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

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по выполнению лабораторныхи самостоятельных работ

по междисциплинарному курсу
МДК 01.02 «Прикладное программирование»

профессионального модуля ПМ.01 «Разработка программных модулей программного обеспечения для компьютерных систем»

специальности 09.02.03 Программирование в компьютерных системах

Тула 2020

УТВЕРЖДЕНЫ

Цикловой комиссий информационных технологий

Протокол от «15» января 2020 г. № 6

Председатель цикловой комиссии И.В.Миляева

Авторы: Туляков С.П., преподаватель

Лабораторная работа №1. (2 часа)

Знакомство с интерфейсом интегрированной среды разработки Visual Studio .NET.

Настройка среды разработки

Среда разработки Visual Studio .NET, как и среды всех современных средств разработки, может быть настроена согласно потребностям конкретного пользователя.

В этом разделе мы рассмотрим различные способы настройки среды Visual Studio .NET, а также узнаем, какие окна и инструменты она содержит.

При первом запуске Visual Studio .NET можно настроить среду разработчика в соответствии с типом задач, которые предстоит решать с ее помощью. Сведения об этом вводятся на странице My Profile

Поля на этой странице имеют следующее назначение:

- Profile — позволяет указать, какой тип приложений разрабатывается наиболее часто (приложения Visual Studio, приложения Visual Basic, Visual C++, Visual InterDev, Visual C# и ряд других);
- Keyboard Scheme — позволяет указать, какое предпочтительный набор горячих клавиш. Возможные значения: Visual Studio Default (набор, принятый по умолчанию), Visual Basic 6.0, Visual C++ 2.0, Visual C++ 6.0, Visual Studio 6.0 и др.;
- Window Layout — позволяет определить предпочтительный способ расположения окон на экране. Возможные значения: Visual Studio Default (набор, принятый по умолчанию), Visual Basic 6.0, Visual C++ 6.0, Student Window Layout (расположение, рекомендуемое начинающим пользователям), No Tools Window Layout (расположение, в котором скрыто большинство окон и инструментов);
- Help Filter — позволяет установить фильтры для справочной системы. Эти фильтры определяют, на каком языке программирования отображать примеры для справочной системы, а также какие части справочной системы отображаются и участвуют в поиске по ключевым словам;
- Show Help — позволяет указать, как отображать окно справочной системы: внутри среды разработки или в отдельном окне;
- At Startup — позволяет выбрать, что именно отображать при запуске Visual Studio .NET. Возможные значения: Visual Studio Start Page (стартовая страница), Most Recent Solution (последний проект, над которым велась работа), диалоговая панель Open Project, диалоговая панель New Project, просто среда разработки без загруженного проекта.

Стартовая страница

Если в диалоговой панели My Profile в поле At Startup указано Show Start Page, при последующих запусках Visual Studio можно увидеть экран стартовой страницы.

В левой части этой страницы находятся ссылки на Web-ресурсы, такие как страница с обновлениями и дополнениями к продукту (What's New), ссылки на страницы сообществ разработчиков, новостей MSDN и поиска в MSDN нужных разделов. Там же можно открыть страницу My Profile. В правой части страницы можно выбрать один из проектов, над которым недавно велась работа, открыть произвольный проект или создать новый.

Создание нового проекта

Если в меню среды разработки выбрать пункт File | New | Project, на экране появится диалоговая панель New Project. Приложение в Visual Studio .NET может состоять из нескольких проектов, совокупность которых называется термином Solution (решение).

В левой части этой диалоговой панели можно выбрать тип проекта. В общем случае можно выбрать проекты, созданные на языках программирования Visual Basic .NET, C#, C++, а также на ряде других. Этот список зависит от того, какие языки были выбраны при установке Visual Studio, а также от того, были ли приобретены и установлены дополнительные языки программирования сторонних производителей.

В правой части экрана можно выбрать один из предложенных шаблонов для данного типа проектов:

- Windows Application — шаблон для Windows-приложения;
- Class Library — шаблон для создания библиотеки классов, которая будет использоваться другими приложениями;
- Control Library — шаблон для создания элементов управления, которые будут использоваться в приложениях с графическим пользовательским интерфейсом для платформы Windows (называемых также приложениями Windows Forms);
- ASP.NET Web Application — шаблон для создания Web-приложений ASP.NET;
- ASP.NET Web Service — шаблон для создания Web-сервисов;
- Web Control Library — шаблон для создания элементов управления, которые будут использоваться в Web-приложениях;
- Console Application — шаблон для создания консольных приложений;
- Windows Service — шаблон для создания сервисов операционной системы;
- Empty Project/Empty Web Project — проект, который создается без использования шаблонов;
- New Project In Existing Folder — добавить новый проект в уже существующую папку.

При создании нового проекта в поле Location необходимо указать имя каталога, в котором следует сохранить его файлы. При этом в данном каталоге автоматически будет создан другой каталог, имя которого совпадает с именем проекта. Например, при создании проекта MyProject и указании в поле Location каталога C:\Projects соответствующее решение будет создано в каталоге C:\Projects\MyProject\ MyProject.sln. По умолчанию проекты сохраняются в файле My Documents\Visual Studio Projects\Имя проекта.

Окна среды разработки Visual Studio

В данном разделе мы рассмотрим окна, имеющиеся в среде разработки (Integrated Development Environment, IDE) Visual Studio .NET. Как уже было сказано выше, вид среды разработки Visual Studio .NET зависит от того, какие настройки были указаны в окне MyProfile.

Как и большинство современных приложений, среда разработки Visual Studio .NET содержит меню и набор инструментальных панелей. В левой части среды разработки присутствует элемент управления со значком окна Server Explorer; это окно появится, если указатель мыши окажется над данным значком. Там же имеется и значок окна Toolbox — оно появится, если поместить указатель мыши над этим значком.

В правой части экрана находится окно Solution Explorer. В нем можно увидеть, из каких проектов состоит решение и какие файлы входят в состав этих проектов.

Ниже окна Solution Explorer расположено окно свойств (Properties). Это окно содержит список атрибутов объекта, выделенного в данный момент.

Давайте выясним, зачем нужны эти и другие окна среды разработки.

Окно Toolbox

В окне Toolbox (его можно отобразить на экране с помощью команды меню View | Toolbox) находится список элементов управления, которые можно использовать на формах приложения. То, какой набор компонентов доступен в данный момент, зависит от типа разрабатываемого приложения. Например, если в данный момент разрабатывается приложение типа Windows Forms, в этом окне будут присутствовать элементы управления, которые можно использовать в Windows-приложениях; если же разрабатывается Web-форма, в этом окне будут находиться инструменты для работы с элементами управления Web Controls, и т.д.

При необходимости можно изменить отображаемый в окне Toolbox набор элементов управления, добавив другие компоненты .NET или элементы ActiveX (в том числе созданные независимыми производителями). Для этой цели можно использовать команду меню Tools | Customize Toolbox и с помощью диалоговой панели Customize Toolbox выбрать элементы управления ActiveX или элементы управления .NET, которые мы хотим отобразить в окне Toolbox.

Окно Solution Explorer

Как мы уже знаем, решение — это набор проектов, из которых состоит приложение. Окно Solution Explorer (которое можно отобразить на экране с помощью команды меню View | Solution Explorer) позволяет просматривать состав проектов, входящих в решение, в виде иерархической структуры, а также связи между проектами и их компонентами . Компонентами проектов могут быть формы, классы, модули, а также другие файлы, которые требуются для создания приложения. Если нужно отредактировать компонент проекта, следует дважды щелкнуть по его имени в окне Solution Explorer.

С помощью кнопок, расположенных в верхней части окна Solution Explorer, можно указать, что именно должно отражаться в среде разработки:

- View Code — код, связанный с файлом, выделенным в окне Solution Explorer;
- View Designer — дизайнер (визуальный редактор) файла, выделенного в окне Solution Explorer;
- Refresh — обновить содержимое окна Solution Explorer;
- Show All Files — все файлы, включая код, связанный с формами;
- Properties — свойства выбранного файла.

Окно Class View

Окно Class View (доступно с помощью команды меню View | Class View) позволяет просмотреть список свойств и методов созданных в приложении классов. Выбрав свойство или метод, можно щелкнуть на его имени правой клавишей мыши и выбрать одно из возможных действий с данным свойством или методом. По двойному щелчку по имени класса произойдет его загрузка в редактор кода.

Окно Server Explorer

Окно Server Explorer (команда меню View | Server Explorer, позволяет просматривать сведения о службах, выполняющихся на конкретных серверах. К таким службам, в частности, относятся службы Crystal Reports Services, журнал событий, очереди сообщений, службы серверов баз данных, таких как Microsoft SQL Server.

Большинство этих служб представлено в окне в виде иерархического дерева, позволяющего просматривать сведения, связанные с данными службами, и иногда добавлять новые элементы. С помощью перетаскивания значка службы или ее элемента в дизайнер можно организовать ее использование в приложении. Так, при переносе значка таблицы сервера баз данных на форму разрабатываемого приложения можно создать компонент DataAdapter для извлечения данных из этой таблицы.

Окно Properties

Окно Properties (команда меню View | Properties Window) предназначено для изменения свойств элементов управления и других классов создаваемого приложения. Свойства можно отсортировать по алфавиту или по категориям (для этой цели в верхней части

этого окна имеются соответствующие кнопки). Редактирование свойств может осуществляться путем ввода значения, выбора его из выпадающего списка либо с помощью установки его значения в отдельной диалоговой панели — это зависит от типа конкретного свойства. Таким же образом можно изменять свойства проекта, приложения и т.п.

Окно Object Browser

Окно Object Browser, доступное с помощью команды меню View | Other Windows | Object Browser, так же как и окно Class View, позволяет просмотреть список классов, их свойств и методов. Однако Object Browser позволяет просмотреть все компоненты, на которые ссылается класс, а также, при необходимости, компоненты, на которые нет ссылок в данном проекте, тогда как с помощью окна Class View можно просматривать сведения только о классах из данного проекта. С помощью Object Browser можно также просмотреть объявления свойств и методов.

Окно Task List

Окно Task List, доступное с помощью команды меню View | Other Windows | Task List, содержит список задач (TO DO list), ошибок компиляции и другую информацию. В этот список можно внести свое задание, щелкнув по надписи Click here to add a new task, добавив в код комментарий вида «TODO: <текст задачи>» либо выбрав из контекстного меню строки кода пункт Add a Task List Shortcut. При щелчке по тексту задачи в этом окне редактор кода откроется в месте, где находится соответствующий комментарий.

Режимы работы среды разработки

Среда разработки Visual Studio .NET содержит два типа окон — окна инструментов и окна документов. Окна инструментов (часть из которых была описана выше) доступны с помощью команд меню View и некоторых других, и их доступность зависит от типа приложения и от того, какие модули расширения (дополнительные утилиты и инструменты, в том числе произведенные сторонними разработчиками) добавлены к среде разработки. В окнах же документов можно редактировать компоненты проектов.

Окна инструментов

С окнами инструментов можно производить различные манипуляции. В частности, можно заставить их автоматически появляться и исчезать, группировать их в виде многостраничного блокнота, варьировать их расположение в среде разработки, делать их «плавающими» и даже отображать на дополнительном мониторе, если использование такового поддерживается операционной системой.

Некоторые окна инструментов, например окно Web Browser, можно создавать в виде нескольких экземпляров (это можно сделать, выбрав пункт меню Windows | New Window). Можно также заставить окна инструментов автоматически исчезать, если они в данный момент не являются активными, — в этом случае на экране отображаются название и пиктограмма окна, над которой можно поместить указатель мыши, если окно нужно отобразить целиком. Если необходимо предотвратить исчезновение окна с экрана, следует щелкнуть мышью по изображению канцелярской кнопки на заголовке окна.

Режимы отображения окон инструментов

Visual Studio .NET поддерживает два режима отображения окон документов в среде разработки: многодокументный интерфейс (Multiple Document Interface, MDI) и интерфейс в виде блокнота (Tabbed Documents). Выбрать нужный режим отображения можно в разделе Environment | General диалоговой панели Options (ее можно отобразить с помощью команды меню Tools | Options).

Окна документов

Окна документов предназначены для редактирования компонентов проектов. Их взаимное расположение зависит от выбранного режима отображения окон в среде разработки.

Редактор кода

Ниже будут рассмотрены возможности, предоставляемые редактором кода Visual Studio .NET.

Как и в прежней версии Visual Studio, после набора имени объекта и ввода точки на экране появляется список свойств и методов данного объекта. При вводе имени метода можно увидеть на экране описание метода и его параметров.

Окно редактирования можно разбить на несколько частей, в которых будут отображаться разные фрагменты кода. Допустимо также отобразить второе окно редактирования с помощью пункта меню Window | New Window.

В редакторе кода можно осуществлять контекстный поиск и замену текста в текущей процедуре, текущем модуле или в выделенном фрагменте кода с помощью стандартной диалоговой панели Windows Find and Replace. В строке для поиска могут содержаться символы «*» и «?», означающие любую последовательность символов и любой символ соответственно.

Возможен также поиск и замена фрагментов текста во всех файлах проекта. В этом случае следует использовать диалоговые панели Find in Files и Replace in Files.

Помимо фрагментов кода можно искать также названия классов и структур — для этой цели используется диалоговая панель Find Symbols. Результаты поиска отображаются в окне Find Symbol Results.

В редакторе кода можно установить закладку на какую-либо строку кода и вернуться к ней позже. Закладки не исчезают и при сохранении проекта.

Можно также создать комментарий, связанный с выделенным фрагментом текста, с помощью команды меню Edit | Advanced | Comment Selection.

Возможно перемещение фрагментов текста посредством мыши в другое место, копирование фрагментов, а также перемещение фрагментов текста из редактора кода в окно Toolbox.

Фрагменты текста в окне Toolbox сохраняются до закрытия Visual Studio и могут при необходимости быть перенесены в редактор кода.

С помощью комбинации клавиш Shift+Ctrl+ Enter можно развернуть окно редактора кода на весь экран и вернуть его в исходное состояние, а комбинации Ctrl+Tab и Shift+Ctrl+Tab позволяют перемещаться между окнами документов.

Буфер обмена

Visual Studio .NET позволяет сохранить до 20 фрагментов текста или иных элементов проектов в буфере обмена (Clipboard Ring). Удерживая клавиши Ctrl и Shift, можно просматривать по очереди элементы, хранящиеся в буфере обмена, нажимая на клавишу V и поочередно отображая эти элементы в редакторе кода. После нахождения необходимого фрагмента можно поместить его в редактор кода, отпустив клавиши Ctrl и Shift. Можно также просмотреть содержимое буфера обмена, щелкнув по закладке Clipboard Ring в окне Toolbox, выбрать нужный элемент и перетащить его в окно редактора кода.

Макросы

Среда разработки Visual Studio .NET позволяет записывать макросы подобно тому, как это делается в приложениях Microsoft Office. Записанный макрос можно сохранить либо на время текущего сеанса работы с Visual Studio, либо в отдельном проекте, который затем можно добавить к любому решению.

Работа с элементами управления

В среде Visual Studio имеется ряд инструментов для манипуляции элементами управления на этапе разработки приложений Windows Forms. Рассмотрим их подробнее.

При создании форм для ввода данных нередко требуется установить определенный порядок обхода элементов управления при нажатии клавиши Tab. Для просмотра и изменения этого порядка следует выбрать пункт меню View | Tab Order и установить порядок обхода элементов управления, выбирая их последовательно с помощью мыши.

Если установить свойство Font формы до того, как на нее будут помещены какие-либо элементы управления, все вновь помещаемые на форму элементы управления унаследуют это свойство формы. При необходимости присвоить одно и то же значение какого-либо свойства нескольким элементам управления можно выделить их (обведя их с помощью мыши или выбрав с помощью щелчков мыши при нажатой клавише Shift или Ctrl) и установить нужные значения общих свойств этих элементов с помощью окна Properties. Для выбора нескольких элементов внутри контейнера следует сначала установить на него фокус ввода.

