сводится к минимуму, зато растет число проверок логических условий. Практически важно и то и другое, так как от числа операторов зависит количество состояний проектируемого автомата, а от числа проверок ЛУ зависит сложность функции переходов на структурном уровне.

Анализ распределений сдвигов применяется для уменьшения числа проверок логических условий.

Сдвигом называется возможность изменения ЛУ при выполнении микрокоманды. Например, при выполнении элементарной арифметической операции может произойти изменение состояния знаковой цифры, обнуление счетчика, переполнение и т.п.

Распределениями сдвигов называются подмножества логических условий, сдвиг которых связан с определенным оператором. Распределений сдвигов столько же, сколько имеется операторов (кроме конечного). Они могут пересекаться.

Универсальным распределением сдвигов называется всё множество ЛУ данного алгоритма. Считается, что универсальным распределением обладает оператор $y_{нач}$, так как перед началом работы значения ЛУ вообще говоря еще не определены.

Если ЛУ содержится в распределении сдвигов некоторого оператора y_i, то после выполнения этого оператора значение ЛУ необходимо проверять заново. В противном случае можно считать ранее выясненное значение ЛУ сохранившимся и исключить из алгоритма его повторную проверку.

На рис. 6.5 показана трехкратная проверка условия u_2. Если выяснится, что ЛУ u_2 не содержится в распределениях сдвигов операторов y_1, y_2 и y_3, то вторую проверку этого условия можно будет исключить, непосредственно соединив y_3 с y_6 и y_2 с y_4. Если u_2 не содержится в распределении сдвигов оператора y_4, то выход y_4 нужно прямо соединить с y_5 - ведь перед выполнением y_4 ЛУ u_2 имело значение 1 и не должно было измениться.

2. Практическая часть

Пример 2. Зададимся схемой алгоритма, представленной на рис. 6.6.

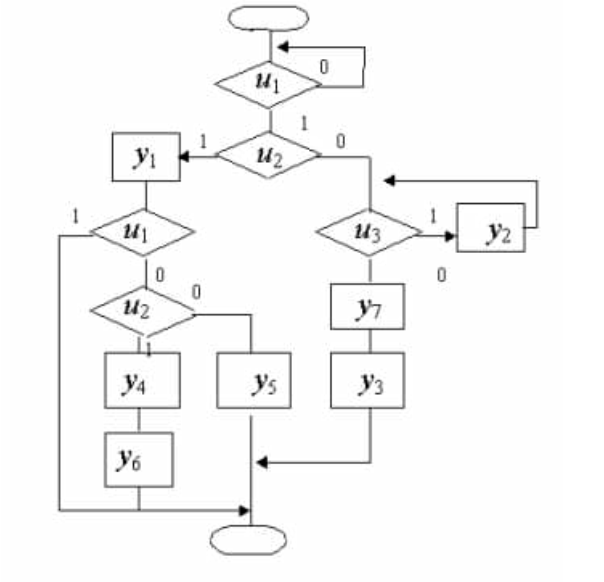


Рис. 6.6

Примем следующее распределение сдвигов:

$y_1\{u_2, u_3\}$; $y_5\{Ж\}$;

$y_2\{u_1, u_3\}$; $y_6\{Ж\}$;

$y_3\{Ж\}$; $y_7\{u_2\}$.

$y_4\{u_1, u_2, u_3\}$;

Оператор y_1 выполняется только после $u_1 = 1$. В распределении сдвигов y_1 ЛУ u_1 не содержится. Из этого рассуждения следует, что после выполнения y_1

повторная проверка u_1 не требуется. Вместо того, чтобы повторно проверять u_1 просто соединяем выход y_1 с конечным терминатором. Исчезает значительная часть схемы (две проверки ЛУ и три оператора). Здесь анализ распределений сдвигов позволил не только сократить число проверок ЛУ, но и заметно упростить алгоритм в целом. Необходимо отметить, что положительный результат получается только за счёт устранения избыточности первоначальных СА. В грамотно составленных содержательных СА такая возможность, как правило, отсутствует.

Министерство науки и высшего образования Российской Федерации
ФГБОУ ВО «Тульский государственный университет»
Технический колледж им. С.И. Мосина

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
по выполнению курсовой работы

по междисциплинарному курсу
МДК 01.01 «Основы проектирования цифровой техники»

специальность

09.02.01 Компьютерные системы и комплексы

Тула 2023

УТВЕРЖДЕНЫ
Цикловой комиссией информационных технологий
Протокол от «13» января 2023 г. № 6
Председатель цикловой комиссии И.В. Миляева

Содержание

Введение

Курсовая работа по дисциплине «Проектирование цифровых устройств» является самостоятельной работой студента.

Целями курсовой работы являются:

1. Закрепление и углубление знаний, полученных в процессе изучения дисциплины;
2. Развитие у студентов способности самостоятельно решать технические задачи, связанные с расчетом базовых узлов цифровой электроники, ознакомление с современной элементной базой цифровой электроники;
3. Обучение использованию справочной литературы и нормативно-технической документации (ГОСТ, ЕСКД, РД);
4. Развитие навыков оформления работы в соответствии с требованиями ГОСТ и ЕСКД.

1. Основные требования к курсовой работе

1.1. Тематика курсовой работы

Темой курсовой работы по дисциплине «Проектирование цифровых устройств» может быть расчет временных характеристик комбинационных и последовательностных схем, минимизация комбинационных схем, разработка различных цифровых узлов.

Примерная тематика курсовых работ приведена в табл. 1.

Таблица 1. Примерная тематика курсовых работ

№ п.п.	Тема
1.	Разработка восьмиразрядной схемы контроля четности.
2.	Разработка цифрового устройства управления шаговым двигателем.
3.	Разработка цифрового автомата контроля последовательного кода.
4.	Разработка счетного блока цифрового частотомера.
5.	Проектирование цифрового узла схемы управления трехфазным двигателем.
6.	Разработка преобразователя кода Грея в код семисегментного индикатора.
7.	Разработка устройства преобразования параллельного кода в последовательный с контролем четности.
8.	Разработка контроллера модуля статического ОЗУ.
9.	Разработка двухпортового ОЗУ.
10.	Разработка преобразователя двоичного кода в код семисегментного индикатора.
11.	Разработка устройства динамической индикации.
12.	Разработка устройства суммирования двух восьмиразрядных чисел.
13.	Разработка модуля оперативной памяти с трехкратным резервированием и индикацией ошибок.
14.	Разработать преобразователь двоичного кода в код Хемминга. Вход – четырехразрядный, выход – семиразрядный.

15.	Разработать трехразрядную схему совпадения, с двумя группами входов по три разряда и одним выходом.
16.	Разработать нереверсивное шестнадцатиразрядное устройство быстрого сдвига. Схема имеет шестнадцать информационных входов параллельной загрузки, тактовый вход С, вход разрешения параллельной загрузки S, шестнадцать выходов.
17.	Разработать комбинационную схему, на входы которой подаются два восьмиразрядных числа без знака X и Y, и сигнал управления S. Схема имеет восьмиразрядный выходной сигнал Z. Если S=1, то выход Z=min(X,Y), если S=0, то Z=max(X,Y).
18.	Разработать приоритетный шифратор, на вход которого поступает четырехразрядный двоичный код, а на выходе формируется приоритетный десятиразрядный код.
19.	Разработать преобразователь шестнадцатиразрядного приоритетного кода в четырехразрядный двоичный код.
20.	Разработать схему приема асинхронного внешнего сигнала в тактируемую последовательностную схему с защитой от сбоев.
21.	Разработать схему преобразования восьмиразрядного параллельного кода в последовательный с формированием на выходе трех сигналов – информационного сигнала в последовательном коде, тактового сигнала и синхронизирующего сигнала, равного единице в течении передачи информационных битов и равного нулю в противном случае.
22.	Разработать схему преобразования последовательного кода в восьмиразрядный параллельный код. Схема имеет три входных сигнала – информационный сигнал в последовательном коде, тактовый сигнал и синхронизирующий сигнал, равный единице в течении передачи информационных битов и равного нулю в противном случае. На выходе формируется информационный сигнал в параллельном коде и синхронизирующий сигнал, равный единице после завершения приема байта.
23.	Разработать самокорректирующийся восьмиразрядный счетчик.
24.	Разработать блок перемножения двух восьмиразрядных чисел без знака.