Редактор форм Visual Studio .NET обладает различными средствами выравнивания элементов на форме (выравнивание по вертикали, горизонтали, установка одинакового размера, установка равных расстояний между элементами по вертикали или горизонтали). Все эти средства можно найти в меню Format, а для их использования следует выбрать нужные компоненты на форме и затем применить к ним одну из команд данного меню. Можно также указать, какие элементы расположены поверх других.

Если выбрать пункт меню Format | Lock Controls, все элементы управления на данной форме будут заблокированы. Блокировка элементов управления применяется тогда, когда пользовательский интерфейс приложения уже спроектирован и нужно избежать случайного смещения элементов управления при щелчках на них с целью добавления связанного с ними кода.

Итак, мы видим, что среда разработки Visual Studio .NET стала гораздо более удобной, нежели среда разработки Visual Studio 6, — в ней появилось много новых инструментов, средств конфигурации и средств, упрощающих проектирование приложений.

Лабораторная работа № 2 (2 часа)

Порядок разработки консольного приложения Visual Studio

Для создания нового консольного приложения запускаем Microsoft Visual Studio 2010 Express и переходим в меню **Файл->Создать->Проект**

В появившемся окне выбираем **Консольное приложение Win32** и задаем имя проекта и нажимаем кнопку **OK**.

В появившемся окне нажимаем кнопку **Далее**.

В следующем окне отмечаем галочку **Дополнительные параметры: Пустой проект** и нажимаем кнопку **Далее**.

В левой части появившегося окна отображается **Обозреватель решений**. Для добавления нового файла программы в проект выбираем по правой кнопке мыши на папке **Файлы исходного кода** меню **Добавить->Создать элемент**.

В появившемся окне выбираем **Файл C++ (.cpp)**, задаем имя файла и нажимаем кнопку **Добавить**.

В появившемся окне набираем текст программы. В качестве примера можно использовать текст программы "Здравствуй, мир!" из раздела Структура программы на языке Си

Для компиляции программы выбираем меню **Отладка->Построить решение**.

В случае успешного построения в нижней части окна отображается **Построение: успешно 1**.

Для запуска приложения выбираем меню **Отладка->Начать отладку**.

Результат выполнения программы:

Для корректного отображения русских символов в левом верхнем углу консоли по нажатию правой кнопки мыши выбираем меню **Свойства**.

В появившемся окне выбрать шрифт, поддерживающий русскую кодировку для кодовой страницы 1251, например Lusida Console.

В результате получаем читаемый русский шрифт.

При желании можно изменить другие параметры окна консоли.

Лабораторная работа № 3 (2 часа)
Консольные приложения для решения логических задач

VisualStudioпозволяет разрабатывать приложения Windows. Но в этой работе мы рассмотрим только создание программ в стилеMS-DOS, называемых *консольными приложениями*. Внешне они выглядят, как программы с текстовым интерфейсом, но способны обращаться к большинству функций Windows. Пример консольного приложения показан ниже, на рис. 1.

Рис. 1. Консольное приложение

Чтобы создать консольное приложение надо дать команду **File ►Newи** в диалоговом окне **NewItems**выбрать значок **Consoleapplication** (рис. 2.).

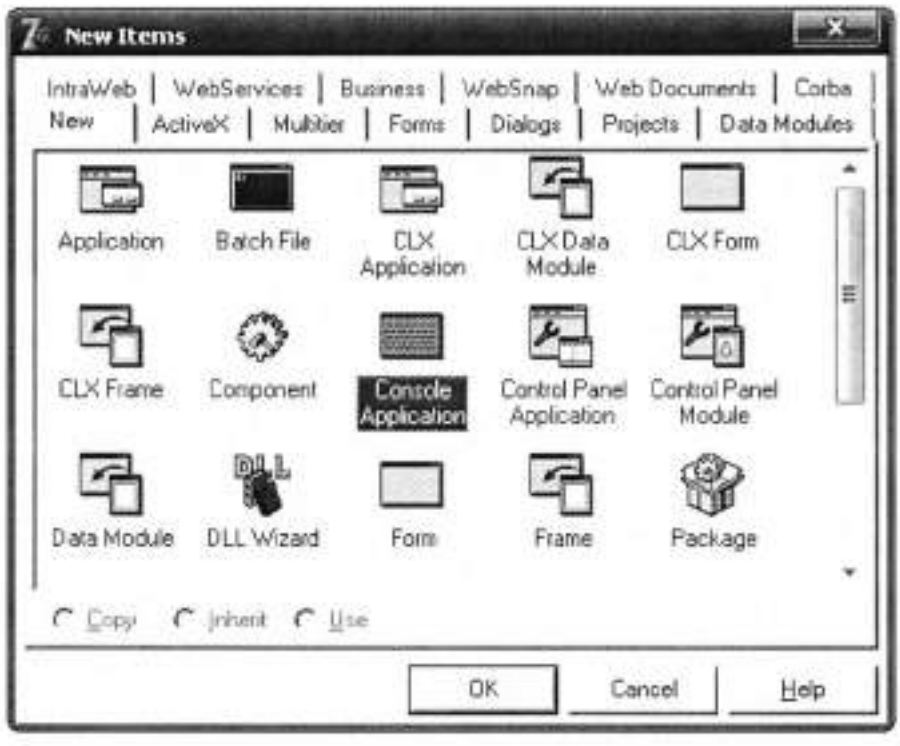


Рис. 2. Выбор категории, к которойотносится создаваемая программа

Задача 1.Некто располагает суммой в A руб. Он хочет купить B книг по C руб. и D журналов по E руб. Написать программу, проверяющую, возможна ли такая покупка.

Решение.

Вначале составим алгоритм решения поставленной задачи.

1. Ввести с клавиатуры исходные данные: *A, B, C, D, E*
2. Проверить корректность введенных данных. Например, очевидно, что количество не может быть меньше единицы, так же как и стоимость не может быть отрицательной.
3. Повторить ввод данных, если они некорректны, или перейти к следующему шагу, если все верно.
4. Подсчитать стоимость всей покупки.
5. Сравнить сумму имеющихся денег (*A*) со стоимостью покупки (*K*) и вывести соответствующее сообщение: если *A*<*K*, то покупка не удастся, в ином случае покупка удастся.

Так как в данном случае количество может быть только целым числом, мы присвоим переменным, обозначающим количество книг и журналов,*B* и *D* целочисленный тип **integer**, а переменным *A, C, E, K* – вещественный тип **real**.

В данной задаче удобно воспользоваться циклом с постусловием **repeat...until**, который повторяется до тех пор, пока не станет истинным условие. Чтобы организовать из него выход, введем еще одну, дополнительную, переменную *f* логического типа **boolean** и присвоим ей значение *false*. Так, если пользователь ввел некорректные данные, то переменной *f* присваивается значение *true*, цикл повторяется, и ввод данных осуществляется снова; если данные корректны, то цикл завершается.

```
repeat//входим в цикл repeat...until
f:=false;//предположим, что ошибок нет
write ('Введите сумму (A): ');
readln(A);
write ('Количество желаемых книг (B): ');
readln(B);
write ('Стоимость одной книги (C): ');
readln(C);
write ('Количество желаемых журналов (D): ');
```

```
readln(D);
write ('Стоимость одного журнала (E): ');
readln(E);
if ((A<1) or (B<1) or (C<0) or (D<1) or (E<0)) then// Проверяемправильностьввода
begin
 f:=true;       //Если что-то введено неправильно, присваиваем переменной f значение true
writeln ('Ошибка!Повторите ввод, пожалуйста');
end;
untilnotf;              //Если ошибок нет, выходим из цикла
```

Посчитаем общую стоимость покупки: количество книг умножим на цену одной книги и сложим с произведением количества журналов на стоимость одного журнала. Получившуюся сумму присвоим переменной *K* и сравним со значением переменной *A*.

```
K := C * B + E * D;          // считаем общую стоимость покупки

If A<Kthen// Проверяем возможность покупки и выводим соответствующее сообщение
```

Для удобства, при выводе результатов типа **real** целесообразно их округлять. Что мы и сделали:

```
writeln('Покупка невозможна, так как ее стоимость (',K:2:2,') превышает ',A:2:2,' р.')

else

writeln ('Покупка возможна, ее стоимость (',K:2:2,') не превышает ',A:2:2,' р.');
```

Двойка после первого двоеточия означает количество позиций, выделяемых для вывода значения численной переменной, включая знак, целую часть, десятичную точку и цифры после запятой. Двойка после второго двоеточия означает количество цифр после десятичной запятой. Лишние позиции будут заменены пробелами перед целой частью числа и нулями после дробной части.

Теперь, когда мы обсудили все основные моменты, приведем полный листинг программы:

```
program Project1;
{$APPTYPE CONSOLE}
uses
```

```
SysUtils;
var B, D:integer;
    A,C,E,K: real;
f:boolean;
begin
repeat//входимвциклгepeat...until
f:=false;
write ('Введитесумму (A): ');
readln(A);
write ('Количество желаемых книг (B): ');
readln(B);
write ('Стоимость одной книги (C): ');
readln(C);
write ('Количество желаемых журналов (D): ');
readln(D);
write ('Стоимость одного журнала (E): ');
readln(E);
if ((A<1) or (B<1) or (C<0) or (D<1) or (E<0)) then//
Проверяемправильностьввода
begin
 f:=true;       //Если что-то введено неправильно, присваиваем переменной f
значение true
writeln ('Ошибка!Повторите ввод, пожалуйста');
end;
untilnotf;              //Если ошибок нет, выходим из цикла
K := C * B + E * D;          // считаем общую стоимость покупки
IfA<Kthen//Проверяем возможность покупки и выводим соответствующее
сообщение
writeln('Покупка невозможна, так как ее стоимость (',K:2:2,') превышает
',A:2:2,' p.')
else
writeln('Покупка возможна, ее стоимость (',K:2:2,') не превышает ',A:2:2,'
p.');
readln;      // Задерживает выполнение программы до нажатия кнопки Enter
end.
```

На рис. 1.1. показана экранная форма с результатами работы программы.

В заключении можно отметить, что в программе остались уязвимые места. Например, если пользователь при вводе десятичного числа вместо точки введет запятую, то возникнет ошибка.

Рис. 1.1. Результат работы программы для задачи 1

Лабораторная работа № 4 (2 часа)

Создание простейших программ с графическим интерфейсом пользователя

В создании пользовательских интерфейсов для приложений Windows Forms имеются три основных этапа:

- Добавление элементов управления на поверхность разработки.
- Установка начальных свойств для элементов управления.
- Написание обработчиков для заданных событий.

Хотя пользовательский интерфейс можно создать, написав собственный код, с помощью конструкторов это можно сделать намного быстрее.

Добавление элементов управления

Элементы управления, такие как кнопки и текстовые поля, можно перетащить мышью на поверхность разработки, представляющую форму. На рисунке ниже показано поле со списком, которое при помощи перетаскивания из панели элементов было добавлено в форму в конструкторе Windows Forms.

При работе в визуальном режиме конструктор преобразует выполняемые действия в исходный код C# и записывает их в файл проекта с именем имя.designer.cs, где имя — имя, назначенное форме. Когда приложение будет выполнено, исходный код разместит элементы пользовательского интерфейса и скорректирует их размер таким образом, как они отображаются на поверхности разработки. Дополнительные сведения см. в разделе Конструктор Windows Forms.

Задание свойств

После добавления элемента управления в форму в окне Свойства можно задать его свойства, такие как цвет фона и текст по умолчанию. Значения, задаваемые в окне Свойства, являются начальными значениями, которые будут назначены соответствующему свойству при создании элемента управления во время выполнения. Во многих случаях доступ к значениям и их изменение возможно программными средствами во время выполнения путем получения или установки свойств в экземпляре класса элемента управления в приложении. Окно Свойство может оказаться полезным во время выполнения, так как с его помощью можно просматривать все свойства, события и методы, поддерживаемые элементом управления. Дополнительные сведения см. в разделе Окно "Свойства".

Обработка событий

Программы с графическим интерфейсом пользователя главным образом основаны на событиях. Такие программы ожидают действий пользователя, например ввода текста в текстовое поле, нажатия кнопки или изменения выбора в поле со списком. При выполнении действия элемент управления, который всего лишь представляет собой экземпляр класса .NET Framework, отправляет событие в приложение. Для обработки события можно написать специальный метод в приложении, который будет вызван при получении события.

В окне Свойства можно указать события, которые должны обрабатываться в коде. Для просмотра событий элемента управления следует выбрать его в конструкторе и нажать кнопку События с изображением молнии в окне Свойства. Следующая схема отображает кнопку событий.

При добавлении обработчика событий с использованием окна Свойства конструктор автоматически напишет основную часть пустого метода, а пользователь должен написать код, на основе которого метод будет выполнять необходимые действия. Большинство элементов управления создают много событий, но приложению часто придется обрабатывать лишь часть из них или даже только одно. Например, возможно потребуется обработать событие Click для кнопки, но не ее событие Paint, если только нет необходимости изменить ее внешний вид каким-либо дополнительным способом. Для каждого элемента управления существует обработчик событий, определенный по умолчанию. Чтобы создать обработчик событий по умолчанию, дважды щелкните элемент управления в форме. Будет создан обработчик событий и откроется редактор кода для написания события для обработки события.

Ключевым средством взаимодействия пользователя с компьютером является графический пользовательский интерфейс (Graphical User Interface, GUI) Из этой главы вы узнаете, как создавать графический пользовательский интерфейс с помощью классов Windows Forms (Формы Windows), которые находятся в NET Framework На практике программирование Windows-приложений предполагает экстенсивное использование различных инструментальных средств и мастеров, которые намного упрощают этот процесс Однако все указанные средства автоматизации заслоняют то, что лежит в основе создания графического пользовательского интерфейса Поэтому сначала мы рассмотрим основы создания графических пользовательских интерфейсов Иными словами, мы научимся создавать простые приложения Windows с самого начала, пользуясь только комплексом инструментальных средств разработки программ NET Framework SDK Это значит, что вначале мы будем создавать простые приложения Windows без применения каких-либо специальных сервисных программ Будут рассмотрены основы рисования с помощью Windows Forms (Формы Windows) с применением шрифтов и кистей, а также необходимые обработчики событий Мы объясним принципы обработки событий в Windows Forms (Формы Windows) и реализуем обработчики событий мыши. С помощью Windows Forms (Формы Windows) мы также реализуем меню и соответствующие обработчики событий. Кроме того, мы рассмотрим управляющие элементы, а после этого изучим среду Visual Studio.NET, посредством которой можно без труда создать простой графический пользовательский интерфейс на C#. С помощью конструктора форм (Forms Designer) добавим в форму управляющие элементы, создадим меню, добавим обработчики событий и другие полезные функциональные возможности. При желании полученный в результате проект на C# можно потом перенести на C++. В заключение будут рассмотрены диалоговые окна и такой элемент управления, как список.

Иерархия Windows Forms (Формы Windows)

Windows Forms (Формы Windows) — это та часть каркаса каркаса .NET Framework, которая поддерживает создание приложений со стандартным графическим пользовательским интерфейсом (GUI) на платформе Windows. Среди классов Windows Forms (Формы Windows) есть обширный набор классов для создания сложных графических пользовательских интерфейсов. Эти классы можно использовать в приложениях, написанных на любом языке .NET

Как правило, ваше приложение будет содержать главное окно, которое реализовано с помощью некоторого класса MyForm, производного от класса Form (Форма). На рис 6.1 изображено, какое место ваш класс MyForm занимает в иерархии классов Windows Forms (Формы Windows).

Создание простых форм с помощью комплекса инструментальных средств разработки программ

Для ознакомления с классами Windows Forms (Формы Windows) полезно будет создать простое приложение SimpleForm (Простая форма) в несколько шагов. Ни на одном из этих шагов мы не будем использовать средства проектирования Visual Studio. Используя интерфейс командной строки, необходимо запустить командный файл build. bat.

Создание простой формы

Приложение SimpleForm (Простая форма) — скелет стандартного — скелет стандартного приложения Windows. Вот код приложения SimpleForm (Простая форма), созданный на шаге 0:

```
//SimpleForm.cpp - Шаг 0
// Эта версия отображает простую форму (simple form)
fusing <mscorlib.dll>
fusing <System.dll>
fusing <System.Drawing.dll> // Система
fusing <System.Windows.Forms.dll> // Система
using namespace System;
// использование пространства имен Система;
using namespace System::Windows::Forms;
// использование пространства имен Система::Windows::Формы;
_gc class Form1 : public Form
// класс сборщика мусора Form1: общедоступная Форма
{
public:
Form1()
{
Size = // Размер
*_nogc new System::Drawing::Size(300,200); // Размер
Text = "Simple Form - Step 0";
// Текст = "Простая Форма - Шаг 0";
}
static void Main()
{
Application::Run(new Form1);
// Приложение:: Выполнить (новая Form1);
}
};
int _stdcall WinMain(
long hInstance, // дескриптор текущего экземпляра
long hPrevInstance, // дескриптор предыдущего экземпляра
long lpCmdLine, // командная строка
int nCmdShow // состояние отображения )
{
Form1::Main();
return 0;
}
```

Класс Form1 является производным от System:: Windows:: Forms:: Form (Система::Windows::Формы::Форма). В классе System::Windows::Forms::Application (Система::Windows::Формы::Приложение) есть статические методы для управления приложением, например Run (Выполнить) и Exit (Выход). Метод WinMain создает новую форму и запускает ее в качестве главного окна приложения.

Обратите внимание, что в примерах, написанных на C++, вместо имени функции main (главная) в качестве точки входа используется WinMain. В принципе можно в функции

main (главная) в рамках консольного приложения реализовать все возможности графического интерфейса пользователя. Но при этом подходе придется создать бездействующее консольное окно, что в приложениях с графическим пользовательским интерфейсом совсем ни к чему. Если же использовать WinMain вместо main (главная), то в программе не создаются консольные окна, а сразу создается главное окно.

Конструктор формы инициализирует форму. Значение в поле Size (Размер) определяет размер формы в пикселях. Поле Text (Текст) определяет заголовок, который отображается в области заголовка окна новой формы.

Ключевым классом Windows Forms (Формы Windows) является базовый класс Form (Форма). Этот класс содержит обширный набор функций, которые наследуются разрабатываемыми нами классами форм, производными от класса Form (Форма).

Чтобы создать приложение, нужно выполнить из командной строки командный файл build.bat. А чтобы запустить командный файл, откройте окно DOS, перейдите в папку SimpleFormXStep(), и введите в командной строке build (компоновка). Помните, что . перед этим необходимо правильно установить значения переменных среды. Для этого достаточно выполнить Visual Studio.NET Command Prompt.

```
cl /CLR SimpleForm.cpp
```

По умолчанию будет откомпилирован исполняемый файл Windows. В исходном коде приложения находятся директивы fusing, в которых указаны используемые библиотеки .NET: System.dll,System.Drawing.dllи System.Windows.Forms.dll.

После того, как вы откомпилировали приложение с помощью командного файла, можете запустить его, введя в командной строке SimpleForm (Простая форма). Вы также можете запустить приложение в проводнике Windows, дважды щелкнув на файле SimpleForm.exe. И хотя приложение SimpleForm (Простая форма) совсем примитивное, в нем заложено множество возможностей, унаследованных созданным нами классом, который является производным от класса Form (Форма). Окно приложения можно перетаскивать по экрану, изменять его размер, свертывать, развертывать, в нем можно открывать системное меню (щелкнув кнопкой мыши в верхнем левом углу окна) и т.д.

Сообщения о работе окна

В Visual Studio.NET есть инструментальное средство под названием Spy++ (Шпион++). Эта программа "шпионит" за окнами, чтобы иметь представление о том, что происходит внутри окна. Чтобы в Visual Studio запустить Spy++ (Шпион++), нужно воспользоваться меню Tools (Сервис). Запустите версию приложения SimpxeForm.exe, полученную на шаге 0, а затем запустите Spy++ (Шпион++). Выберите Spy (Шпион) Find Window (Найти окно) — появится диалоговое окно Find Window (Найти окно). В этом диалоговом окне установите переключатель Messages (Сообщения).

Левой кнопкой мыши перетащите инструмент Finder Tool (Средство поиска) (в диалоговом окне Find window (Найти окно) этот инструмент отображается в виде специальной пиктограммы — перекрестия) на окно приложения SimpleForm (Простая форма), а потом щелкните на кнопке ОК. Теперь в окно программы-шпиона Spy++ будут выводиться сообщения, информирующие обо всех взаимодействиях с окном.

Чтобы обрабатывать события, приложения Windows должны иметь специальную структуру Операционная система Windows в ответ на действие пользователя, например щелчок кнопкой мыши, выбор меню или ввод символов с клавиатуры, посылает приложению сообщение Приложения Windows должны иметь такую структуру, которая позволяет реагировать на эти сообщения

Удобство создания Windows-программ с помощью классов NET Framework состоит в том, что программировать можно на очень высоком уровне абстракции На шаге 0 вы уже убедились, насколько просто создать приложение В последующих разделах мы будем добавлять в приложение новые основные свойства графических пользовательских интерфейсов, и таким образом проиллюстрируем основы создания графических пользовательских интерфейсов с помощью классов Windows Forms (Формы Windows)

Лабораторная работа № 5 (2 часа)
Основные визуальные компоненты и их использование в программах

Свойства компонентов

Все рассматриваемые компоненты расположены на странице **Standard** палитры компонентов **Visual Studio**. Ниже рассматриваются свойства, присущие многим компонентам. Далеко не все свойства относятся к стандартным визуальным компонентам, тем не менее, они также описаны в этой лекции.

Свойство Align

Задает тип выравнивания компонента внутри формы. Во время выполнения программы этим свойством обладают все интерфейсные элементы. Может принимать одно из следующих значений:

Значение	Описание
alNone	Выравнивание не используется. Компонент располагается на том месте, куда был помещен во время создания программы. Значение по умолчанию
alTop	Компонент перемещается в верхнюю часть формы, и его ширина становится равной ширине формы. Высота компонента при этом не изменяется
alBottom	Компонент перемещается в нижнюю часть формы, и его ширина становится равной ширине формы. Высота компонента при этом не изменяется
alLeft	Компонент перемещается в левую часть формы, и его высота становится равной высоте формы. Ширина компонента при этом не изменяется
alRight	Компонент перемещается в правую часть формы, и его высота становится равной высоте формы. Ширина компонента при этом не изменяется
alClient	Компонент полностью занимает всю рабочую область формы

Свойство QI3D

Позволяет задать вид компонента. Если значение этого свойства равно **False**, компонент имеет 2-мерный вид, иначе — 3-мерный (значение по умолчанию).

Свойство Cursor

Позволяет определить вид курсора мыши, который будет отображаться, когда курсор будет находиться в активной области компонента.

Свойство DragCursor

Позволяет определить вид курсора мыши, который будет отображаться, когда в компонент «перетаскивается» другой компонент. Значения этого свойства те же, что и у свойства **Cursor**.

Свойство Color

Задает цвет фона формы или цвет компонента или графического объекта. Может иметь одно из 16 следующих значений: **clBlack** (*черный*), **clMaroon** (*темно-красный*), **clGreen** (*зеленый*), **clOlive** (*оливковый*), **clNavy** (*темно-синий*), **clPurple** (*сиреневый*), **clTeal** (*сине-зеленый*), **clGray** (*серый*), **clSilver** (*серебряный*), **clRed** (*красный*), **clLime** (*ярко-зеленый*), **clBlue** (*голубой*), **clFuchsia** (*сиреневый*), **clAqua** (*ярко-голубой*), **clWhite** (*белый*).

Помимо перечисленных в таблице цветов, значение свойства Color может задаваться шестнадцатеричными значениями. Значения основных цветов представлены в следующей таблице.

Цвет	Значение	Цвет	Значение
Черный	$000000	**Синий**	$000080
Светло-синий	$0000FF	**Светло-зеленый**	$008000
Зеленый	$00FF00	**Сине-зеленый**	$008080
Голубой	$00FFFF	**Коричневый**	$800000
Светло-красный	$800000	**Темно-сиреневый**	$800080
Сиреневый	$FF00FF	**Оливковый**	$808000
Светло-желтый	$FFFF00	**Темно-серый**	$808080
Белый	$FFFFFF	**Светло-серый**	$C0C0C0
Красный	$FF0000		

Свойство DragMode

Позволяет определить режим поддержки протокола «drag & drop». Возможны следующие значения:

dmAutomatic - Компонент можно «перетаскивать», «зацепив» мышью.

dmManual - Компонент не может быть «перетащен» без вызова метода **BeginDrag**

Свойство Enabled

Если это свойство имеет значение **True**, компонент реагирует на сообщения от мыши, клавиатуры и таймера. В противном случае (значение **False**) эти сообщения игнорируются.

Свойство Font

Многие визуальные компоненты используют шрифт по умолчанию. Начальное значение свойства **Height** высчитывается следующим образом:

```
Height := -MulDiv(10, GetDeviceCaps(DC, LOGPIXELSY), 72)
```

Свойство Height

Это свойство задает вертикальный размер компонента или формы. Совместно со свойствами **Width, Left** и **Top** это свойство характеризует размер и положение компонента.

Свойство HelpContext

Задает номер контекста справочной системы. Этот номер должен быть уникальным для каждого компонента. Если компонент активен (находится в фокусе), нажатие клавиши F1 приводит к отображению экрана справочной системы (если такой существует для данного компонента).

Свойство Hint

Задает текст, который будет отображаться при обработке события **OnHint**, происходящего при нахождении курсора в области компонента.

Свойство Left

Задает горизонтальную координату левого угла компонента относительно формы в пикселях. Для форм это значение указывается относительно экрана.

Свойство ParentColor

Это свойство позволяет указать, каким цветом будет отображаться компонент. Если значение этого свойства равно **True**, компонент использует цвет (значение свойства **Color**) родительского компонента. Если же значение свойства **ParentColor** равно **False**, компонент использует значение собственного свойства **Color**.

Свойство ParentCtlBD

Это свойство позволяет указать, каким образом компонент будет определять, является он 3-мерным или нет. Если значение этого свойства равно **True**, то вид компонента задается значением свойства **Ctl3D** его владельца, если же значение этого свойства равно **False**, то значением его собственного свойства **Ctl3D**.

Свойство ParentFont

Это свойство позволяет указать, каким образом компонент будет определять используемый им шрифт. Если значение этого свойства равно **True**, используется шрифт, заданный у владельца компонента, если же это значение равно **False**, то шрифт задается значением его собственного свойства **Font**.

Свойство PopupMenu

Это свойство задает название локального меню, которое будет отображаться при нажатии правой кнопки мыши. (Локальное меню отображается только в случае, когда свойство **AutoPopup** имеет значение **True** или когда вызывается метод **Popup**).

Свойство TabOrder

Задает порядок получения компонентами фокуса при нажатии клавиши **Tab**. Компонент, значение свойства **TabOrder** которого равно 0, получает фокус при отображении формы. Для изменения этого порядка необходимо изменить значение свойства **TabOrder** определенного компонента.

Свойство TabStop

Позволяет указать, может ли компонент получать фокус или нет. Компонент получает фокус, если значение его свойства **TabStop** равно **True**.

Свойство Tag

С помощью этого свойства можно «привязать» к любому компоненту значение типа **LongInt**.

Свойство Top

Задает вертикальную координату левого верхнего угла интерфейсного элемента относительно формы в пикселях. Для формы это значение указывается относительно экрана.

Свойство Visible

Позволяет определить, виден ли компонент на экране. Значением этого свойства управляют методы **Show** и **Hide**.

Свойство Width

Задает горизонтальный размер элемента или формы в пикселях.

Использование визуальных компонентов

Главное меню MainMenu

Для создания меню в Visual Studio предусмотрен дизайнер меню. С активизацией элементов меню связано событие **OnClick**. Обработчик этого события выполняется, когда пользователь выбирает элемент меню либо с помощью мыши, либо с помощью клавиши F10. В ряде случаев возможен доступ к элементам меню по комбинации ALT+клавиша активизации. Такая клавиша указывается с помощью символа **&** в свойстве **Caption** соответствующего элемента меню. Например, чтобы меню **File** вызывалось по нажатию комбинации клавиш **Alt+F**, свойство **Caption** должно иметь следующее значение:

```
Caption := &File;
```

Свойство **Items** предоставляет доступ к элементам меню. Например, если существует меню **File** с элементами **New, Open и Save**, то FileMenu.Items[1] обеспечивает доступ к элементу меню **Open**.

Свойство **Count** содержит число элементов меню. Так, например, чтобы изменить состояние всех элементов меню, необходимо выполнить следующие действия:

```
For I := 0 to MenuItem1.Count-1 do
    MenuItem1.Items[I].Enabled := True;
```

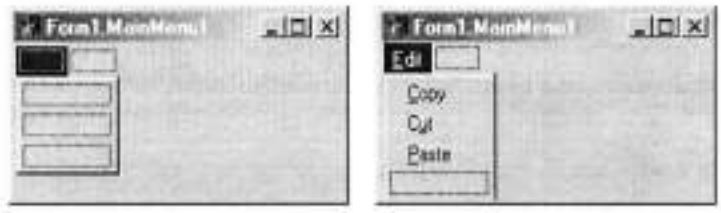
Изменение состояния элементов меню

Рассмотрим пример изменения состояния элемента меню **Edit**, выполняющего копирование **(Copy),** вырезание **(Cut)** и вставку **(Paste)** с областью обмена данными **(Clipboard)**. Элемент Paste должен быть доступным, только когда в области обмена данных находится какая-либо

информация, а элементы Cut и Copy — только в тех случаях, когда имеется какой-то выделенный текст в окне **Memo** (project1.dpr).

Создадим новый проект и разместим на нем элементы Memo1 и MainMenu (Standart).