1.2. Исходные данные к курсовой работе

Пример исходных данных к курсовой работе приведен ниже.

Выполнить анализ временных характеристик восьмиразрядной схемы проверки на четность, построенной на элементах «исключающее или», если время задержки распространения сигнала в одном логическом элементе составляет от 3 до 5 нс, время задержки в соединениях между элементами составляет от 0,01 до 0,1 нс, Разброс времени поступления входных сигналов составляет 1 нс. Требуемое время прибытия на выходе схемы составляет 12 нс.

1.3. Задание на курсовую работу

Задание на курсовую работу оформляется на бланке задания (см. Приложение 1). Задание подписывается студентом, принявшим задание, и

преподавателем, выдавшим задание. Получив задание, студент обязан внимательно ознакомиться с его содержанием, и в случае неясностей в поставленной задаче получить необходимые консультации у руководителя.

В задании указываются исходные данные к курсовому проекту - тип схемы, функциональное назначение проектируемого устройства, требуемый частотный диапазон, параметры входных и выходных сигналов и другие необходимые требования, определяемые назначением проектируемого устройства.

1.4. Объем и содержание курсовой работы

Курсовая работа включает в себя графическую часть и расчетно-пояснительную записку.

Графическая часть состоит из одного листа формата А1, содержание которого оговаривается в задании.

На листе графической части обычно приводится принципиальная электрическая схема устройства, могут быть приведены расчеты схемы.

Расчетно-пояснительная записка должна иметь объем 20-25 страниц машинописного или рукописного текста на листах формата А4. Записка должна содержать материалы, поясняющие назначение, состав, и принцип действия схемы. Должен быть проведен расчет номиналов всех элементов схемы и выбор стандартных элементов по рассчитанным параметрам. В записке также должен содержаться расчет параметров спроектированной схемы и выводы о проделанной работе.

Примерное распределение материала расчетно-пояснительной записки по разделам приведено в табл. 2.

Таблица 2. Содержание пояснительной записки

Разделы курсового проекта	Объем раздела
Введение	1-2 стр. (5-10 %)
Обзор и анализ возможных решений	7-8 стр. (10-20 %)
Основная расчетная часть	15-20 стр.(50-60 %)
Анализ полученных результатов	2-3 стр. (5 – 15 %)
Список использованной литературы	1-2 стр. (5-7%)
Приложения	

Если в работе содержатся расчеты, выполненные с применением ЭВМ, то в расчетно-пояснительной записке обязательно должна быть ссылка на используемое программное обеспечение. В случае использования самостоятельно разработанных программ необходимо приводить описание и обоснование используемых алгоритмов.

При оформлении пояснительной записки не следует включать в нее справочный материал, кроме необходимых для расчета значений параметров элементов. При этом обязательно наличие ссылок на используемую техническую и справочную литературу.

При оформлении пояснительной записки на компьютере ее следует набирать шрифтом “Times New Roman” 14 pt, через одинарный или полуторный интервал. Поля страницы – все, кроме левого – 1,5 см, левое - 2,5 см.

Оформление текстовых документов регламентируется ГОСТ 2.105-95 ЕСКД – "Общие требования к текстовым документам", ГОСТ 2.106-68 – "Текстовые документы", ГОСТ 2.105-79 ЕСКД "Основные требования к текстовым документам" и другими нормативными документами, указанными в приложении 2.

Список использованных источников оформляется в соответствии с требованиями ГОСТ 7.1-2003;

Графическую часть следует выполнять в соответствии с требованиями следующих основных нормативных документов:

-ГОСТ 2.109-73 ЕСКД "Основные требования к чертежам";

-ГОСТ 2.108-68 ЕСКД "Спецификация (1-1-73*)";

-ГОСТ 2.407-78 ЕСКД "Правила выполнения чертежей печатной платы";

-ГОСТ 2.702-75 ЕСКД "Правила выполнения электрических схем".

Более полный список нормативных документов указан в приложении 2.

Примечание: Допускается не делать рамку на листах пояснительной записки.

1.5. Выполнение курсовой работы

Выполнение курсовой работы определяется графиком, приведенным в табл. 3.

Таблица 3. График выполнения курсовой работы

Недели	Проценты	Этап выполнения
1-2	5%	Получение задания
3-4	10%	Патентно-библиографический поиск
5–6	10%	Анализ источников, выбор решения
7-8	25%	Разработка и расчет схемы
9-10	25%	Разработка и расчет печатной платы
11-12	20%	Оформление пояснительной записки и графической части
13-14	5 %	Защита проекта

1.6. Защита курсовой работы

Защита курсовой работы проводится публично перед комиссией в составе 2-3 преподавателей. Перед защитой курсовой проект должен быть подписан выполнившим его студентом, руководителем проекта и нормоконтролером. К защите представляются чертежи и расчетно-пояснительная записка. Студент делает доклад по содержанию выполненной работы и отвечает на вопросы членов комиссии.

При оценке курсового проекта учитываются:

- содержание пояснительной записки (соответствие расчетной части заданию, правильность выполнения расчетной части, соответствие оформления записки требованиям ГОСТ и ЕСКД, стиль изложения материала, аккуратность выполнения записки);

- содержание графической части (соответствие заданию, правильность выполнения электрических принципиальных схем, печатных плат, их взаимное соответствие, соответствие оформления листов требованиям ГОСТ и ЕСКД);

- качество доклада (понимание студентом излагаемого материала, общая подготовка в области выполняемой работы, умение представить материал);

- соблюдение сроков, предусмотренных графиком выполнения проекта.

Учет всех критериев определяет объективность оценки защиты проекта. Основные критерии оценки курсового проекта и их значимость приведены в табл. 4.

Таблица 4. Критерии оценки курсовой работы

качество работы	оценка рецензента	качество доклада	уровень защиты	сумма баллов
до 35	до 5	до 20	до 40	до 100

2. Методические указания по выполнению курсовой работы

2.1. Основные этапы проектирования цифровых устройств

Достоинством цифровых устройств обработки сигнала является возможность неограниченно долгого хранения запомненных сигналов, легкая возможность адаптации, высокая технологичность в производстве, отсутствие регулировок в процессе изготовления, большие перспективы микроминиатюризации. Исходя из общих принципов проектирования цифровых устройств, можно предложить следующую последовательность их проектирования:

1. Уточнение технического задания на цифровое устройство.
2. Расчет цифрового устройства в целом.
3. Разработка функциональной схемы цифрового устройства.
4. Оценка возможностей выбранной функциональной схемы.
5. Выбор элементной базы.
6. Разработка принципиальной схемы цифрового устройства.
7. Компьютерное моделирование цифрового устройства.

Уточнение технического задания на цифровое устройство. Как правило, при проектировании новой техники первоначальное задание приходится уточнять, поскольку многие технические требования являются взаимно противоречивыми. Например, требования по помехоустойчивости и требования по повышению быстродействия во многом являются взаимно исключающими, поскольку повышение частоты работы системы повышает уровень излучаемых помех. Кроме того, возможности имеющейся элементной базы могут быть недостаточными для удовлетворения всех требований первоначального технического задания.

Расчет цифрового устройства в целом. При системном подходе к проектированию, важным этапом является расчет общих параметров цифровой системы, который показывает принципиальные возможности реализации системы.

Разработка функциональной схемы. На этом этапе определяют метод решения поставленной технической задачи, способ реализации структуру устройства.

Оценка возможностей выбранной функциональной схемы. После предварительной проработки схемы определяются ее параметры. Этот этап позволяет оценить возможность реализации требований технического задания при выбранном схемотехническом решении.