Вызовем дизайнер меню, щелкнув правой кнопкой на элементе MainMenu, добавим три строки и присвоим пунктам меню соответствующие названия:



Создадим соответствующие обработчики событий:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
 Copy1.Enabled := False;
 Cut1.Enabled := False;
 Paste1.Enabled := False;
 Memo1.Lines.LoadFromFile('c:\autoexec.bat');
end;
procedure TForm1.Edit1Click(Sender: TObject);
begin
if Memo1.SelLength > 0 then // Если текст выделен
 begin Copy1.Enabled := True; Cut1.Enabled := True; end  // Включить Copy1 Cut1
else                    // Если текст не выделен
 begin Copy1.Enabled := False; Cut1.Enabled := False; end; // Выключить Copy1 Cut1
 Paste1.Enabled := ClipBoard.HasFormat(CF_TEXT) //Включить Paste если в ClipBoard
есть текст
end;
procedure TForm1.Copy1Click(Sender: TObject);
begin
 Memo1.CopyToClipboard;
end;
```

```
procedure TForm1.Cut1Click(Sender: TObject);
begin
 Memo1.CutToClipboard;
end;
procedure TForm1.Paste1Click(Sender: TObject);
begin
 Memo1.PasteFromClipboard;
end;
```

Работа программы: в момент создания формы (TForm1.FormCreate) все три команды меню Edit объявляются как недоступные, а в Memo1 считывается текст из файла. Далее, при активизации элемента меню Edit (TForm1.Edit1Click), проверяется соблюдение следующих условий: если в компоненте Memo1 имеется выделенный текст, то становятся доступными команды Cut и Copy, а если в области обмена данными содержится какой-либо текст, то становится доступной команда Paste.

Два главных меню в программе

В программе можно использовать более одного главного меню, например, англоязычное и русскоязычное.

Способ первый:

Создать англоязычное меню EnglishMenu и русскоязычное меню RussianMenu.

Заполнить оба меню необходимыми элементами с помощью редактора меню.

Ассоциировать каждый элемент англоязычного меню с обработчиком события.

Ассоциировать элементы русскоязычного меню с аналогичными обработчиками, уже созданными для англоязычного меню с помощью Object Inspector.

Выбирать способ переключения меню, изменяя свойство **Menu** формы. (В приведенном ниже примере это кнопка, но более удобно использовать локальное меню формы):

```
procedure TForm1.Button1Click(Sender; TObject); begin
// Проверить тип меню
If Menu = EnglishMenu Then Menu := RussianMenu // Переключиться на русскоязычное
Else
```

```
Menu := EnglishMenu; // Переключиться на англоязычное
End;
```

Способ второй:

В приведенном выше примере добавить на форму кнопку выбора языка меню и написать следующий обработчик (project 2.dpr):

```
procedure TForm1.Button1Click(Sender: TObject);
begin
// Проверить тип меню
If Button1.Tag = 0 Then
Begin  // Переключиться на русскоязычное
 Menu.Items[0].Caption:='&Редактор';
 Copy1.Caption:='&Копировать';
 Cut1.Caption:='&Вырезать';
 Paste1.Caption:='&Вставить';
 Button1.Tag := 1
End
Else // Переключиться на англоязычное
Begin
 Menu.Items[0].Caption:='&Edit';
 Copy1.Caption:='&Copy';
 Cut1.Caption:='C&ut';
 Paste1.Caption:='&Paste';
 Button1.Tag := 0
End
End;
```

Локальное меню PopupMenu

Локальное меню становится доступным, когда пользователь нажимает правую кнопку мыши в рабочей области формы или компонента. Для создания локальных меню используется компонент **TPopupMenu**, который

ассоциируется со свойством **PopupMenu** формы или компонента. Локальное меню создается так же, как и главное, с помощью редактора меню **Menu Designer**.

Для того чтобы локальное меню отображалось при нажатии правой кнопки мыши, необходимо присвоить свойству **AutoPopup** значение **True**.

Свойство **Items** задает список элементов локального меню и может использоваться для доступа к элементам меню.

Свойство **PopupComponent** позволяет определить, на каком из имеющихся компонентов появилось локальное меню в том случае, если несколько компонентов имеют одно и то же локальное меню.

Пример: Создать локальное меню для элемента «метка». Команды меню должны определять способ выравнивания текста метки — по центру, по левой или правой границе (project3.dpr).

Поместим в форму компонент типа TLabel (Label1). Установим значение свойств AutoSize в **False** (это даст возможность позиционировать текст) и изменим свойство Color, чтобы видеть границы компонента.

Создадим локальное меню (компонент **TPopUpMenu)** и в нем - три команды: Left, Center и Right.

Для каждой команды меню напишем обработчики событий (изменения значения поля Alignment элемента Label1):

```
procedure TForm1.Center1Click(Sender: TObject);
begin
Label1.Alignment := taCenter
end;
procedure TForm1.Right1Click(Sender: TObject);
begin
Label1.Alignment := taRightJustify
end;
procedure TForm1.Left1Click(Sender: TObject);
begin
Label1.Alignment := taLeftJustify
end;
```

В методе FormCreate установим значение свойства AutoPopup объекта TPopupMenu в **True**:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
```

```
PopupMenu1.AutoPopup := True;

end;
```

Статический текст Label

Элемент этого типа обычно используется, когда необходимо отобразить текст, который не может быть отредактирован пользователем, например заголовков компонентов, не имеющих собственного свойства Caption. Для того чтобы компонент **Label** динамически изменял свой размер в зависимости от размера текста (свойство Caption), необходимо присвоить свойству **AutoSize** значение **True** (значение по умолчанию). Чтобы текст располагался в нескольких строках, присвойте свойству **Wordwrap** значение **True**.

Свойство **Alignment** позволяет задавать тип выравнивания текста — по левой границе (taLeftJustify), по центру (taCenter) или по правой границе (taRightJustiry).

Связать компонент **Label** с другим компонентом можно, присвоив свойству **FocusControl** значение, соответствующее названию компонента. Обычно предоставляется возможность выбора компонента **ListBox** или **Memo** с клавиатуры. Например, если свойство **Caption** компонента **Label**, связанного с компонентом **Memo,** имеет значение &DataList, то клавишей активизации будет клавиша Alt+«D».

Свойство **Transparent** позволяет указать, будет ли прозрачным фон компонента Label. Это удобно при использовании метки совместно с графическими изображениями, когда Label содержит пояснительный текст. В этом случае значение свойства Transparent должно быть **True** (project4.dpr).

Строка редактирования Edit

Строка редактирования - это прямоугольное окно, в котором возможны ввод и редактирование текста: (функции выделения, копирования, удаления). Компонент TEdit не распознает символов конца строки (#13#10).

Для отображения нередактируемого текста необходимо присвоить свойству **Readonly** значение **True**.

Использовать выделенный текст во время работы программы позволят свойства **SelLength, SelStart и SelText**, а также метод **SelectAll**. Для того чтобы ограничить длину строки, необходимо присвоить свойству **MaxLength** требуемое значение. Метод **Clear** используется для удаления всего текста, а метод **ClearSelection** — для удаления выделенного текста. Для работы с областью обмена данными (Clipboard) используются методы **CopyToClipboard, CutToClipboard и PasteFromClipboard**. (Более подробно об этом см. главу «Средства обмена данными».) Свойство **Modified** используется для того, чтобы определить, изменилось ли содержимое компонента TEdit. Свойство **CharCase** используется для изменения регистра символов, содержащихся в тексте.

Компонент можно использовать для ввода пароля. Для этого необходимо присвоить свойству **PasswordChar** значение символа, который будет отображаться при вводе вместо символов. Это может быть, например, символ «*».

Ограничение ввода

В предыдущей лекции было показано, как в строке редактирования разрешить ввод только цифр. Отметим, что обработчик OnKeyPress позволяет реализовать и другие преобразования вводимой информации, например, преобразование вводимых символов в символы верхнего или нижнего регистров (project4.dpr):

```
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);

begin

 If Key in ['a'..'z'] then //Только для букв от «a» до «z»

 Key := Chr(Ord(Key)-32);

end;
```

В Visual Studio реализован компонент TMaskEdit, который задаёт сложные правила ввода в строке редактирования. Шаблон вводимой информации задается с помощью значения свойства **EditMask,** выбор которого приводит к появлению специального редактора шаблонов Input Mask Editor, содержащего примеры ряда шаблонов: для ввода телефонных номеров, даты, времени, почтового индекса и т.д.

Поле, которое не может быть пустым

Информация, вводимая пользователем, может подразделяться на обязательную и дополнительную. Первая должна вводиться всегда. Ниже показано, как создать строку редактирования, которая не может быть пустой.

```
procedure TForm1.Edit1Exit(Sender: TObject);

begin

if Length(Edit1.Text) < 1 then begin

  MessageBox(0, 'Поле не может быть пустым','Ошибка', mb_Ok);

  ActiveControl:=Edit1; end;

end;
```

Событие **OnExit** происходит при потере элементом фокуса. В обработчике этого события проверяем длину текста, введенного в строке редактирования, и, если эта длина равна нулю (т.е. текст не введен), сообщаем о том, что данное поле не может быть пустым и запрещаем перемещение фокуса.

Редактор Memo

Компонент Memo может содержать несколько строк, которые задаются свойством **Text** (для доступа ко всему содержимому компонента) либо свойством **Lines** (для построчного доступа).

Можно ограничить число вводимых символов, присвоив свойству **MaxLength** необходимое значение. Методы **Add, Delete и Insert** используются для добавления, удаления и вставки строк. Например, добавить строку:

```
Memo1.Items.Add('Еще одна строка');
```

Для работы с областью обмена данными используются методы **CopyToClipboard, CutToClipboard и PasteFromClipboard.** Если значение свойства **Readonly** равно **True**, то пользователь не может изменять содержимого редактора.

Свойство **ScrollBars** задает наличие полос прокрутки, свойство **AutoSize** указывает на необходимость изменения размера компонента при изменении размера шрифта, а свойство **Word-Wrap** — на необходимость деления текста на строки.

Если значение свойства **WantTabs** равно **True**, то в тексте можно использовать символы табуляции. Если значение свойства **WantReturns** равно **False**, то для ввода символа Enter необходимо нажать комбинацию Ctrl+Enter.

Для того чтобы выделить весь текст, используется метод SelectAll, выделенный текст доступен через свойство **SelText.** Для выделения части текста необходимо воспользоваться свойствами **SelStart** и **SelLength.**

Заполнение компонента Memo содержимым текстового файла

Компонент Memo хранит информацию в массиве Lines типа TStrings. Для того чтобы заполнить Memo содержимым текстового файла, используется метод LoadFromFile объекта TStrings:

```
If FileExists ('c:\autoexec.bat') then

Memo1.Lines.LoadFromFile('c:\autoexec.bat');
```

При необходимости можно сохранить изменения с помощью метода SaveToFile объекта TStrings. Таким же образом можно заполнить компонент ListBox, описание которого приводится ниже. В этом случае список элементов хранится в массиве Items:

```
If FileExists('c:\autoexec.bat') then ListBox1.Items.LoadFromFile('c:\autoexec.bat');
```

Примечание. Метод **LoadFromFile** может использоваться многими объектами Visual Studio, например **TOLEContainer, TBitmap, TGraphic, TMetaFile, TOutline** и рядом других.

Стандартная кнопка Button

Кнопка может содержать текст, описывающий выполняемое при ее нажатии действие. Нажатие кнопки приводит к незамедлительному выполнению программой каких-либо функций.

Кнопкой по умолчанию считается кнопка, которая посылает событие **OnClick** при нажатии клавиши Enter. Для этого необходимо присвоить свойству кнопки **Default** значение **True**.

Кнопкой «Cancel» считается кнопка, которая посылает событие **OnClick** при нажатии клавиши **Esc**. Для этого необходимо присвоить свойству **Cancel** значение **True**.

Если в какой-то момент работы программы кнопка должна быть недоступной для нажатия, необходимо присвоить свойству **Enabled** значение **False**. Кнопкой могут обрабатывать два события — OnClick и OnDblClick.

Наряду со стандартными в Visual Studio поддерживаются еще два типа кнопок: TBitBtn и TSpeedButton.

Кнопка с независимой фиксацией CheckBox

Кнопка с независимой фиксацией позволяет пользователю выбрать/отменить определенную опцию. При включении или отключении кнопки происходит событие **OnClick**. Кнопка может находиться во включенном или выключенном состоянии (свойство **Checked).** Свойство **State** позволяет установить значение кнопки.

Если значение свойства **AllowGrayed** равно **False**, то кнопка может находиться в двух состояниях — включенном или выключенном. Если же значение этого свойства равно **True**, то кнопка дополнительно может находиться в неактивном состоянии.

Для группировки кнопок с независимой (и зависимой) фиксацией внутри формы их необходимо разместить внутри компонента Panel, GroupBox или ScrollBox.

Кнопка с зависимой фиксацией RadioButton

Кнопки с зависимой фиксацией предназначены для выбора одной опции из нескольких взаимоисключающих. Поэтому они объединяются в группу из нескольких элементов (компонент TGroupBox). Состояние кнопки содержится в свойстве **Checked.** При включении или отключении кнопки происходит событие *OnClick*, обработчик которого может быть единым для всех кнопок, входящих в группу. Например (project 5):

```
procedure TForm1.RadioButton1Click(Sender: TObject);
begin
If Sender = RadioButton1 then
  CheckBox1.Checked := RadioButton1.Checked;
If Sender = RadioButton2 then
  CheckBox2.Checked := RadioButton2.Checked;
If Sender = RadioButton3 then
  CheckBox3.Checked := RadioButton3.Checked;
end;
```

Список ListBox

Компонент **ListBox** содержит список элементов, которые могут быть выбраны при помощи клавиатуры или «мыши», например — список имен файлов.

Список элементов задается свойством **Items**. Методы **Add, Delete и Insert** используются для добавления, удаления и вставки строк. Для того чтобы определить, какой элемент списка выбран, используется значение свойства **ItemIndex,** а для определения того, какой элемент

выбран, используется значение свойства **Selected.** Для сортировки элементов списка установите свойство **Sorted** в **True**. Вертикальный размер элементов задается значением свойства **ItemHeight.** Если вы хотите, чтобы в списке все элементы отображались полностью, присвойте свойству **IntegralHeight** значение **True**. Свойство **MultiSelect** позволяет задать возможность одновременного выбора нескольких элементов. Значение **SelCount** позволяет узнать, сколько элементов выбрано. Число колонок в списке задается значением свойства **Columns.** При выборе элемента в списке происходит событие *OnClick*. В следующем примере показано использование компонента ListBox:

```
// Заполнить содержимое списка

//procedure TForm1.FormCreate(Sender: TObject);

begin

ListBox1.Items.LoadFromFile('C:\AUTOEXEC.BAT' );

end;

//

// Отобразить выбранный элемент списка

//

procedure TForm1.ListBox1Click(Sender: TObject);

begin

With ListBox1 do

Edit1.Text := Items[ItemIndex];

end;
```

Комбинированный список ComboBox

Комбинированный список включает список и строку редактирования. Элементы списка задаются значением свойства **Items**. Тип списка определяется свойством Style:

- **csDropDown** Выпадающий список с окном редактирования, позволяющим пользователю вводить или редактировать текст
- **csSimple** Развернутый список с окном редактирования, позволяющим пользователю вводить или редактировать текст
- **csDropDownList** Выпадающий список без окна редактирования (project6.dpr).

Полоса прокрутки ScrollBar

Полосы прокрутки используются для скроллирования содержимого компонентов, ассоциированных с ними. При использовании полосы прокрутки возникает событие **OnScroll**, в обработчике которого можно предусмотреть действия, выполняемые при скроллировании. Свойство **Position** задает начальное положение бегунка полосы прокрутки. Шаг бегунка задается свойствами **LargeChange** и **SmallChange**.

Минимальное и максимальное значение, на которое может перемещаться бегунок, задается свойствами Min и Max. Во время работы программы значения свойств **Position, Min и Max** задаются с помощью метода **SetParams**.

Полосы прокрутки могут использоваться как средство для наглядного ввода информации, например, для изменения цветов какого-либо компонента, например, панели, расположенной в форме (project7.dpr).

```
// При создании формы
//
procedure TForm1.FormCreate(Sender: TObject);
begin
Panel1.Color := RGB(RedBar.Position, GreenBar.Position, BlueBar.Position);
Label1.Caption := IntToStr(RedBar.Position);
Label2.Caption := IntToStr(GreenBar.Position);
Label3.Caption := IntToStr(BlueBar.Position);
end;
// При изменении положения бегунка в одной из полос
//
procedure TForm1.OnBarChange(Sender: TObject);
begin
Panel1.Color := RGB(RedBar.Position, GreenBar.Position, BlueBar.Position);
Label1.Caption := IntToStr(RedBar.Position);
Label2.Caption := IntToStr(GreenBar.Position);
Label3.Caption := IntToStr(BlueBar.Position);
end;
```

Группа GroupBox

Компонент TGroupBox представляет собой прямоугольник, который обрамляет несколько интерфейсных элементов (обычно кнопок с зависимой или независимой фиксацией). Заголовок группы отображается в левом верхнем углу прямоугольника и задается с помощью свойства **Caption**. После того как компоненты помещены в группу, она становится их родительским классом (свойство **Parent**). Для группировки компонентов можно также использовать компоненты Panel и ScrollBox.