Выбор элементной базы. Определяется требованиями к цифровому устройству, заданным уровнем интеграции, рабочим диапазоном частот.

Выбор принципиальной схемы. Предполагает расчет всех узлов и элементов цифрового устройства. Первым этапом разработки принципиальной схемы цифрового устройства является разработка его структурной схемы, где проектируемое устройство разбивается на узлы (структурные блоки более низкого уровня, чем проектируемое цифровое устройство) - шифраторы, мультиплексоры, сумматоры, счетчики, регистры и так далее. Синтез таких устройств проводится известными методами. Для этого сначала составляется таблица истинности проектируемого узла и далее из соображений оптимизации доопределяется проектировщиком, если такая возможность имеется. На основе такой доопределенной таблицы создается аналитическая модель узла, которая, как правило, нуждается в оптимизации. После минимизации вычисленные минимальные формы реализуются в различных базисах. Для полученных вариантов подбирается элементная база и результаты сравниваются по различным критериям: потребляемой мощности, размерам, быстродействию и т.д.

Компьютерное моделирование является заключительным этапом проектирования. Позволяет наглядно проверить работоспособность разработанного устройства.

2.2.. Этапы разработки логической схемы цифрового устройства (ЦУ)

При наличии у разработчика небольшого опыта проектирования ЦУ можно рекомендовать следующую последовательность действий.

Этап 1. Составление таблицы истинности цифрового устройства.

Основная цель этапа – формализация задачи, в процессе которой нужно определить значения функции для каждой комбинации значений аргументов. Результат этапа – таблица истинности. Это задача, неоднозначное толкование которой невозможно. После этого можно строить аналитическую модель цифрового устройства.

Этап 2. Построение карт Карно и доопределение функции.

Минимизация логических функций небольшого числа аргументов обычно выполняется на основе карт Карно. Если функция определена не при всех наборах аргументов, то ее нужно доопределить. При малом числе неопределенных значений лучше рассмотреть несколько вариантов. Если же число безразличных значений или самих аргументов большое, то, возможно, придется полностью определить функцию всеми нулями или всеми единицами, чтобы в результате уменьшить число членов аналитической модели.

Этап 3. Минимизация цифрового устройства.

Минимизация логических функций осуществляется на основе карт Карно. При минимизации функции с несколькими выходами для каждой выходной переменной строится карта Карно. Для максимальной минимизации логического устройства также строятся карты Карно для произведений логических функций, соответствующих различным выходным переменным. Это позволяет выявить

произведения, общие для нескольких выходов. Такие произведения реализуются один раз для нескольких выходов, что позволяет избежать дублирования и уменьшить сложность схемы.

Этап 4. Реализация полученных минимальных форм в различных базисах.

Карты Карно позволяют получить минимальную реализацию в виде суммы произведений или в виде произведения сумм, т.е. на элементах «или», «и», «не». Однако в некоторых случаях реализация в нестандартных базисах оказывается проще. Примером такого устройства является схема контроля на четность, которая легче всего строится на элементах «исключающее или».

Этап 5. Оценка различных полученных вариантов и выбор наилучшего.

Практически любая схема допускает несколько вариантов реализации. Выбор того или иного варианта определяется наиболее важным критерием оптимизации системы – максимальным быстродействием, наименьшей сложностью, минимальной потребляемой мощностью, простотой тестирования и отладки, и.т.д.

Этап 6. Оценка динамических характеристик схемы. Синтез схемы по картам Карно определяет логику работы схемы, но не учитывает динамических характеристик. Наиболее простым способом определения соответствия схемы требованиям по быстродействию является статический временной анализ.

2.3. Пример проектирования цифрового устройства

Процедуру проектирования комбинационной части рассмотрим на примере разработки дешифратора кода Грея в код семисегментного индикатора.

Первый этап – составление таблицы истинности для перехода от кода Грея к двоичному коду.

Среди невзвешенных двоичных кодов специальное применение находят такие, у которых переход к соседнему числу сопровождается изменениями только в одном разряде (коды с обменной единицей). Так, в технике аналого-цифрового преобразования и пересчетных устройствах широко используется код Грея, называемый также циклическим или рефлексно-двоичным кодом. Он позволяет существенно сократить преобразования, упросить кодирующую логику, а также повысить эффективность защиты от нежелательных сбоев. Недостатком кода Грея является то, что в нем затруднено выполнение арифметических операций и цифроаналогового преобразования. Поэтому при необходимости код Грея преобразуется в обычный двоичный код.

Переход от двоичного кода к коду Грея осуществляется по правилу:

старшие разряды двоичного кода и кода Грея совпадают, а любой следующий разряд Y_k кода Грея равен сумме по модулю 2 соответствующего X_k и предыдущего X_{k-1} разрядов двоичного кода, т.е.

$Y_k = X_k \oplus X_{k-1}$.

Таким образом, получаем таблицу соответствия десятичного кода, двоичного кода и кода Грея:

Таблица 5.Соответствие двоичного кода и кода Грея

Десятичный код	Двоичный код	Код Грея
0	0000	0000
1	0001	0001

2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101

Один из способов цифровой индикации ориентирован на использовании семисегментного индикатора (рис. 1).

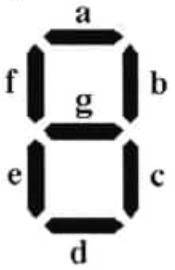


Рисунок 1 - Обозначение элементов семисегментного индикатора

Для синтеза преобразователя, управляющего семисегментным индикатором, при помощи кодов Грея составим таблицу соответствия кодов. Десятичные цифры, которые необходимо высвечивать на индикаторе, задаются в 4-разрядном коде Грея. Если элемент светится, то он находится в состоянии «1», если нет – «0». На основании указанных соображений построим таблицу соответствия (табл. 6).

Таблица 6. Соответствие кода Грея и кода семисегментного индикатора

	Код Грея				Код семисегментного индикатора						
	X4	X3	X2	X1	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	1	1	1	0	1	1	0	1
3	0	0	1	0	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1	0	0	1	1
5	0	1	1	1	1	0	1	1	0	1	1
6	0	1	0	1	1	0	1	1	1	1	1
7	0	1	0	0	1	1	1	0	0	0	0
8	1	1	0	0	1	1	1	1	1	1	1
9	1	1	0	1	1	1	0	1	1	1	1

Второй и третий этапы – построение карт Карно и минимизация логических выражений.

Для минимизации логического блока дешифратора на основании табл. 6 построим карты Карно, связывающие выходные переменные с входными. Карты Карно для первых четырех разрядов семисегментного кода показаны ниже:

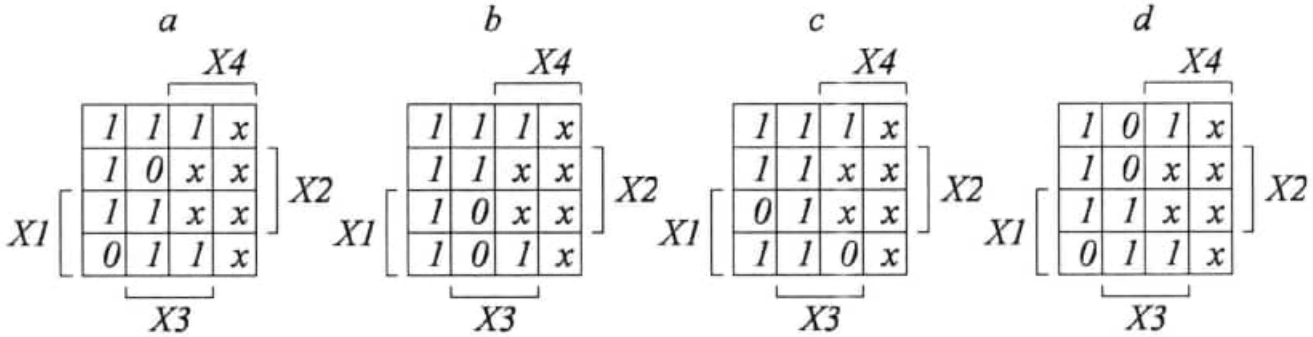


Рисунок 2 - Карты Карно дешифратора

Рассматриваемый дешифратор является неполным, поскольку таблица истинности определяет не все возможные комбинации входных переменных. В карте Карно эти комбинации обозначены символом "x".

Доопределение выполняется на основе различных соображений. Существует два основных подхода – максимальной простоты и максимальной надежности.

При использовании подхода максимальной простоты доопределение карт Карно проводится исходя из получения наибольших областей покрытия.

Четвертый этап. Реализация полученных минимальных форм в различных базисах. При построении схемы по картам Карно получают две основные формы – в виде суммы произведений, при выделении единичных контуров, и в виде произведения сумм, при выделении нулевых контуров. Выбор той или иной формы представления обусловлен получаемой сложностью схемы. В случае, если требуется противогоночная реализация, вводятся дополнительные произведения или суммы, для исключения источников опасностей.

Пятый этап. Оценка различных полученных вариантов и выбор наилучшего.

Практически любая схема допускает несколько вариантов реализации. Выбор того или иного варианта определяется наиболее важным критерием оптимизации системы – максимальным быстродействием, наименьшей сложностью, минимальной потребляемой мощностью, простотой тестирования и отладки, и.т.д.

Этап 6. Оценка динамических характеристик схемы. Синтез схемы по картам Карно определяет логику работы схемы, но не учитывает динамических характеристик. Наиболее простым способом определения соответствия схемы требованиям по быстродействию является статический временной анализ.

Главные компоненты задержки в последовательных схемах:

1. Задержки в логических элементах, обусловленные ограниченной скоростью переключения элементов;
2. Задержки в соединениях логических элементов, связанные с распространением сигнала в длинных проводах, индуктивностью и емкостью нагрузки на выходе логического элемента;
3. Задержки, обусловленные неодновременным поступлением входных сигналов.
4. Задержки, обусловленные неодновременным считыванием выходных сигналов.

Современные комбинационные устройства имеют высокую сложность и большое количество входов/выходов, поэтому определить их быстродействие

посредством полного моделирования невозможно из-за неприемлемых затрат времени. Метод статического временного анализа позволяет оперативно оценить временные характеристики комбинационных схем, хотя во многих случаях дает заниженные оценки быстродействия (ведет к перестраховке в проектировании).

При статическом временном анализе комбинационную схему представляют в виде направленного ациклического графа, где каждое ребро и вершина несут на себе вес - задержку проводов и логических элементов, соответственно.

Задача статического временного анализа традиционно решается за два прохода схемы: прямой (от первичных входов к первичным выходам) и обратный (от выходов к входам). Во время прямого прохода для каждой вершины схемы вычисляется позднее фактическое время прибытия сигнала и раннее фактическое время прибытия сигнала. Во время обратного прохода вычисляется позднее требуемое время прибытия сигнала и раннее требуемое время прибытия сигнала.

Интервал между ранним и поздним фактическим временем прибытия - это временное окно, внутри которого в данной вершине возможно изменение состояния выходов.

Интервал между ранним и поздним требуемым временем прибытия – это временное окно, в котором считываются значения выходов комбинационной схемы.

Если в каждом узле схемы интервал фактического времени прибытия содержится внутри интервала требуемого времени прибытия, ограничения на быстродействие схемы выполняются.

Фактическое время прибытия на входных контактах схемы считается изначально определенным, и используется для расчета фактического времени прибытия на выходе каждого логического элемента.

Позднее фактическое время прибытия $t_{\text{п.ф. max}}$ определяется по формуле:

$$t_{\text{п.ф. max}}(i) = \max_{j \in FI(i)} \left(t_{\text{п.ф. max}}(j) + t_{\max}(j,i) \right), \quad (1)$$

где i – номер вершины; $FI(i)$ – множество вершин, соединенных с входом вершины i; $t_{\text{п.ф. max}}(j)$ – позднее фактическое время прибытия в вершине j; $t_{\max}(j,i)$ – максимальная задержка между выходом вершины j и выходом вершины i.

Раннее фактическое время прибытия $t_{\text{п.ф. min}}$ определяется по формуле:

$$t_{\text{п.ф. min}}(i) = \min_{j \in FI(i)} \left(t_{\text{п.ф. min}}(j) + t_{\min}(j,i) \right), \quad (2)$$

где i – номер вершины; $FI(i)$ – множество вершин, соединенных с входом вершины i; $t_{\text{п.ф. min}}(j)$ – раннее фактическое время прибытия в вершине j; $t_{\min}(j,i)$ – минимальная задержка между выходом вершины j и выходом вершины i.

Требуемое позднее время прибытия для выходных контактов схемы считается заданным требованиями технического задания. На основании этого времени считается требуемое время прибытия на выходе каждого логического элемента схемы:

$$t_{\text{п.тр. max}}(i) = \min_{j \in FO(i)} \left(t_{\text{п.тр max}}(j) - t_{\max}(j,i)\right), \quad (3)$$

где i – номер вершины; $FO(i)$ – множество вершин, подключенных к выходу вершины i; $t_{\text{п.тр max}}(j)$ – раннее позднее время прибытия в вершине j; $t_{\max}(j,i)$ – максимальная задержка между вершинами j и i.

Требуемое раннее время прибытия для выходных контактов схемы считается заданным требованиями технического задания. На основании этого времени считается требуемое время прибытия на выходе каждого логического элемента схемы:

$$t_{\text{п.тр. min}}(i) = \max_{j \in FO(i)} \left(t_{\text{п.тр min}}(j) - t_{\min}(j,i)\right), \quad (4)$$

где i – номер вершины; $FO(i)$ – множество вершин, подключенных к выходу вершины i; $t_{\text{п.тр min}}(j)$ – раннее требуемое время прибытия в вершине j; $t_{\min}(j,i)$ – минимальная задержка между вершинами j и i.

Для проверки соответствия схемы требованиям проектирования для каждой вершины вычисляются временные запасы:

$$t_{\text{зап min}}(i) = t_{\text{п.тр min}}(i) - t_{\text{п.ф min}}(i), \quad (5)$$

где $t_{\text{зап min}}$ – временной запас по минимальному времени прибытия.

$$t_{\text{зап max}}(i) = t_{\text{п.тр max}}(i) - t_{\text{п.ф max}}(i), \quad (6)$$

где $t_{\text{зап min}}$ – временной запас по минимальному времени прибытия.

Отрицательный временной запас в любой точке схемы означает, что временные ограничения по работоспособности не выполняются. Положительный запас на всех выходах означает, что схема выполняет требования.

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

Основная литература

1 Виноградов, М. В. Проектирование цифровых устройств : учебное пособие для СПО / М. В. Виноградов, Е. М. Самойлова. — Саратов : Профобразование, Ай Пи Ар Медиа, 2019. — 106 с. — ISBN 978-5-4488-0429-8, 978-5-4497-0229-6. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: http://www.iprbookshop.ru/86704.html

2 Сажнев, А. М. Микропроцессорные системы: цифровые устройства и микропроцессоры : учебное пособие для среднего профессионального образования / А. М. Сажнев. — 2-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2020. — 139 с. — (Профессиональное образование). — ISBN 978-5-534-12092-9. — Текст : электронный // ЭБС Юрайт [сайт]. — URL: https://urait.ru/bcode/457218

3 Ушенина, И. В. Проектирование цифровых устройств на ПЛИС : учебное пособие / И. В. Ушенина. — Санкт-Петербург : Лань, 2019. — 408 с. — ISBN 978-5-8114-3657-6. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/119638

Дополнительная литература

1 Калабеков, Б.А. Цифровые устройства и микропроцессорные системы : Учебник для сред.спец.учеб.заведений / Б.А.Калабеков. 2-е изд.,перераб.и доп. М. : Горячая линия-Телеком, 2000. 336с. : ил. ISBN 5-93517-008-6