Панель Panel

Компонент типа TPanel служит для создания панелей инструментов и статусных строк, в качестве контейнера. В случае панели инструментов являются кнопки и списки, а в случае статусных строк внутри них чаще всего используются компоненты типа TLabel.

Использование компонента Panel вместо главного окна приложения

Расположите Panel внутри формы и установите требуемый размер панели. Установите значение свойства BorderStyle формы в bsNone. Создайте следующий обработчик события OnCreate формы: (project8.dpr)

```
procedure TForm1.FormCreate(Sender: TObject);
begin
 Form1.Panel1.Left:=0;
 Form1.Panel1.Top:=0;
 Form1.Width:= Form1.Panel1.Width;
 Form1.Height:= Form1.Panel1.Height;
end;
```

Компонент ScrollBox

Этот компонент используется для создания в рамках формы областей, которые могут скроллироваться, то есть содержать больше компонентов, чем отображается в данный момент. Наличие горизонтальной и вертикальной полос прокрутки задается свойствами HorzScrollBar и VertScrollBar, а для того, чтобы определить, виден ли тот или иной компонент, помещенный в компонент TScrollBox, используется метод ScrollInView.

Задание

1. Изучить теоретический материал материал.
2. Создать программу для вычисления арифметического выражения $y = f(a,b,x)$ по индивидуальному варианту.
3. Требования к программе:
   3.1. Программа должна содержать следующие пункты меню выбора типа выполняемых вычислительных процессов: «Линейный», «Разветвляющийся», «Циклический» и пункт «Выход».
   3.2. Пункт «Линейный» должен содержать, в свою очередь, разворачивающийся список режимов: «Ввод», «Расчёт» и «Выход».
   3.3. Ввод исходных происходит в полях редактирования **Edit1-Edit3**, вывод результата осуществляется в поле **Memo1**.
   3.4. Следует предусмотреть контроль ввода в полях **Edit** только числовой информации.

3.5. Выход из пустого поля **Edit** невозможен (с выводом предупреждающего сообщения).

3.6. Активизация пунктов меню должна предусматриваться только в том случае, когда он может быть выполнен, например, режим «Расчёт» должен быть доступен только в случае ввода в поля **Edit** всех исходных данных, а режимы «Разветвляющийся», «Циклический» в данной работе недоступны.

3.7. После ввода исходных данных и выполнения расчёта должен быть активен только пункт «Выход».

4. Отчет должен содержать текст обработчиков событий и описание действий Инспектора событий.

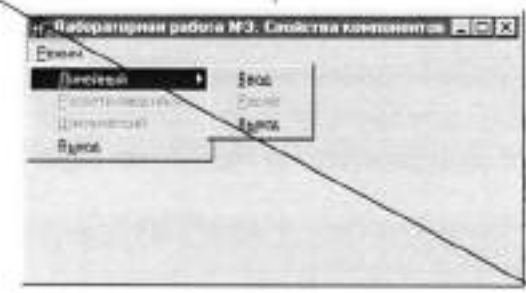
Текст программы:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Menus;

type
  TForm1 = class(TForm)
    GroupBox1: TGroupBox;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    MainMenu1: TMainMenu;
    N1: TMenuItem;
    N2: TMenuItem;
    N3: TMenuItem;
    N4: TMenuItem;
    N5: TMenuItem;
    Memo1: TMemo;
    N6: TMenuItem;
    N7: TMenuItem;
    N8: TMenuItem;
    Edit3: TEdit;
    Edit2: TEdit;
    Edit1: TEdit;
    procedure N5Click(Sender: TObject);
    procedure N2Click(Sender: TObject);
    procedure EditKeyPress(Sender: TObject; var Key: Char);
    procedure EditExit(Sender: TObject);
    procedure N7Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
```

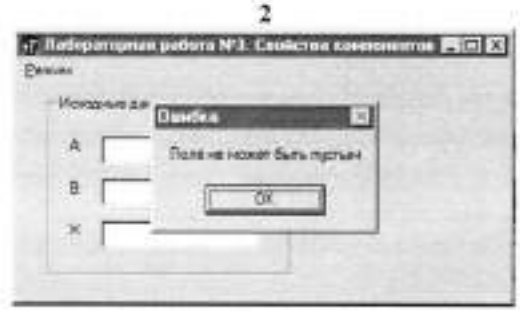
```
implementation

{$R *.dfm}

procedure TForm1.N5Click(Sender: TObject);
begin
Close
end;

procedure TForm1.N2Click(Sender: TObject);
begin
GroupBox1.Visible:=True;

GroupBox1.Enabled:=True;
Edit1.Enabled:=true;
Edit2.Enabled:=true;
Edit3.Enabled:=true;
Label1.Enabled:=true;
Label2.Enabled:=true;
Label3.Enabled:=true;
ActiveControl:=Edit1;
end;

procedure TForm1.EditKeyPress(Sender: TObject; var Key: Char);
begin
If not ((Key in ['0'..'9']) or (Key=',') or (Key='e') or (Key=#8))then Key := #0;
end;

procedure TForm1.EditExit(Sender: TObject);
begin
if ((Length(Edit1.Text) > 0 ) and
   (Length(Edit2.Text) > 0 ) and
   (Length(Edit3.Text) > 0 ))
   then  N7.Enabled:=true else  N7.Enabled:=false;
if (((Sender=Edit1) and (Length(Edit1.Text) < 1 )) or
   ((Sender=Edit2) and (Length(Edit2.Text) < 1 )) or
   ((Sender=Edit3) and (Length(Edit3.Text) < 1 ))) then
   begin
    MessageBox(0, 'Поле не может быть пустым','Ошибка', mb_Ok);
    if (Sender=Edit1) then ActiveControl:=Edit1;
    if (Sender=Edit2) then ActiveControl:=Edit2;
    if (Sender=Edit3) then ActiveControl:=Edit3;
   end;
end;

procedure TForm1.N7Click(Sender: TObject);
var a,b,x,y:real;
begin
a:=StrToFloat(Edit1.Text);
b:=StrToFloat(Edit2.Text);
x:=StrToFloat(Edit3.Text);
y:=a+b-x;
Memo1.Text:=FloatToStr(y);
N2.Enabled:=false;
GroupBox1.Enabled:=false;
end;
```

end.

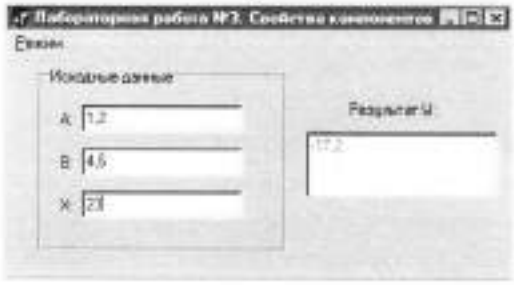
1


Лабораторная работа №3. Свойства компонентов
Режим
Ввод
Вывод

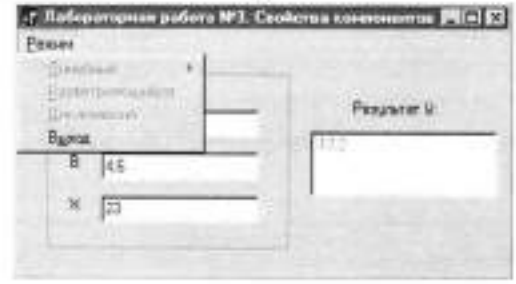
2


Лабораторная работа №3. Свойства компонентов
Режим
Ошибка
Поле не может быть пустым
OK
A
B
X

3


Лабораторная работа №3. Свойства компонентов
Режим
Исходные данные
A 1,2
B 4,5
X 23
Результат U
17,2

4


Лабораторная работа №3. Свойства компонентов
Режим
Вывод
B 4,5
X 23
Результат U

Лабораторная работа № 6 (2 часа)
Создание меню в программах

Чтобы создать стандартное меню

1. В меню Представление, щелкните Представление ресурсов, а затем щелкните правой кнопкой мыши заголовком Меню и выберите команду Добавить ресурс. Выберите Меню.
2. Выберите поле **Новый элемент**, (прямоугольник с надписью "Введите здесь текст") в строке меню.

Поле "Новый элемент"

Введенный текст появляется и в редакторе **Меню**, и в поле **Заголовок** в окне "Свойства". Изменить свойства нового пункта меню можно в любом из этих мест.

Как только вы присвоите название новому меню в строке меню, справа появится поле нового элемента (чтобы вы могли добавить новое меню), а другое меню нового элемента откроется под первым меню, чтобы вы могли добавлять.

Поле "Новый элемент" со смещаемым фокусом после введения названия меню

Примечание

Чтобы создать меню с одним элементом в строке меню, установите значение свойства "Контекстное меню" равным "*False*".

Задание:

Используя программу предыдущей работы создать меню.

Лабораторная работа № 7 (2 часа)
Использование диалоговых окон

Для работы с файлами в Visual Studio можно использовать диалоговые окна. Рассмотрим свойства некоторых компонентов, с помощью которых в Visual Studio реализуются диалоговые окна.

Диалоговое окно выбора имени открываемого файла OpenDialog предназначено для просмотра файловой системы компьютера и выбора имени требуемого файла. Компонент OpenDialog не предназначен для автоматического открытия файлов. Он позволяет лишь получить имя выбранного пользователем файла. Непосредственное открытие файла осуществляется при помощи стандартных процедур языка Object Pascal либо специальных методов, определённых, например, в классе TStrings.

Рассмотрим основные свойства класса TOpenDialog, экземпляром которого является компонент OpenDialog.

property DefaultExt : string;
Содержит расширение, добавляемое к имени файла, если у него не указано расширение.

property FileName : string;
Содержит имя выбранного файла.

property Files : TStrings;
Содержит список имён выделенных файлов.

property Filter : string;
Содержит описание одного или нескольких файловых фильтров. Например, фильтр * . pas поможет пользователю отображать в диалоговом окне только файлы, имеющие расширение . pas.

property InitialDir : string;
Определяет папку, содержимое которой появляется при открытии диалогового окна.

function Execute : boolean;
Размещает диалоговое окно на экране в модальном режиме. Модальный режим означает, что выполнение приложения приостанавливается до тех пор, пока пользователь не закроет модальное окно. Функция возвращает значение true, если окно закрыто кнопкой **Открыть**, и false, если закрыто кнопкой **Отмена**.

Диалоговое окно SaveDialog очень похоже на окно OpenDialog, но в отличие от него используется при сохранении файла.

Диалоговое окно FontDialog позволяет пользователю выбирать шрифт и устанавливать его характеристики. Основным свойством компонента FontDialog является свойство Font, задающее характеристики шрифта.

Пример использования компонентов OpenDialog, SaveDialog и FontDialog.

Создадим простой текстовый редактор, который позволил бы с помощью диалоговых окон открывать и сохранять текстовые файлы, а также изменять характеристики шрифта.

Разместим на форме следующие компоненты OpenDialog1, SaveDialog1, FontDialog1 выберем из страницы Dialogs, Memo1, Button1, Button2, Button3 – из страницы Standard.

Выберем свойство Filter компонента OpenDialog1 и щёлкнем по появившейся кнопке с тремя точками. Появится диалоговое окно Filter Editor, с помощью которого можно задать фильтры.

Filter Name	**Filter**
Текстовые файлы (*.txt, *.doc)	*.txt; *.doc
Все файлы (*.*)	*.*

После заполнения нажать кнопку Ok.

Для компонента SaveDialog значение свойства DefaultExt установим равным txt. т.е., если при сохранении файла расширение не будет указано, то по умолчанию добавится расширение txt.

Кнопкам Button1, Button2, Button3 установим свойство Caption равным 'Открыть', 'Сохранить', 'Шрифт'.

```
unit Unit1;
interface
uses
 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
 Dialogs, StdCtrls;
type
 TForm1 = class(TForm)
 OpenDialog1: TOpenDialog;
 SaveDialog1: TSaveDialog;
 FontDialog1: TFontDialog;
 Memo1: TMemo;
 Button1: TButton;
 Button2: TButton;
 Button3: TButton;
 procedure Button1Click(Sender: TObject);
 procedure Button2Click(Sender: TObject);
 procedure Button3Click(Sender: TObject);
 private
 { Private declarations }
 public
 { Public declarations }
 end;
var
```