2 Пухальский, Г. И. Проектирование цифровых устройств : учебное пособие / Г. И. Пухальский, Т. Я. Новосельцева. — Санкт-Петербург : Лань, 2012. — 896 с. — ISBN 978-5-8114-1265-5. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/68474

3 Акимова, Е. В. Вычислительная техника : учебное пособие / Е. В. Акимова. — Санкт-Петербург : Лань, 2020. — 68 с. — ISBN 978-5-8114-4925-5. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/142354

Интернет-ресурсы

1 ЭБС Юрайт. - Интернет- ссылка https://urait.ru/
2 ЭБС BOOK.ru. - Интернет- ссылка https://www.book.ru/
3 ЭБС Лань. - Интернет-ссылка https://e.lanbook.com/
4 ЭБС IPRBooks. - Интернет- ссылка http://www.iprbookshop.ru/
5 НЭБ eLibrary. - Интернет-ссылка https://www.elibrary.ru/

Приложение 1
ПРИМЕР ОФОРМЛЕНИЯ ЗАДАНИЯ НА КУРСОВУЮ РАБОТУ

Министерство науки и высшего образования Российской Федерации
ФГБОУ ВО «Тульский государственный университет»
Технический колледж им. С.И. Мосина

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

по МДК 01.02 «Проектирование цифровых устройств»_____

Студент _____группа _____

1. Тема «Разработка цифрового устройства управления шаговым двигателем»

2. Срок представления проекта к защите "_____"_____

3. Исходные данные для проектирования: Разработать цифровое устройство управления шаговым двигателем, обеспечивающее формирование четырехфазной последовательности импульсов со скважностью 0,5 на основе реверсивного сдвигового регистра. Устройство имеет один тактовый вход C, вход управления реверсом – N, вход отключения выходов – EZ и четыре выхода Q0, Q1, Q2, Q3. Обеспечить возможность реверса последовательности, самовосстановление состояния цифрового устройства в случае сбоя, защиту от возникновения одинакового состояния на противофазных выходах._____

4. Содержание пояснительной записки курсового проекта: 1. Построение таблицы состояний и диаграммы состояний; 2. Построение таблицы переходов; 3. Построение уравнений возбуждения; 4. Построение уравнений выхода. 5. Построение принципиальной схемы устройства.

5. Перечень графического материала: 1. Схема электрическая принципиальная – 1 л. А2. 2. Таблица «переход/выход», таблица возбуждения - 1 л. А2._____

Руководитель _____/Смирнов В.А./
(подпись, дата) (инициалы, фамилия)

Задание принял к исполнению _____
(подпись, дата)

Приложение 2

СПИСОК НОРМАТИВНОЙ ДОКУМЕНТАЦИИ

Общие правила выполнения конструкторской документации

1. ГОСТ 2.102-68 ЕСКД - Виды и комплекты конструкторской документации;
2. ГОСТ 2.104-68 ЕСКД - Основные надписи (1-1-73)*;
3. ГОСТ 2.105-79 ЕСКД - Основные требования к текстовым документам;
4. ГОСТ 2.106-68 ЕСКД - Текстовые документы;
5. ГОСТ 2.108-68 ЕСКД - Спецификация (1-1-73)*;

Общие правила выполнения чертежей

6. ГОСТ 2.301-68* ЕСКД - Форматы;
7. ГОСТ 2.302-68* ЕСКД - Масштабы;
8. ГОСТ 2.303-68* ЕСКД - Линии;
9. ГОСТ 2.304-81 ЕСКД - Шрифты чертежные;
10. ГОСТ 2.307-68* ЕСКД - Нанесение размеров и предельных отклонений;
11. ГОСТ 2.314-68* ЕСКД - Указания на чертежах о маркировании и клеймении изделий;
12. ГОСТ 2.316-68* ЕСКД - Правила нанесения на чертежах надписей, технических требований и таблиц;

Правила выполнения схем

13. ГОСТ 2.707-84 ЕСКД - Схемы. Виды и типы;
14. ГОСТ 2.702-75* ЕСКД - Правила выполнения электрических схем;
15. ГОСТ 2.705-70 ЕСКД - Правила выполнения электрических схем обмоток и изделий с обмотками;
16. ГОСТ 2.708-81 ЕСКД - Правила выполнения электрических схем цифровой вычислительной техники;
17. ГОСТ 2.709-79 ЕСКД (переиздание 1983 г. с изменением №1) - Система обозначений цепей в электрических схемах;
18. ГОСТ 2.710-81 ЕСКД - Обозначения буквенно-цифровые в электрических схемах;
19. ГОСТ 2.414-75 ЕСКД - Правила выполнения чертежей жгутов, кабелей, и проводов;
20. ГОСТ 2.415-68* ЕСКД - Правила выполнения чертежей изделий с электрическими обмотками;
21. ГОСТ 2.416-68* ЕСКД - Условные изображения сердечников магнитопроводов;

Графические обозначения в схемах

22. ГОСТ 2.721-74* ЕСКД - Обозначения общего применения;
23. ГОСТ 2.722-68* ЕСКД - Машины электрические;
24. ГОСТ 2.723-68* ЕСКД - Катушки индуктивности, дроссели, трансформаторы, автотрансформаторы и магнитные усилители;
25. ГОСТ 2.725-68* ЕСКД - Устройства коммутирующие;
26. ГОСТ 2.726-68* ЕСКД - Токосъемники;
27. ГОСТ 2.727-68* ЕСКД - Разрядники; предохранители;

28.ГОСТ 2.728-74* ЕСКД - Резисторы; конденсаторы;
29.ГОСТ 2.729-68** ЕСКД - Приборы электроизмерительные;
30.ГОСТ 2.730-73* ЕСКД - Приборы полупроводниковые;
31.ГОСТ 2.731-81 ЕСКД - Приборы электровакуумные;
32.ГОСТ 2.732-68* ЕСКД - Источники света;
33.ГОСТ 2.733-68* ЕСКД - Детекторы ионизирующих излучений;
38.ГОСТ 2.738-68* ЕСКД - Элементы телефонной аппаратуры;
39.ГОСТ 2.741-68* ЕСКД - Приборы акустические;
40.ГОСТ 2.742-68* ЕСКД - Источники тока электрохимические;
41.ГОСТ 2.743-82* ЕСКД - Элементы цифровой техники;
44.ГОСТ 2.747-68* ЕСКД - Размеры условных графических изображений;
45.ГОСТ 2.750-68 ЕСКД - Род тока и напряжения; виды соединения обмоток; формы импульсов;
46.ГОСТ 2.752-71* ЕСКД - Устройства телемеханики;
47.ГОСТ 2.755-74* ЕСКД - Устройства коммутационные и контактные соединения;
48.ГОСТ 2.756-76* ЕСКД - Воспринимающая часть электромеханических устройств;
49.ГОСТ 2.757-81 ЕСКД - Элементы коммутационного поля коммутационных систем;
50.ГОСТ 2.759-82 ЕСКД - Элементы аналоговой техники;

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Тульский государственный университет»

Технический колледж им. С.И. Мосина

Методические указания по выполнению самостоятельных работ по профессиональному модулю ПМ.01 ПРОЕКТИРОВАНИЕ ЦИФРОВЫХ СИСТЕМ основной профессиональной образовательной программы среднего профессионального образования – программы подготовки специалистов среднего звена

специальности
09.02.01 Компьютерные системы и комплексы

Тула 2023

УТВЕРЖДЕНЫ
Цикловой комиссией информационных технологий
Протокол от «13» января 2023 г. № 6
Председатель цикловой комиссии И.В. Миляева

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью и задачами изучения профессионального модуля (дисциплины) является формирование у студентов общих и профессиональных компетенций, знаний, умений и практического опыта, указанных в разделах 1, 4 рабочей программы профессионального модуля (дисциплины). В рамках достижения указанной цели учебным планом и рабочей программой предусмотрена самостоятельная работа студентов.

Самостоятельная работа студентов является одной из важнейших составляющих образовательного процесса. Независимо от полученной специальности и характера работы любой начинающий специалист должен обладать фундаментальными знаниями, профессиональными умениями и навыками деятельности своей квалификации, опытом творческой и исследовательской деятельности по решению новых проблем, опытом социально-оценочной деятельности.

Все эти составляющие образования формируются именно в процессе самостоятельной работы студентов, так как предполагает максимальную индивидуализацию деятельности каждого студента и может рассматриваться одновременно и как средство совершенствования творческой индивидуальности.

Основным принципом организации самостоятельной работы студентов является комплексный подход, направленный на формирование навыков репродуктивной и творческой деятельности студента в аудитории, при внеаудиторных контактах с преподавателем на консультациях и домашней подготовке.

Самостоятельная работа студента предполагает углубленное изучение тем, входящих в содержание профессионального модуля (дисциплины) (пункт 2.2 рабочей программы) и выполнение дополнительных заданий теоретического и практического характера.

Среди основных видов самостоятельной работы студентов традиционно выделяют: подготовка к лекциям, лабораторным работам и практическим занятиям, зачетам и экзаменам, презентациям и докладам; написание рефератов, выполнение лабораторных и контрольных работ.

1 Виды и формы организации самостоятельной работы студентов

Любой вид занятий, создающий условия для зарождения самостоятельной мысли, познавательной и творческой активности студента связан с самостоятельной работой. В широком смысле под самостоятельной работой понимают совокупность всей самостоятельной деятельности студентов как в учебной аудитории, так и вне ее, в контакте с преподавателем и в его отсутствие.

Самостоятельная работа может реализовываться:

- непосредственно в процессе аудиторных занятий – на лекциях, лабораторных работах, практических занятиях, при выполнении контрольных и лабораторных работ и др.;
- в контакте с преподавателем вне рамок аудиторных занятий – на консультациях по учебным вопросам, в ходе творческих контактов, при ликвидации задолженностей, при выполнении индивидуальных заданий и т.д.;
- в библиотеке, дома, в общежитии и других местах при выполнении студентом учебных и творческих заданий.

Цель самостоятельной работы студента – осмысленно и самостоятельно работать сначала с учебным материалом, заложить основы самоорганизации и самовоспитания с тем, чтобы привить умение в дальнейшем непрерывно повышать свою профессиональную квалификацию.

В учебном процессе выделяют два вида самостоятельной работы:

- аудиторная – самостоятельная работа выполняется на учебных занятиях под непосредственным руководством преподавателя и по его заданию;
- внеаудиторная – самостоятельная работа выполняется студентом по заданию преподавателя, но без его непосредственного участия.

Содержание аудиторной и внеаудиторной самостоятельной работы студентов определяется в соответствии с рекомендуемыми видами учебных заданий, представленными в рабочей программе профессионального модуля (дисциплины).

Самостоятельная работа помогает студентам:

1) овладеть знаниями:

- чтение текста (учебника, первоисточника, дополнительной литературы и т.д.);
- составление плана текста, графическое изображение структуры текста, конспектирование текста, выписки из текста и т.д.;
- работа со справочниками и др. справочной литературой;
- ознакомление с нормативными и правовыми документами;
- учебно-методическая и научно-исследовательская работа;
- использование компьютерной техники и Интернета и др.;

2) закреплять и систематизировать знания:

- работа с конспектом лекции;

- обработка текста, повторная работа над учебным материалом учебника, первоисточника, дополнительной литературы, аудио и видеозаписей;

- подготовка плана;

- составление таблиц для систематизации учебного материала;

- подготовка ответов на контрольные вопросы;

- заполнение рабочей тетради;

- аналитическая обработка текста;

- подготовка мультимедиа презентации и докладов к выступлению на семинаре (конференции, круглом столе и т.п.);

- подготовка реферата;

- составление библиографии использованных литературных источников;

- разработка тематических кроссвордов и ребусов;

- тестирование и др.;

3) формировать умения:

- решение ситуационных задач и упражнений по образцу;

- выполнение расчетов (графические и расчетные работы);

- решение профессиональных кейсов и вариативных задач;

- подготовка к контрольным работам;

- подготовка к тестированию;

- подготовка к деловым играм;

- проектирование и моделирование разных видов и компонентов профессиональной деятельности;

- опытно-экспериментальная работа;

- анализ профессиональных умений с использованием аудио- и видеотехники и др.

Самостоятельная работа может осуществляться индивидуально или группами студентов в зависимости от цели, объема, конкретной тематики самостоятельной работы, уровня сложности и уровня умений студентов.

Контроль результатов самостоятельной работы студентов должен осуществляться в пределах времени, отведенного на обязательные учебные занятия и внеаудиторную самостоятельную работу студентов по профессиональному модулю (дисциплине), может проходить в письменной, устной или смешанной форме.

Формы самостоятельной работы студента могут различаться в зависимости от цели, характера, профессионального модуля (дисциплины), объема часов, определенных учебным планом: подготовка к лекциям, семинарским, практическим и лабораторным занятиям; изучение учебных пособий; изучение и конспектирование хрестоматий и сборников документов; изучение в рамках программы курса тем и проблем, не выносимых на лекции и семинарские занятия; написание тематических докладов, рефератов и эссе на проблемные темы; аннотирование монографий или их отдельных глав, статей; выполнение исследовательских и творческих

заданий; написание контрольных и лабораторных работ; составление библиографии и реферирование по заданной теме.

2 Требования к организации самостоятельной работы студентов при подготовке к аудиторным занятиям

2.1. Подготовка к лекциям

Главное в период подготовки к лекционным занятиям – научиться методам самостоятельного умственного труда, сознательно развивать свои творческие способности и овладевать навыками творческой работы. Для этого необходимо строго соблюдать дисциплину учебы и поведения. Четкое планирование своего рабочего времени и отдыха является необходимым условием для успешной самостоятельной работы.

В основу его нужно положить рабочие программы изучаемых в семестре дисциплин.

Каждому студенту следует составлять еженедельный и семестровый планы работы, а также план на каждый рабочий день. С вечера всегда надо распределять работу на завтрашний день. В конце каждого дня целесообразно подводить итог работы: тщательно проверить, все ли выполнено по намеченному плану, не было ли каких-либо отступлений, а если были, по какой причине это произошло. Нужно осуществлять самоконтроль, который является необходимым условием успешной учебы. Если что-то осталось невыполненным, необходимо изыскать время для завершения этой части работы, не уменьшая объема недельного плана.

Самостоятельная работа на лекции

Слушание и запись лекций – сложный вид аудиторной работы. Внимательное слушание и конспектирование лекций предполагает интенсивную умственную деятельность студента. Краткие записи лекций, их конспектирование помогает усвоить учебный материал. Конспект является полезным тогда, когда записано самое существенное, основное и сделано это самим студентом.

Не надо стремиться записать дословно всю лекцию. Такое «конспектирование» приносит больше вреда, чем пользы. Запись лекций рекомендуется вести по возможности собственными формулировками. Желательно запись осуществлять на одной странице, а следующую оставлять для проработки учебного материала самостоятельно в домашних условиях.

Конспект лекции лучше подразделять на пункты, параграфы, соблюдая красную строку. Этому в большой степени будут способствовать пункты плана лекции, предложенные преподавателям. Принципиальные места, определения, формулы и другое следует сопровождать замечаниями «важно», «особо важно», «хорошо запомнить» и т.п. Можно делать это и с помощью разноцветных маркеров или ручек. Лучше если они будут собственными, чтобы не приходилось просить их у однокурсников и тем самым не отвлекать их во время лекции.

Целесообразно разработать собственную «маркографию» (значки, символы), сокращения слов. Не лишним будет и изучение основ стенографии. Работая над конспектом лекций, всегда необходимо использовать не только учебник, но и ту литературу, которую дополнительно рекомендовал лектор. Именно такая серьезная, кропотливая работа с лекционным материалом позволит глубоко овладеть знаниями.

2.2 Подготовка отчета по лабораторно-практической работе

Отчёт по лабораторно-практической работе должен иметь структуру:

- тема;
- цель;
- ход работы;
- результаты работы;
- ответы на контрольные вопросы.