```
 Form1: TForm1;
implementation
{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
begin
if not opendialog1.execute then exit;
memo1.Lines.LoadFromFile(OpenDialog1.filename)
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
if not savedialog1.execute then exit;
memo1.Lines.SaveToFile(savedialog1.filename)
end;
procedure TForm1.Button3Click(Sender: TObject);
begin
if not fontdialog1.execute then exit;
memo1.Font:=fontdialog1.font
end;
end.
```

Лабораторная работа № 8 (2 часа)
Отладка программ

1 Цель и порядок работы

Цель работы – изучить инструментальные средства и возможности отладки программ в интегрированной среде Microsoft Visual C++

Порядок выполнения работы:

- ознакомиться с описанием лабораторной работы;
- получить задание у преподавателя, согласно своему варианту;
- написать программу и отладить ее на ЭВМ;
- оформить отчет.

2 Краткая теория

Интегрированная интерактивная среда разработки программ Microsoft Visual C++ (IDE) включает в себя ряд средств, облегчающих процесс нахождения ошибок в программе, которые не позволяют ей корректно работать.

2.1 Понятие отладки

Отладка – это процесс поиска и исправления ошибок в программе, препятствующих корректной работе программы.

Отладка программы является одним из наиболее важных и трудоемких этапов разработки. Трудоемкость и эффективность отладки напрямую зависит от способа отладки и от средств языка программирования.

Существует ряд простых вещей, которые могут облегчить отладку Вашей программы. Для большинства случаев справедливо то, что не следует располагать на одной строке программы более одного оператора. Так как выполнение программы в процессе отладки происходит строка за строкой, это требование будет обеспечивать выполнение не более одного оператора каждый раз.

В то же время допускаются и случаи, когда можно разместить на строке несколько операторов. Если есть список операторов, которые нужно выполнить, но которые фактически не имеют отношения к отладке, то Вы можете произвольно располагать их на одной или двух строках так, чтобы их можно было быстрее пройти при пошаговом выполнении.

В конце концов, лучшей отладкой является профилактическая отладка. Хорошо разработанная, ясно написанная программа может иметь не только немного ошибок, но и будет облегчать для Вас трассировку и фиксацию местоположения этих ошибок. Существует несколько основных положений, о которых следует помнить при составлении программы:

- программируйте с постепенным наращиванием. При возможности кодируйте и отлаживайте программу небольшими секциями. Прорабатывайте каждую секцию до конца прежде чем переходить к следующей;
- разбивайте программу на части: модули, процедуры, функции. Избегайте построения функций, размер которых больше 25-30 строк, в противном случае разбивайте их на несколько меньших по размеру функций;
- старайтесь передавать информацию только через параметры, вместо использования глобальных переменных внутри процедур и функций. Это поможет Вам избежать побочных явлений и облегчит отладку программы, так как Вы сможете легко прослеживать всю информацию, входящую и выходящую из заданной процедуры или функции;

– не торопитесь. Сосредоточьте действия на том, чтобы программа работала правильно, прежде чем предпринимать шаги по ускорению ее работы.

2.2 Разновидности ошибок

Существует три основных типа ошибок: ошибки этапа компиляции, ошибки этапа выполнения и логические ошибки.

2.2.1 Ошибки этапа компиляции

Ошибки этапа компиляции или синтаксические ошибки происходят, когда исходный код нарушает правила синтаксиса языка C++. Visual C++ не может скомпилировать программу, пока она не будет содержать допустимые операторы и меть правильную структуру. Когда компилятор встречает оператор, который он не может распознать, то в окно Output среды разработки, заносится номер строки, в которой была найдена ошибка и описание ошибки. Дважды кликнув на запись об ошибке вы попадаете на строку в которой она произошла.

Наиболее общей причиной ошибок этапа компиляции являются ошибки набора (опечатки), пропущенные точки с запятой, ссылки на неописанные переменные, передача неверного числа (или типа) параметров функции и присваивание переменной значений неверного типа.

После устранения в программе всех синтаксических ошибок и ее успешной компиляции программа будет готова к выполнению и поиску ошибок этапа выполнения и логических ошибок.

2.2.2 Ошибки этапа выполнения

Ошибки этапа выполнения или семантические ошибки происходят, когда после компиляции полной программы, при ее выполнении делается что-то недопустимое. То есть, программа содержит допустимые операторы, но при выполнении операторов что-то происходит неверно. Например, программа может пытаться выполнить деление на ноль или открыть для ввода несуществующий файл.

2.2.3 Логические ошибки

Логические ошибки – это ошибки проектирования и реализации программы. То есть, все операторы допустимы и что-то делают, но не то, что предполагалось. Эти ошибки часто трудно отследить, поскольку IDE не может найти их автоматически, как синтаксические и семантические ошибки. К счастью, IDE включает в себя средства отладки, помогающие найти логические ошибки.

Логические ошибки приводят к некорректному или непредвиденному значению переменных, неправильному виду графических изображений или невыполнению кода, когда это ожидается. Далее рассматриваются методы отслеживания этих логических ошибок.

2.3 Методы отладки

Иногда, когда программа делает что-то непредвиденное, причина достаточно очевидна, и можно быстро исправить код программы. Но другие ошибки более трудноуловимы и вызываются взаимодействие различных частей программы. В этих случаях лучше всего остановить программу в заданной точке, пройти ее шаг за шагом и просмотреть состояние переменных и выражений. Такое управляемое выполнение – ключевой элемент отладки.

2.3.1 Установка точки прерывания

Точка прерывания позволяет остановить выполнение программы перед любой выполняемой инструкцией (оператором) с тем, чтобы продолжать выполнение программы либо в пошаговом режиме, либо в непрерывном режиме до следующей точки прерывания.

Чтобы задать точку прерывания перед некоторым оператором, необходимо установить перед ним текстовый курсор и нажать клавишу F9 или кликнуть мышью на левом боковом поле окна редактирования напротив оператора. Точка прерывания обозначается в виде красного кружка на левом поле окна редактирования. Повторное действие (щелчок на указанной кнопке или нажатие F9) снимает точку прерывания. В программе может быть несколько точек прерывания.

Для просмотра всех точек остановки необходимо выбрать пункт Debug | Windows | Breakpoints или нажать комбинацию клавиш Ctrl+Alt+B.

2.3.2 Выполнение программы до точки прерывания

Программа запускается в отладочном режиме с помощью команды Debug | Start Debugging (или нажатием клавиши F5).

В результате код программы выполняется до строки, на которой установлена точка прерывания. Затем программа останавливается и отображает в окне Editor ту часть кода, где находится точка прерывания, причем желтая стрелка на левом поле указывает на строку, которая будет выполняться на следующем шаге отладки.

2.3.3 Прекращение отладки

В ходе сеанса отладки иногда желательно начать все сначала. Выбор команды Debug | Stop Debugging или нажатие клавиши Shift+F5 приведет к полному сбросу, так что выполнение по шагам, или трассировка прекратится.

2.3.4 Пошаговое выполнение программы и трассировка

Команды выполнения по шагам Step Over (клавиша F10) и трассировки Trace Into (клавиша F11) меню Debug дают возможность построчного выполнения программы. Единственное отличие выполнения по шагам и трассировки состоит в том, как они работают с вызовами функций.

Выполнение по шагам вызова функции интерпретирует вызов как простой оператор и после завершения подпрограммы возвращает управление на следующую строку. Трассировка подпрограммы загружает код этой подпрограммы и продолжает ее построчное выполнение.

Нажимая клавишу F10, можно выполнять один оператор программы за другим. Предположим, что при пошаговом выполнении программы вы дошли до строки, в которой вызывается некоторая функция func1(). Если вы хотите пройти через код вызываемой функции, то надо нажать клавишу F11, а если внутренняя работа функции вас не интересует, а интересен только результат ее выполнения, то надо нажать клавишу F10.

Допустим, что вы вошли в код функции func1(), нажав клавишу F11, но через несколько строк решили выйти из него, т.е. продолжить отладку после возврата из функции. В этом случае надо нажать клавиши Shift+F11 или выбрать пункт Debug | Step Out.

2.3.5 Выполнение программы до курсора

Иногда, конечно, нежелательно выполнять по шагам всю программу только для того, чтобы добраться до того места, где возникает проблема. Отладчик позволяет выполнить сразу большой фрагмент программы до той точки, где необходимо начать выполнение по шагам.

Существует возможность пропустить пошаговое выполнение некоторого куска программы: установите текстовый курсор в нужное место программы и нажмите клавиши

Ctrl+F10. Программа будет выполнена до курсора и остановится в данной точке ожидая дальнейших действий пользователя.

Причем, это можно сделать как в начале сеанса отладки, так и когда уже часть программы выполнена по шагам или протрассирована.

Чтобы продолжить дальнейшее выполнение программы без пошагового режима нажмите F5.

2.3.6 Отслеживание значений переменных во время выполнения программы

Выполнение программы по шагам или ее трассировка могут помочь найти ошибки в алгоритме программы, но обычно желательно также знать, что происходит на каждом шаге со значениями отдельных переменных.

Для наблюдения за выводом программы встроенный отладчик имеет средства для просмотра значений переменных, выражений и структур данных.

Чтобы узнать значение переменной задержите над ней указатель мыши. Рядом с именем переменной на экране появляется подсказка со значением этой переменной.

Часто программисту необходимо отслеживать значение конкретной переменной или выражения при выполнении программы по шагам.

Помимо экранной подсказки, переменные со своим значением отображается в окне Locals, расположенном в левом нижнем углу экрана. В этом окне приведены значения последних переменных, с которыми работал Visual C++. Кроме этого, в окне «Watch 1» (которое находится также в левом нижнем углу, но на другой вкладке) можно задать имя любой переменной, за значениями которой вы хотите понаблюдать.

Также можно добавить переменную в список просмотра «Watch 1» нажав правую кнопку мыши над именем переменой и выбрав пункт «Add Watch». Кроме этого в окне просмотра «Watch 1» можно добавлять или удалять отслеживаемые элементы. В этом диалоговом окне можно проверять переменные и выражения и изменять значения любых переменных, включая строки, указатели, элементы массива и поля записей, что позволяет проверить реакцию программы на различные условия.

Оба средства вычисление и просмотра работают на уровне выражений, поэтому важно определить, что считается выражением. Выражение состоит из констант, переменных и структур данных, скомбинированных с помощью операций и большинства встроенных функций. Почти все, что можно использовать в правой части оператора присваивания, может также использоваться в качестве отладочного выражения.

Во время отладки можно изменить значение переменной. Для этого нужно ввести новое значение переменной в столбце Value, после чего оно отобразится красным цветом.

2.3.7 Создание условной точки останова

Среда разработки поддерживает создание дополнительных условий для точек остановки. Задав точку остановки, вы можете указать дополнительное условие ее срабатывания. Нажав правой кнопкой мыши над строкой с точкой остановки в меню Breakpoint появятся новые подпункты. В этих точках останова можно задать несколько дополнительных параметров:

Location – место остановки (строка и модуль для остановки);

Condition – проверка условие изменения (на равенство какому-либо значению или на изменение значения);

Hit Count – проверка на счетчик проходов;

Filter – комбинация нескольких условий для значений переменной;

When Hit – задает действия при возникновении остановки.

2.4 Работа с отладчиком

При отладке программы, наименьшим выполняемым элементом является строка. Поэтому, если на одной строке программы содержится несколько операторов, нельзя отладить эти операторы индивидуально. Следовательно, нужно разбить оператор на несколько строк, которые будут выполняться пошагово.

2.4.1 Выполнение программы по шагам без захода в функцию

Это простейший способ выполнения программы по элементарным фрагментам. Выбор команды Debug | Step Over или нажатие клавиши F10 вызывает выполнение отладчиком всего кода в операторе, указанном строкой выполнения, включая любые вызываемые на ней процедуры или функции, пока управление не вернется обратно к программисту. После этого строка выполнения указывает **следующий** выполняемый оператор.

Возьмем следующий пример программы.

Пример 1 – Простая программа, выполняемая по шагам.

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int sqr(int x)
{                                   //10
  int q = x*x;                      //11
   return q;                        //12
}                                   //13
void main()
{                                   //0
  const int N = 10;                 //1
  int a[N] = {5, 2, 7, -9, 4, 8, -1, 0, 3, 6};//2
  //найдем сумму квадратов          //3
  //положительных элементов массива //4
  int s = 0;                        //5
  for (int i = 0; i < N; i++)       //6
    if (a[i] > 0) s += sqr(a[i]);   //7
  cout << s << endl;                //8
  return;                           //9
}
```

Если нажать клавишу F10, то строка выполнения перемещается на фигурную скобку в начале программы (строка 0), поскольку это первое, что выполняется в программе. Второе нажатие клавиши F10 перемещает строку выполнения вниз до оператора объявления константы размерности массива N на следующей строке (строка 1). После этого нажатие F10 переводит указатель к строке с объявлением массива и его инициализации (строка 2). Далее, при нажатии F10, строки 3 и 4 будут пропущены, так как они состоят только из комментариев и указатель переместится к строке 5. В ней производится объявление переменной для хранения суммы квадратов элементов массива s и ее инициализация нулем.

После этого нажатие F10 вызывает начало выполнения цикла for. Первое нажатие инициализирует переменную i, и указатель переходит к строке 7. Далее нажатие F10 приводит к выполнению оператора if и указатель переходит на строку 8. Однако далее управление будет передано обратно на строку 6, так как цикл еще не завершен.

Обратите внимание, что в окне Locals выводятся значения счетчика i, максимального элемента m, размерности массива N и сам массив a.

Сравнение (строка 7) вызывается 10 раз, но не возможно понять выполняется ли при этом операция присвоение нового значения m. Выполнение по шагам не позволяет отладчику показывать детали любых вычислений для отдельной строки.

Выполнение по шагам вызывает выполнение всего оператора сразу, поэтому невозможно видеть изменения в ходе выполнения цикла.

Если необходимо видеть подробности, то в пример нужно внести следующее простое изменение:

Пример 2 – Модифицированная программа, для лучшего выполнения по шагам

```
#include "stdafx.h"

#include <iostream>

using namespace std;

int sqr(int x)

{                                   //10

  int q = x*x;                      //11

   return q;                        //12

}                                   //13

void main()

{                                   //0

  const int N = 10;                 //1

  int a[N] = {5, 2, 7, -9, 4, 8, -1, 0, 3, 6};//2

  //найдем сумму квадратов          //3

  //положительных элементов массива //4

  int s = 0;                        //5

  for (int i = 0; i < N; i++)       //6

     if (a[i] > 0)                  //7.0

        s += sqr(a[i]);             //7.1

  cout << s << endl;                //8

  return;                           //9

}
```

Если теперь нажимать клавишу F10, то указатель после строка 7.0 будет при выполнении условия переходить на строку 7.1, а при не выполнении на строку 8.

2.4.2 Выполнение программы по шагам с заходом в функцию (трассировка)

Чтобы выполнить трассировку кода, необходимо выбрать команду Debug | Trace Into или нажать клавишу F11.

Первое нажатие на F11 также передает управление на фигурную скобку основной программы (строка 0). Повторные нажатия F11 снова перемещают строку управления на теже операторы, что и F10, пока мы не доходим до строки 7.1. После этого нажатие клавиши F11 трассирует вызов функции sqr – строка выполнения перемещается на фигурную скобку в блоке функции (строка 10). Если продолжать нажимать F11, строка выполнения перемещается по функции, а затем, когда дойдет до оператора return (при этом указатель будет располагаться напротив закрывающей фигурной скобки), возвращается к оператору вызова.

Следует обратить внимание на то, что трассировке подвержены и функции стандартных библиотек. Так, например, при выводе на экран s при помощи стандартного потока cout будет произведена трассировка функции вывод. Во избежание данного поведения необходимо использовать пошаговое выполнение (Step Over) по клавише F10, либо нажимать Shift + F11 для выхода из трассируемой функции.

3 Контрольные вопросы

1. Что такое отладка?
2. Для чего предназначена отладка?
3. Какие разновидности ошибок существуют?
4. Какие средства отладки предоставляет среда разработки MS Visual C++?
5. Что такое точка остановки (breakpoint)?
6. Какие дополнительные условия можно устанавливать в точке остановки?
7. Какие возможности существуют для слежения за значениями переменных во время отладки?
8. Как изменить значение переменной?
9. Какие горячие клавиши для работы с отдатчиком вы знаете?

4 Задание

1. Выполнить задание в соответствии с пунктом 5.1.
2. Выполнить задание в соответствии с пунктом 5.2.
3. Оформить отчёт.

5 Задание для выполнения работы

В отчете должен присутствовать текст программы с номерами строк. Последовательность выполняемых строк записывается в столбец, с указанием значений изменяющихся переменных через запятую. При выполнении программы до точки остановки, в последовательности выполнения указать строку с точкой остановки и значения переменных в данной точке.

Например:

```
...
const int N = 10;                    //1

int a[N] = {5, 2, 7, -9, 4, 8, -1, 0, 3, 6};//2

...
```

Последовательность выполнения:

```
...
2: a = {5, 2, 7, 9, 4, 8, -1, 0, 3, 6}
5: i = 0
6: i = 0, s = 25
...
```

5.1 Часть первая

1. Набрать программу примера 1 (раздел 2.4.1)
2. Выполнить программу по шагам, фиксируя в отчете, в строке с каким номером находится строка выполнения при каждом нажатии на F10.
3. Внести в программу изменения в соответствии с примером 2.
4. Выполнить задание пункта 2.
5. Выполнить трассировку исходной программы примера 3 (раздел 2.4.2), фиксируя в отчете, в строке с каким номером находится строка выполнения при каждом нажатии на F11.

5.2 Часть вторая

1. Набрать и откомпилировать следующую программу:

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int sum(int *x, const int N)
{
  int s = 0;
  for (int i = 0; i < N ; i++)
     s += x[i];
   return s;
}
void main()
{
  const int N = 10;
  int a[N] = {1, 3, -5, 0, 4, 6, -1, 9, 3, 2};
  //найдем максимальный элемент массива
  int m = a[0];
  for (int i = 1; i < N; i++)
     if (m < a[i])
        m = a[i];
  cout << m << endl;
  //найдем сумму элементов массива
  int s;
  s = sum(a, N);
  cout << s << endl;
  int z = s / m;
  int k = 0;
  for (int i = 0; i < N; i++)
     if (a[i] > z)
        k += a[i];
     else
        k -= a[i];
  cout << k << endl;
  int x, y;
  cin >> x >> y;
  s = 0;
  while ((x != 0) && (x != 0))
  {
```

```
    x--;
    y--;
    s += x + y;
  }
  cout << s << endl;
  return;
}
```

2. После каждой строки программы проставить номер. Например, //1, //2 и т.д.
3. Выполнить трассировку программы (без захода в функции стандартных библиотек), наблюдая за переменными в окне Locals.
4. Остановить отладку программы.
5. Установить точку остановки на операторе if (a[i] > z).
6. Выполнить программу до курсора на строке s = sum(a, N);
7. Продолжить выполнить программы до точки остановки. Далее продолжать пошаговое выполнение до строки cout << k << endl;.
8. Добавить в окно Watch 1 переменные x, y, s для наблюдения изменения их значений. Продолжать пошаговое выполнение.
9. Остановить отладку программы.
10. В операторе цикла while задать условную точку останова по числу проходов. Запустить программу для отладки.
11. Продолжать пошаговое выполнение до конца программы, наблюдая изменение значений x, y, s в окне Watch 1.
12. Записать полученные результаты.
13. Остановить отладку программы.
14. В операторе цикла while задать еще одну условную точку останова по логическому условию. Запустить программу для отладки
15. Продолжать пошаговое выполнение до конца программы, наблюдая изменение значений x, y, s в окне Watch 1.
16. Записать полученные результаты.

6 Содержание отчета

1. Титульный лист.
2. Наименование и цель работы.
3. Краткое теоретическое описание.
4. Задание на лабораторную работу.
5. Схема алгоритма.
6. Листинг программы.
7. Результаты выполнения программы.

Лабораторная работа № 8 (2 часа)
Создание программ с многооконным (MDI) интерфейсом

1. Многооконный интерфейс

Приложение с однооконным интерфейсом не всегда может полностью удовлетворять потребностям пользователя. Если нужно одновременно работать с несколькими документами, вы, конечно, можете одновременно запустить несколько копий одного приложения, но гораздо удобнее использовать приложения с многооконным интерфейсом.

В таких приложениях вы одновременно можете открыть несколько документов. Каждому документу будет отведено собственное окно просмотра, но, тем не менее, все окна просмотра документов будут расположены внутри главного окна приложения, будут иметь общее меню, а также панели управления и состояния.

Рассмотрим приложение Multi с многооконным интерфейсом, созданное с использованием средств MFC AppWizard. Сначала мы создадим простое приложение, а затем покажем, как его можно изменить, добавив возможности создания новых документов, рисования в окне и сохранения изменений в файле на диске.

Далее мы усовершенствуем наше приложение так, что оно сможет работать с документами двух типов – графическими и текстовыми.

В ходе своих объяснений мы иногда будем ссылаться на однооконное приложение Single, также созданное с использованием средств MFC AppWizard. Приложение Single мы рассматривали в 24 томе "Библиотеки системного программиста". Если у вас нет под рукой 24 тома, вы можете быстро создать приложение Single, воспользовавшись MFC AppWizard.

Приложение Multi

Создайте новое приложение с многооконным интерфейсом и назовите его Multi. При определении свойств приложения оставьте все предложения по умолчанию. Приложение Multi не будет использовать технологию OLE и сетевые технологии, не будет работать с базами данных. Процедура создания приложений с использованием MFC AppWizard описана в разделе "Приложение с оконным интерфейсом" 24 тома серии "Библиотека системного программиста", поэтому мы будем считать, что вы уже создали проект.

Постройте проект и запустите полученное приложение. На экране появится главное окно. Внутри главного окна расположены меню, панель управления и панель состояния.

Сразу после запуска приложения Multi, открывается дочернее окно, предназначенное для просмотра документа, которое получает название Multi1. Вы можете создать новые дочерние окна, выбрав из меню File строку New – открыть новый документ или строку Open – открыть файл (рис. 1.1). Для просмотра уже открытого документа можно открыть еще одно окно (рис. 1.11). В названии такого окна указывается дополнительный числовой индекс.

Если одновременно открыто несколько окон, то можно упорядочить расположение этих окон и пиктограмм, представляющих минимизированные окна. Для этого специально предназначено меню Window.

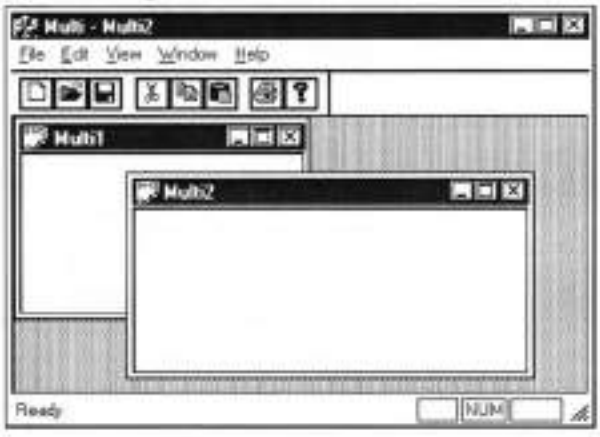


Рис. 1.1. Приложение Multi

Теперь рассмотрим внимательно сам проект Multi, подготовленный для нас MFC AppWizard. Найдите окно Project Workspace и откройте страницу FileView. Вы увидите список всех исходных файлов, входящих в проект (рис. 1.2). В отдельную папку Dependencies будут помещены названия вспомогательных файлов проекта. Эти файлы не входят в проект непосредственно, но используются либо для хранения ресурсов, либо как включаемые файлы, указанные директивой #include в одном или нескольких основных файлах проекта.

Рис. 1.2. Окно Project Workspace, файлы проекта

В следующей таблице кратко описаны основные файлы проекта Multi. Ниже мы подробно рассмотрим ресурсы приложения Multi, а также опишем составляющие его классы и их методы.

Имя файла	Описание
ChildFrm.cpp	Файл содержит определение методов класса CChildFrame
ChildFrm.h	В файле находится определение класса дочернего окна MDI – CChildFrame
MainFrm.cpp	Файл содержит определения методов класса CMainFrame
MainFrm.h	Содержит описание класса главного окна приложения, который называется CMainFrame. Класс CMainFrame наследуется от базового класса CFrameWnd, определенного в библиотеке классов MFC
Multi.cpp	Основной файл приложения. В нем определены методы основного класса приложения CMultiApp
Multi.h	В этом файле перечислены другие включаемые файлы и описан главный класс приложения CMultiApp
Multi.pch	Файл создается во время первой трансляции программы, если вы используете предварительную компиляцию включаемых файлов
Multi.rc	Файл ресурсов. В этом файле описаны все ресурсы приложения. Сами ресурсы могут быть записаны в каталоге RES, расположенном в главном каталоге проекта
MultiDoc.cpp	Включает определение методов класса CMultiDoc
MultiDoc.h	Содержит определение класса документов приложения – CMultiDoc
MultiView.cpp	Включает определение методов класса CMultiView
MultiView.h	Содержит описание класса окна просмотра приложения – CMultiView
ReadMe.txt	Текстовый файл, содержащий описание проекта. В нем кратко рассмотрен каждый файл, входящий в проект, перечислены классы приложения, а также представлена некоторая другая дополнительная информация
res\Multi.ico	Пиктограмма приложения
res\Multi.rc2	В этом файле определены ресурсы, которые нельзя редактировать с помощью редактора ресурсов среды Visual C++
res\MultiDoc.ico	Пиктограмма для документов приложения
res\Toolbar.bmp	Файл содержит растровое изображение кнопок панели управления
Resource.h	Файл содержит определения идентификаторов ресурсов приложения, например, идентификаторы строк меню
StdAfx.h,	Использование этих файлов позволяет ускорить процесс

StdAfx.cpp	повторного построения проекта. Более подробное описание файлов представлено ниже

Ресурсы приложения

Рассмотрим ресурсы, которые MFC AppWizard создал для нашего приложения. Откройте страницу RecourceView в окне проекта Project Workspace. В нем отображается полный список всех ресурсов приложения (рис. 1.3).

Рис. 1.3. Окно Project Workspace, ресурсы приложения

Сравните эти ресурсы с ресурсами приложения с однооконным интерфейсом (том 24 серии "Библиотека системного программиста"). Вы заметите, что в состав приложения с многооконным интерфейсом входит больше ресурсов. Так, например, для многооконного приложения определены два меню, две пиктограммы, больше размер таблицы текстовых строк.

Национальные ресурсы

Обратите внимание, что все ресурсы, представленные на странице RecourceView в окне проекта Project Workspace, английские. К сожалению, MFC AppWizard из доступных нам версий Microsoft Visual C++ не позволяет выбрать для создаваемого приложения русские ресурсы (язык для ресурсов выбирается в первой панели MFC AppWizard – Step 1, во время определения свойств приложения). Поэтому для приложения Multi и всех других приложений, созданных с помощью MFC AppWizard, мы выбрали английский язык.

Если вы в Control Panel с помощью приложения Regional Settings выбрали русский язык, то в некоторых случаях ClassWizard может работать неправильно. Например, если вы добавите к английской диалоговой панели новые органы управления, то ClassWizard не позволит автоматически

привязать к ним переменные. Возникнут также сложности при использовании русского текста в строковых ресурсах, помеченных как английские. Чтобы избежать этих проблем, измените язык, используемый для ресурсов. Для этого достаточно в окне Project Workspace щелкнуть по идентификатору ресурса правой кнопкой мыши и выбрать из открывшегося контекстного меню строку Properties. На экране появится диалоговая панель со свойствами выбранного ресурса. Измените в ней язык ресурса, выбрав из списка Language строку Russian.

Шаблон меню

Для многооконного приложения в ресурсах проекта определены два меню с идентификаторами IDR_MAINFRAME и IDR_MULTITYPE. Приложение использует одно из этих меню, в зависимости от того, открыт документ или нет.

Меню с идентификатором IDR_MAINFRAME используется, если в приложении не открыт ни один документ. Как видите, идентификатор меню совпадает с идентификатором меню приложения с однооконным интерфейсом, однако строки этих меню различаются:

```
////////////////////////////////////////////////////////////// Меню IDR_MAINFRAME
IDR_MAINFRAME MENU PRELOAD DISCARDABLE
BEGIN
 POPUP "&File"
 BEGIN
  MENUITEM "&New\tCtrl+N",   ID_FILE_NEW
  MENUITEM "&Open...\tCtrl+O",ID_FILE_OPEN
  MENUITEM SEPARATOR
  MENUITEM "P&rint Setup...", ID_FILE_PRINT_SETUP
  MENUITEM SEPARATOR
  MENUITEM "Recent File",    ID_FILE_MRU_FILE1, GRAYED
  MENUITEM SEPARATOR
 MENUITEM "E&xit",          ID_APP_EXIT
 END
 POPUP "&View"
 BEGIN
  MENUITEM "&Toolbar",       ID_VIEW_TOOLBAR
  MENUITEM "&Status Bar",    ID_VIEW_STATUS_BAR
 END
 POPUP "&Help"
 BEGIN
  MENUITEM "&About Multi...", ID_APP_ABOUT
 END
END
```

Меню, имеющее идентификатор IDR_MULTITYPE, отображается, когда пользователь создает новый документ или открывает документ, уже записанный в файле на диске.

Создание проекта программы

1. Создайте новый проект(у меня MDI), использующая MDI интерфейс с поддержкой MFC. Все шесть шагов в MFC AppWizard оставте без изменения.

2. Если вы сделали всё правильно, то создадутся пять классов : CMDIApp, CMainFrame, CChildFrame, CMDIDoc и CMDIView. В классе документов CMDIDoc вы пишите код для поддержки данных программы, а в классе представления CMDIView - код, отвечающий за то, что вы видите на экране. Вы будете писать код в функциях-элементах только этих двух классов.

3. Объявляем элементы данных класса документа. Их будет два : координаты круга по X и по Y. Для этого открываем файл CMDIDoc.h и изменяем объявление класса CMDIDoc следующим образом:

class CMDIDoc : public CDocument { protected:

// create from serialization only CMDIDoc(); DECLARE_DYNCREATE(CMDIDoc) // Attributes public: int m_X; // координаты круга по x int m_Y;

// координаты круга по y // Operations ... ...

4. Объявляем элементы данных класса представления. Их будет тоже два : координаты круга по X и по Y. Для этого открываем файл CMDIView.h и изменяем объявление класса CMDIView следующим образом:

class CMDIView : public CView { protected:

// create from serialization only CMDIView(); DECLARE_DYNCREATE(CMDIView) // Attributes public: CMDIDoc* GetDocument(); int m_X;

// координаты круга по x int m_Y;

// координаты круга по y // Operations ... ...

Как вы видите, имена переменных могут совпадать(обычно так и делается).

5. Инициализируем элементы данных класса документа. Для этого откройте файл MDIDoc.cpp, найдите в нём функцию OnNewDocument() и напишите в ней следующий код:

BOOL CMDIDoc::OnNewDocument() { if (!CDocument::OnNewDocument()) return FALSE; m_X = 100; // начальное положение по X=100 m_Y = 100;

// начальное положение по Y=100

// TODO: add reinitialization code here // (SDI documents will reuse this document) return TRUE; }

6. Инициализируем элементы данных класса представления. Для этого нужно создать функцию-элемент OnInitialUpdate() класса представления:

Выберите ClassWizard в меню View. На странице Message Maps выберите следующие события:

Class neme : CMDIView Object ID : CMDIView Message : OnInitialUpdate

и нажмите на кнопку Add Function

Напишите следующий код в функцию OnInitialUpdate():

void CMDIView::OnInitialUpdate() { CView::OnInitialUpdate();

// TODO: Add your specialized code here and/or call the base class CMDIDoc* pDoc = GetDocument();

// получить указатель на документ

// обновить элементы данных представления

// соответствующими значениями документа. m_X = pDoc->m_X; m_Y = pDoc->m_Y; pDoc->SetTitle("ANDY");

// всем документам даётся название ANDY }

7. Теперь напишем код для вывода круга на экран.

Функция OnDraw() класса представления автоматически выполняется всякий раз, когда нужно вывести окно документа.

Напишите следующий код в функции OnDraw() :

void CMDIView::OnDraw(CDC* pDC) { CMDIDoc* pDoc = GetDocument(); ASSERT_VALID(pDoc);

// TODO: add draw code for native data here pDC->Ellipse(m_X - 20, m_Y - 20, m_X + 20, m_Y + 20);

// рисуем круг диаметром 20 }

8. Напишем код для сохранения и считывания данных из файла.

Откройте файл MDIDoc.cpp, найдите в нём функцию Serialize() и измените её:

void CMDIDoc::Serialize(CArchive& ar) { if (ar.IsStoring()) {

// TODO: add storing code here(это выполняется,

если выбрать SAVE) ar<<m_X;

// записываем m_X в выбранный файл ar<<m_Y;

// записываем m_Y в выбранный файл } else {

// TODO: add loading code here (это выполняется,

если выбрать OPEN) ar>>m_X;

// считываем значение из выбранного файла в m_X ar>>m_Y;

// считываем значение из выбранного файла в m_Y } }

9. Часто бывает нужно изменить некоторые параметры программы, такие как заголовок главного окна или тип файла по умолчанию, который выводится в диалоговых панелях SAVE и OPEN. Для этого нужно выбрать закладку ResourceView и открыть пункт String Table. Вы увидите список переменных проекта(три колонки : ID, Value и Caption).

IDR_MAINFRAME - заголовок главного окна (изменяется в поле Caption)

IDR_MCIRCLTYPE - тип файла по умолчанию, вы увидите 6 подстрок разделёнными знаком n. Третья и четвёртая подстроки определяют тип документа по умолчанию. У меня CIR FILES(*.cir) и .cir соответственно. Вы можете поставить свои значения.

10. Теперь создадим кнопку в панеле инструментов. Для этого нужно выбрать закладку ResourceView и открыть пункт Toolbar. Вы увидите панель инструментов в режиме редактирования. Нажмите на самую правую кнопку(пунктирный квадрат), ниже нарисуйте кнопку по вашему усмотрению. Теперь дважды нажмете на вашу кнопку и введите ID: ID_MYBUTTON и Prompt: Изменение координат кругаИзменение координат круга. Ну вот и всё, кнопка готова. Теперь нужно создать функцию, которая будет выполняться при нажатии на вашу кнопку :

Выберите пункт меню View далее ClassWizard, выберите закладку Message Maps, Project: MDI, Class name: CMDIView, Object IDs: ID_MYBUTTON, Message: COMMAND и нажмите на кнопку Add Function. В ответ создастся функция void CMDIView::OnMybutton().

11. Теперь по аналогии с главой 15 создадим собственное диалоговое окно с ID: IDD_MY_DIALOG и классом CMyDialog и разместим в нём четыре Edit Box с переменными типа INT: m_DX - текущая позиция по X, m_DY - текущая позиция по Y, m_DXN - новая позиция по X, m_DYN - новая позиция по Y. Не забудте написать #include "MyDialog.h" в файлах MDIDoc.cpp и MDIView.cpp.

12. Теперь напишем код в функцие OnMybutton().