Критерии оценки работы:

- использование рабочего времени;
- уровень знания возможностей конкретной программы;
- последовательность выполнения задания;
- самостоятельность в работе;
- форма фиксации результатов работы;
- достаточная полнота и формулировка ответов на контрольные вопросы.

Работа, сделанная не по своему варианту или копирующая работу другого студента, засчитывается не будет вне зависимости от качества её выполнения.

2.3 Подготовка презентации и доклада

Презентация, согласно толковому словарю русского языка Д.Н. Ушакова: «… способ подачи информации, в котором присутствуют рисунки, фотографии, анимация и звук».

Для подготовки презентации рекомендуется использовать: Libre Office Impress, MS Power Point, текстовый процессор, Acrobat Reader.

Для подготовки презентации необходимо собрать и обработать начальную информацию. Последовательность подготовки презентации:

1. Четко сформулировать цель презентации: вы хотите свою аудиторию мотивировать, убедить, заразить какой-то идеей или просто формально отчитаться.

2. Определить каков будет формат презентации: живое выступление (тогда, сколько будет его продолжительность) или электронная рассылка (каков будет контекст презентации).

3. Отобрать всю содержательную часть для презентации и выстроить логическую цепочку представления.

4. Определить ключевые моменты в содержании текста и выделить их.

5. Определить виды визуализации (картинки) для отображения их на слайдах в соответствии с логикой, целью и спецификой материала.

6. Подобрать дизайн и форматировать слайды (количество картинок и текста, их расположение, цвет и размер).

7. Проверить визуальное восприятие презентации.

К видам визуализации относятся иллюстрации, образы, диаграммы, таблицы. Иллюстрация – представление реально существующего зрительного ряда. Образы – в отличие от иллюстраций – метафора. Их назначение – вызвать эмоцию и создать отношение к ней, воздействовать на аудиторию. С помощью хорошо продуманных и представляемых образов, информация может надолго остаться в памяти человека. Диаграмма – визуализация количественных и качественных связей. Их используют для убедительной демонстрации данных, для пространственного мышления в дополнение к логическому. Таблица – конкретный, наглядный и точный показ данных. Ее основное назначение – структурировать информацию, что порой облегчает восприятие данных аудиторией.

Практические советы по подготовке презентации

- готовьте отдельно: печатный текст + слайды + раздаточный материал;
- слайды – визуальная подача информации, которая должна содержать минимум текста, максимум изображений, несущих смысловую нагрузку, выглядеть наглядно и просто;
- текстовое содержание презентации – устная речь или чтение, которая должна включать аргументы, факты, доказательства и эмоции;
- рекомендуемое число слайдов 10-12;
- обязательная информация для презентации: тема, фамилия и инициалы выступающего; план сообщения; краткие выводы из всего сказанного; список использованных источников;
- раздаточный материал – должен обеспечивать ту же глубину и охват, что и живое выступление: люди больше доверяют тому, что они могут унести с собой, чем исчезающим изображениям, слова и слайды забываются, а раздаточный материал остается постоянным осязаемым напоминанием; раздаточный материал важно раздавать в конце презентации; раздаточный материалы должны отличаться от слайдов, должны быть более информативными.

Доклад, согласно толковому словарю русского языка Д.Н. Ушакова: «… сообщение по заданной теме, с целью внести знания из дополнительной литературы, систематизировать материл, проиллюстрировать примерами, развивать навыки самостоятельной работы с научной литературой, познавательный интерес к научному познанию».

Тема доклада должна быть согласованна с преподавателем и соответствовать теме учебного занятия. Материалы при его подготовке, должны соответствовать научно-методическим требованиям и быть указаны в докладе. Необходимо соблюдать регламент, оговоренный при получении задания. Иллюстрации должны быть достаточными, но не чрезмерными.

Работа студента над докладом-презентацией включает отработку умения самостоятельно обобщать материал и делать выводы в заключении, умения ориентироваться в материале и отвечать на дополнительные вопросы слушателей, отработку навыков ораторства, умения проводить диспут.

Докладчики должны знать и уметь: сообщать новую информацию; использовать технические средства; хорошо ориентироваться в теме всего семинарского занятия; дискутировать и быстро отвечать на заданные вопросы; четко выполнять установленный регламент (не более 10 минут); иметь представление о композиционной структуре доклада и др.

Структура выступления

Вступление помогает обеспечить успех выступления по любой тематике. Вступление должно содержать: название, сообщение основной идеи, современную оценку предмета изложения, краткое перечисление рассматриваемых вопросов, живую интересную форму изложения, акцентирование внимания на важных моментах, оригинальность подхода.

Основная часть, в которой выступающий должен глубоко раскрыть

суть затронутой темы, обычно строится по принципу отчета. Задача основной части – представить достаточно данных для того, чтобы слушатели заинтересовались темой и захотели ознакомиться с материалами. При этом логическая структура теоретического блока не должны даваться без наглядных пособий, аудио-визуальных и визуальных материалов.

Заключение – ясное, четкое обобщение и краткие выводы, которых всегда ждут слушатели.

2.4. Подготовка к зачету и экзамену

Каждый учебный семестр заканчивается сессией. Подготовка к сессии, выполнение контрольной работы, сдача зачетов и экзаменов является также самостоятельной работой студента. Основное в подготовке к сессии – повторение всего учебного материала междисциплинарного курса, профессионального модуля (дисциплины).

Только тот студент успевает, кто хорошо усвоил учебный материал. Если студент плохо работал в семестре, пропускал лекции, слушал их невнимательно, не конспектировал, не изучал рекомендованную литературу, то в процессе подготовки к сессии ему придется не повторять уже знакомое, а заново в короткий срок изучать весь учебный материал. Все это зачастую невозможно сделать из-за нехватки времени.

Для такого студента подготовка к зачету или экзамену будет трудным, а иногда и непосильным делом, а конечный результат – возможное отчисление из учебного заведения.

3 Требования к студентам при подготовке письменных работ

3.1. Подготовка реферата

Реферат – письменный доклад по определенной теме, в котором собрана информация из одного или нескольких источников. Рефераты

пишутся обычно стандартным языком, с использованием типологизированных речевых оборотов вроде: «важное значение имеет», «уделяется особое внимание», «поднимается вопрос», «делаем следующие выводы», «исследуемая проблема», «освещаемый вопрос» и т.п.

К языковым и стилистическим особенностям рефератов относятся слова и обороты речи, носящие обобщающий характер, словесные клише. У рефератов особая логичность подачи материала и изъяснения мысли, определенная объективность изложения материала.

Признаки реферата

Реферат не копирует дословно содержание первоисточника, а представляет собой новый вторичный текст, создаваемый в результате систематизации и обобщения материала первоисточника, его аналитико-синтетической переработки.

Будучи вторичным текстом, реферат составляется в соответствии со всеми требованиями, предъявляемыми к связанному высказыванию: так ему присущи следующие категории: оптимальное соотношение и завершенность (смысловая и жанрово-композиционная). Для реферата отбирается информация, объективно-ценная для всех читающих, а не только для одного автора. Автор реферата не может пользоваться только ему понятными значками, пометами, сокращениями.

Работа, проводимая автором для подготовки реферата должна обязательно включать самостоятельное мини-исследование, осуществляемое студентом на материале или художественных текстов по литературе, или архивных первоисточников по истории и т.п.

Организация и описание исследования представляет собой очень сложный вид интеллектуальной деятельности, требующий культуры научного мышления, знания методики проведения исследования, навыков оформления научного труда и т.д. Мини-исследование раскрывается в реферате после глубокого, полного обзора научной литературы по проблеме исследования.

В зависимости от количества реферируемых источников выделяют следующие виды рефератов:

- монографические – рефераты, написанные на основе одного источника;
- обзорные – рефераты, созданные на основе нескольких исходных текстов, объединенных общей темой и сходными проблемами исследования.

Структура реферата

1. Титульный лист
2. Содержание
3. Введение
4. Основная часть
5. Заключение
6. Список использованных источников
7. Приложения

Титульный лист является первой страницей и заполняется по строго определенным правилам (Приложение А).

После титульного листа помещают содержание, в котором приводятся все заголовки работы и указываются страницы, с которых они начинаются. Заголовки оглавления должны точно повторять заголовки в тексте. Сокращать их или давать в другой формулировке и последовательности нельзя. Все заголовки начинаются с прописной буквы без точки на конце. Последнее слово каждого заголовка соединяют отточием с соответствующим ему номером страницы в правом столбце оглавления. Заголовки одинаковых ступеней рубрикации необходимо располагать друг под другом.

Введение к реферату – важнейшая его часть. Здесь обычно обосновывается актуальность выбранной темы, цель и задачи, краткое содержание, указывается объект рассмотрения, приводится характеристика источников для написания работы и краткий обзор имеющейся по данной теме литературы. Актуальность предполагает оценку своевременности и социальной значимости выбранной темы, обзор литературы по теме отражает знакомство автора с имеющимися источниками, умение их систематизировать, критически рассматривать, выделять существенное, определять главное.

Основная часть. Основная часть реферата структурируется по главам и параграфам (пунктам и подпунктам), количество и название которых определяются автором. Содержание глав основной части должно точно соответствовать теме работы и полностью ее раскрывать. Данные главы должны показать умение студента сжато, логично и аргументировано излагать материал, обобщать, анализировать и делать логические выводы. Основная часть реферата, помимо почерпнутого из разных источников содержания, должна включать в себя собственное мнение студента и сформулированные выводы, опирающиеся на приведенные факты.

В основной части реферата обязательными являются ссылки на авторов, чьи позиции, мнения, информация использованы в реферате. Ссылки на источники могут быть выполнены по тексту работы постранично в нижней части страницы (фамилия автора, его инициалы, полное название работы, год издания и страницы, откуда взята ссылка) или в конце цитирования - тогда достаточно указать в квадратных скобках номер литературного источника из списка использованной литературы с указанием конкретных страниц, откуда взята ссылка, (например, [7, с. 67–89]). Номер литературного источника должен указываться после каждого нового отрывка текста из другого литературного источника.

Цитирование и ссылки не должны подменять позиции автора реферата. Излишняя высокопарность, злоупотребления терминологией, объемные отступления от темы, несоразмерная растянутость отдельных глав, разделов, параграфов рассматриваются в качестве недостатков основной части реферата.

Заключительная часть предполагает последовательное, логически стройное изложение обобщенных выводов по рассматриваемой теме.

Заключение не должно превышать объем 2 страниц и не должно слово в слово повторять уже имеющийся текст, но должно отражать собственные выводы о проделанной работе, а может быть, и о перспективах дальнейшего исследования темы. В заключении целесообразно сформулировать итоги выполненной работы, краткого и четкого изложить выводы, представить анализ степени выполнения поставленных во введении задач и указать то новое, что лично для себя студент вынес из работы над рефератом.

Список использованных источников составляет одну из частей работы, отражающую самостоятельную творческую работу автора, и позволяет судить о степени фундаментальности данного реферата. В список использованной литературы необходимо внести все источники, которые были изучены студентами в процессе написания реферата.

В работах используются следующие способы построения библиографических списков: по алфавиту фамилий авторов или заглавий; по тематике; по видам изданий; по характеру содержания; списки смешанного построения. Литература в списке указывается в алфавитном порядке (более распространенный вариант – фамилии авторов в алфавитном порядке), после указания фамилии и инициалов автора указывается название литературного источника без кавычек, место издания и название издательства – при города Москва и Санкт-Петербург как место издания обозначаются сокращенно – М.; СПб., название других городов пишется полностью. (М.: Академия), год издания, страницы – общее количество или конкретные.

Список использованных источников, приводится в следующей последовательности: 1) законодательные акты (в хронологическом порядке);

2) статистические материалы и нормативные документы (в хронологическом порядке); 3) литературные источники (в алфавитном порядке) – книги, монографии, учебники и учебные пособия, периодические издания, зарубежные источники, Интернет-источники. Например:

1. Указ Президента РФ “О защите потребителей от недобросовестной рекламы” от 10.06.94 г. № 1183// Российская газета. 1994. 16 июня. № 112.

2. Блинова М.С. Социология миграции: история становления и перспективы развития: учебное пособие/ М.С. Блинова. – М.: КДУ, 2009. – 192 с.

Для работ из журналов и газетных статей необходимо указать фамилию и инициалы автора, название статьи, а затем наименование источника со всеми элементами титульного листа, после чего указать номер страницы начала и конца статьи. Например:

1. Петренко К.В. Демографические характеристики трудового потенциала нефтегазодобывающих регионов Севера России// Научное обозрение. Серия 2. Гуманитарные науки. – М., 2012. – № 5. – С. 85 – 89.

2. Артемьев З. Мигрантам дадут на работу три года// Вечерняя Москва. 2013. № 184

21

(26509).

Для Интернет-источников необходимо указать название работы, источник работы и сайт. Например:

1. О мерах по созданию и развитию малых предприятий [Электронный ресурс]: постановление Совета министров СССР от 8 авг. 1990 г. № 790. – Режим доступа:

[14.05.2012]// http://www.consultant.ru. – Загл. с экрана.

2. Информационные ресурсы справочно-поисковой системы Рамблер - //http:// www.rambler.ru

После списка использованных источников могут быть помещены различные приложения (таблицы, графики, диаграммы, иллюстрации и пр.). В приложение рекомендуется выносить информацию, которая загромождает текст реферата и мешает его логическому восприятию. В содержательной части работы эта часть материала должна быть обобщена и представлена в сжатом виде. На все приложения в тексте реферата должны быть ссылки. Каждое приложение нумеруется и оформляется с новой страницы.

Требования к оформлению реферата

Работа выполняется на компьютере (гарнитура Times New Roman, шрифт 14) через 1,5 интервала с полями: верхнее, нижнее – 2; левое – 3; правое – 1,5. Отступ первой строки абзаца – 1,25. Сноски – постраничные (шрифт 12), их нумерация должна быть сквозной по всему тексту реферата.

Нумерация страниц должна быть сквозной (номер не ставится на титульном листе, но в общем количестве страниц учитывается).

Таблицы и рисунки встраиваются в текст работы, их нумерация должна быть сквозной по всему реферату. Они все должны иметь название и в самом тексте реферата на них должна быть ссылка. (Например: Как следует из таблицы 1 общая численность безработных в первое десятиление XXI века в разрезе ряда европейских стран резко увеличивалась). После названия таблицы и рисунка точка не ставится.

Общее количество страниц в реферате, без учета приложений, не должно превышать 15 страниц. Значительное превышение установленного объема является недостатком работы и указывает на то, что студент не сумел отобрать и переработать необходимый материал.

В приложении помещают вспомогательные или дополнительные материалы, которые загромождают текст основной части работы (таблицы, рисунки, карты, графики, неопубликованные документы, переписка и т.д.).

Каждое приложение должно начинаться с новой страницы с указанием в центре верхней строки слова «ПРИЛОЖЕНИЕ», иметь номер и тематический заголовок. При наличии в работе более одного приложения они нумеруются русскими буквами (без знака «№»). Нумерация страниц, на которых даются приложения, должна быть сквозной и продолжать общую нумерацию страниц основного текста. Связь основного текста с приложениями осуществляется через ссылки, которые помещаются в круглые скобки – например, (ПРИЛОЖЕНИЕ А).

ПРИЛОЖЕНИЕ А

Пример оформления титульного листа реферата

Министерство науки и высшего образования Российской Федерации
ФГБОУ ВО «Тульский государственный университет»
Технический колледж им. С.И. Мосина

РЕФЕРАТ

по МДК «……»

на тему: « »

Автор работы,
студент гр._____ **А.А.Петров**

Руководитель,
преподаватель **П.П.Иванов**

Тула 20__