```
void CMDIView::OnMybutton() {
// TODO: Add your command handler code here CMDIDoc* pDoc = GetDocument();
// получаем указатель на документ CMyDialog MyD;
// создаём переменную класса CMyDialog MyD.m_DX = MyD.m_DXN = pDoc->m_X;
// инициализмруем переменные диалога MyD.m_DY = MyD.m_DYN = pDoc->m_Y; .
// MyD.DoModal();
// создаём новый диалог pDoc->m_X = MyD.m_DXN;
// получаем новые значения pDoc->m_Y = MyD.m_DYN;
// OnInitialUpdate();
// синхронизируем данные Invalidate( TRUE );
// перерисовываем экран( вызов OnDraw() ) pDoc->SetModifiedFlag();
// ставим флаг изменения документа }
```

Лабораторная работа № 10(2 часа)
Использование списков и полей со списками

Список строк Visual Studio **TStringList** - это структура данных, напоминающая компонент **ListBox**, но не визуальная, а просто хранящая в памяти и имеющая свойства и методы для работы со строками типа **TString**.

Для работы со списком строк типа **TStringList** его сначала необходимо создать с помощью конструктора - **Create**:

```
var StrList: TStringList;
begin
 StrList:=TStringList.Create;
```

Теперь нужно описать способ хранения строк в списке. Список строк типа TStringList может сортировать добавляемые строки или хранить без сортировки, а также может игнорировать попытки добавления новой строки при наличии уже сохранённого дубликата:

```
StrList.Sorted:=True;          //True - сортировать, False - не сортировать
StrList.Duplicates:=dupIgnore; //dupAccept - сохранять дубликаты (значение
                               по умолчанию), dupIgnore - игнорировать,
                               dupError - вызвать сообщение об ошибке
```

Теперь можно добавить в список строки типа TString:

```
StrList.Add('Новая строка');          //Добавление в конец списка (или в
                                      порядке сортировки). Возвращается
                                      индекс строки
StrList.Insert(Index, 'Новая строка'); //Строка добавляется на позицию с
                                      номером Index. Если список
                                      отсортирован, то возникает
                                      исключительная ситуация.
```

Удалить строку:

```
StrList.Delete(Index);  // Удаление строки с номером Index;
```

Количество строк в списке:

```
N:=StrList.Count;
```

Доступ к строке с номером Index:

```
Str:=StrList[Index];
```

Index отсчитывается от 0, поэтому номер последней строки:

```
Last:=StrList.Count-1;
```

Очищаем список строк:

```
StrList.Clear;
```

Если список отсортирован, есть метод для поиска:

```
StrList.Find(S, Index);      // В случае удачного поиска возвращает True, и индекс строки S в переменной Index
```

Если список не отсортирован, то тоже есть функция поиска:

```
I:=StrList.IndexOf(Text);    // Возвращает индекс строки с заданным текстом Text, или -1, если поиск неудачен
```

В конце работы со списком его необходимо удалить из памяти:

```
StrList.Free;
end;
```

Как видим, список типа TStringList обладает ценным свойством, которым вряд ли обладает какой-либо другой объект в Visual Studio, а именно возможностью обнаруживать дубликаты строк. Как узнать, что строка уже содержится в списке? Вот алгоритм:

1. Запретить списку запись дубликатов, что возможно только при задании сортировки списка. Это делается один раз, при создании списка, например, по событию Формы OnCreate:
   ```
   StrList.Duplicates:=dupIgnore;
   StrList.Sorted:=True;
   ```
2. Сохранить в переменной количество строк в списке:
   ```
   begin
     Count:=StrLst.Count;
   ```
3. Попробовать записать новую строку:
   ```
     StrList.Add(NewLine);
   ```
4. Сравнить количество строк в списке с прежним значением. Если оно совпадёт с количеством перед записью, то такая строка в списке уже есть:

```
    if StrList.Count=Count
      then ShowMessage('Попытка добавления дубликата');
    end;
```

Аналогичный, алгоритм реализации этой процедуры связан с использованием метода обработки исключений. Если вместо **StrList.Duplicates:=dupIgnore** для обработки дубликатов использовать значение **dupError**, это вызовет исключительную ситуацию при попытке добавить дубликат. Соответственно, при добавлении оригинальной строки программа идёт дальше, а при попытке добавления дубликата переходит к операторам секции **except/end**.

Эти методы работают в случае если список отсортирован. Однако есть задачи, где сортировка не нужна, а находить значение в списке всё равно необходимо. В этом случае нужно пользоваться функцией **IndexOf**.

Рассмотрим пример

Есть таблица, содержащая список студентов с указанием их роста и веса. Нужно выделить группу студентов, чей рост или вес встречаются наиболее часто.
Алгоритм решения задачи будет такой. Создадим три списка строк. В цикле будем просматривать таблицу и добавлять в первый список строк значение анализируемого параметра, если его ещё в списке нет. Во второй список будем сохранять фамилии студентов, обладающих данным ростом или весом, а в третьем - в соответствующей строке подсчитывать сколько раз это значение уже встретилось. Вот что должно получиться:

Исходная таблица			Первый список	Второй список	Третий список
Фамилия	**Рост**	**Вес**	Анализ по росту		
Иванов	179	80	179	Иванов, Сидоров	2
Петров	180	82	180	Петров	1
Сидоров	179	78	175	Ковалёв	1
Ковалёв	175	78			

Переходя к новой строке в исходной таблице, мы анализируем нужный параметр, рост или вес - ищем в первом списке. Если такого значения у нас пока нет, мы в каждый список добавляем по строке, в первый записываем значение параметра, во второй - фамилию, а в третий - пишем единичку. А если такое значение параметра уже есть, то в соответствующую строку второго списка добавляем через запятую очередную фамилию, а в третьем значение в этой строке увеличиваем на 1. Естественно, списки строк не должны сортировать вводимые значения, иначе данные перепутаются!

Дойдя до конца таблицы, нам останется в третьем списке найти строку с максимальным числом, и вывести как результат содержимое этой строки второго списка.

Сортировка в StringList'e

С помощью StringList можно реализовать интересный вариант сортировки чисел. Хотя StringList сортирует строки, он также будет сортировать и записанные в него в виде строк числа. Естественно, напрямую отсортировать числа таким образом не получится, так как строка, например, '100' будет меньше, чем строка '20', что для чисел 100 и 20, естественно, неверно. Однако достаточно выровнять длину строк добавлением к целой части слева, а к дробной справа пробелов ' ', чтобы сортировка сработала верно. Программисты, создавшие класс TStringList, наделили его очень быстрым алгоритмом сортировки (к тому же реализованным, видимо, с использованием низкоуровневого программирования), чем мы просто не можем не воспользоваться. По сравнению с сортировкой пузырьком, которая 100 000 чисел сортирует за 32 с половиной секунды на моём компе (или 24 сек улучшенный алгоритм), сортировка в StringList'e длится всего 530 мсек! Вот процедура для целых чисел:

Здесь Grid - обычная таблица типа TStringGrid, предварительно заполненная целыми числами. Продолжительность сортировки возвращается в заголовке Формы.

Задание:
Разработать программу сортировки целых чисел, используя StringList.

Лабораторная работа № 11 (2 часа)
Работа с датой и временем

Функции для работы со временем: чтобы узнать текущее время есть функция - **TimeString**, чтобы узнать дату - **DateString**.. На форме размести таймер(Enabled = True, Interval = 1000), и 2 метки(Label)(AutoSize = True(чтобы размеры метки автоматически регулировались, взависимости от текста в метке), Text = "").

Вот код:

```
Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
Label1.Text = "Время: " & TimeString
Label2.Text = "Дата: " & DateString
End Sub
```

Помимо этих двух функций в VB.Net сохранились функции для работы со временем: **Hour()**(часы), **Minute()**(минуты), **Second()**(секунды). Эти функции вырезают часы, минуты, секунды из указанно времени. Например, *Minute(TimeString)* - вырезает минуты из текущего времени. И с прежних версий VB остались функции для работы с датами: **Day()**(год), **Month()**(месяц), **Year**(год). Эти функции вырезают из текущей даты: день, месяц, год. Например, *Month(DateString)*, вырезайет из текущей даты месяц.

Еще сохранилась функция ***WeekDay***, она нужна для определения дня недели(например Вторник). Сейчас мы сделаем программку, которая будет определять день недели, на форму добавь еще одну метку(Label3), свойства такие же как и у предыдущих меток), вот код:

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load ' При загрузке формы:

Dim Den As Integer

Den = Weekday(DateString) ' Определяем день недели
If Den = 0 Then Label3.Text = "Воскресенье"
If Den = 1 Then Label3.Text = "Понедельник"
If Den = 2 Then Label3.Text = "Вторник"
If Den = 3 Then Label3.Text = "Среда"
If Den = 4 Then Label3.Text = "Четверг"
If Den = 5 Then Label3.Text = "Пятница"
If Den = 6 Then Label3.Text = "Суббота"

End Sub
```

Задание

В программе предыдущей работы добавить вывод текущего времени и даты.

Лабораторная работа № 12 (2 часа)
Использование гиперссылок в программах

Алгоритм создания гиперссылки:

1. ставим на форму метку (TLabel),
2. приводим ее внешний вид к привычному виду гиперссылки в нашем браузере (рассмотрим на примере IE);
3. пишем обработчик события OnClick.

Чтобы можно было постоянно использовать гиперссылку в программах, создадим компонент. Ставим на форму нашего проекта метку (TLabel), пусть ее имя останется Label1. Теперь мы напишем обработчик события OnClick, для нее:

```
procedure TForm1.Label1Click(Sender: TObject);
begin
 ShellExecute(handle, 'open', 'http://Visual Studioworld.narod.ru/', nil, nil, SW_SHOW);
end;
```

Функция ShellExecute предназначена для открытия или печати файла, как исполняемого, так и документа. Первый параметр это handle родительского окна, второй параметр - строка, указывает, что надо сделать с файлом, третий параметр содержит имя открываемого файла, четвертый параметр указывает дополнительные параметры запуска исполняемого файла, пятый параметр определяет директорию по умолчанию, последний параметр определяет где будет отображен файл после октрытия.

Если Вы уже попробовали запустить приведенный код, то скорее всего у Вас ничего не вышло, потому что функция ShellExecute, находится в модуле ShellAPI, который конечно же надо добавить в секцию uses, кода нашего приложения.

Теперь разберем параметры относительно нашего случая:

handle - это дескриптор главной формы (аналогично Form1.handle)
open - тип действия с файлом. Нам надо его открыть.
http://Visual Studioworld.narod.ru/ - имя файла, который надо открыть. У нас это может быть гиперссылка, содержащая абсолютный URI.

nil - здесь никаких дополнительных параметров открытия файла не должно быть, поэтому nil.
nil - директория по умолчанию нас так же не интересует.
SW_SHOW - активирует окно и отображает его с текущими размерами и положением. Об остальных режимах можно узнать в хелпе (о функции ShellExecute).

Надо отметить, что второй и третий параметры функции являются нуль терминированными строками, т.е. строками типа PChar, поэтому для использования в функции имени файла, полученного из OpenDialog1, нужно использовать PChar(OpenDialog1.Filename).

Поскольку в браузере (при настройках по умолчанию) ссылка меняет цвет в зависимости от своего состояния и действий пользователя. Мы тоже сделаем так. Для этого заведем три константы (в них будут определяться цвета), которые надо поместить в раздел Implementation:

```
...
implementation
const
  link = clBlue; //цвет ссылки
  alink = clRed; //цвет ссылки в момент нажатия
  vlink = clPurple; // цвет посещеной ссылки
{$R *.DFM}
...
```

Теперь в обработчике события формы OnCreate надо написать:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Label1.font.Color := link;
end;
```

В обработчике события метки OnMouseDown мы напишем:

```
procedure TForm1.Label1MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  Label1.Font.Color := alink;
end;
```

А в обработчике события OnMouseUp нашей метки мы напишем:

```
procedure TForm1.Label1MouseUp(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  Label1.Font.Color := vlink;
end;
```

Для придания полной реалистичности нашей ссылке, надо установить свойство метки Cursor в crHandPoint. Т.е. при наведении на ссылку указатель будет иметь вид привычной нам кисти руки с вытянутым указательным пальцем и сделать ссылку подчеркнутой.

Напишем компонент.

В качестве родительского класса (Ancestor Type) мы конечно же должны выбрать TLabel.

Полный код модуля компонента Link класса Tlink (текст модуля надо сохранить в файле Link.pas):

```
unit Link;

interface

uses
  Windows, SysUtils, Classes, Forms, StdCtrls, ShellAPI, Menus,
  Controls, Graphics;

type
  TLink = class(TLabel)

  private
    //поле для хранения URL
    FURL: string;

    // поля для хранения настроек цветов ссылки
    FlinkColor, FAlinkColor, FVlinkColor: TColor;

  protected

  public
    //конструктор и деструктор
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
```

```
    //события, которые нам надо переопределить
    procedure Click; override;
    procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
      override;
    procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
      override;

  published
    //свойства, которые будут видны в Object Inspector
    property URL: string read FURL write FURL;
    property LinkColor: TColor read FLinkColor write FLinkColor default clBlue;
    property ALinkColor: TColor read FALinkColor write FALinkColor default
      clRed;
    property VLinkColor: TColor read FVLinkColor write FVLinkColor default
      clPurple;

  end;

procedure Register;

implementation

constructor TLink.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  //устанавливаем цвета
  FALinkColor := clRed;
  FLinkColor := clBlue;
  FVLinkColor := clPurple;
  //сделаем метку синей и подчеркнутой
  with Font do
  begin
    Color := FLinkColor;
    Style := [fsUnderline];
  end;
end;

procedure TLink.Click;
begin
```

```
  inherited Click;
  //если поле URL заполнено, то перейдем по ссылке
  if FURL <> '' then
    ShellExecute(GetDesktopWindow(), 'open', PChar(FURl), nil, nil, SW_SHOW);
end;

procedure TLink.MouseDown(Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
begin
  if Button = mbLeft then
  begin
    FLinkColor := Font.Color;
    Font.Color := FALinkColor;
  end;
  inherited;
end;

procedure TLink.MouseUp(Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
begin
  if Button = mbLeft then
    Font.Color := FVLinkColor;
  inherited;
end;

destructor TLink.Destroy;
begin
  //уничтожим компонент
  inherited;
end;

procedure Register;
begin
  //надо, чтобы компонент появился в палитре
  RegisterComponents('http://Visual Studioworld.narod.ru/', [TLink]);
end;

end.
```

Лабораторная работа № 13 (2 часа)
Работа с графикой

Будем использовать Graphics Device Interface (GDI+) - подсистему Windows, ответственную за вывод графики и текста на дисплей и принтер. Именно CGI+ обеспечивает вывод на экран всего того, что видит пользователь Windows в окне монитора. GDI+ является базовым способом вывода графики в Windows.

CGI+ - это библиотека функций и программных средств на базе API, позволяющая приложениям абстрагироваться от особенностей конкретного графического оборудования. .Net Framework использует расширенный графический интерфейс GDI+.

С графикой Windows (при использованием GDI+) связано понятия контекста устройства (device context, DC). Это структура данных, содержащая информацию о параметрах и атрибутах вывода графики на устройство (дисплей, принтер...). .Net Framework освобождает программиста от необходимости обращения к DC. Основной объект для графического отображения Graphics - класс имитации поверхности рисования в GDI+. Класс не имеет конструкторов по причине того, что объекты этого класса зависят от контекста конкретных устройств вывода. Создаются объекты специальными методами разных классов, например, он может быть создан из объекта Bitmap, или к нему можно получить доступ, как к некоторому объекту, инкапсулированному в некоторые контролы, в том числе, и в объект формы приложения. Метод CreateGraphics класса Control - наследника класса Form - возвращает объект, ассоциированный с выводом графики на форму.

Рассмотрим простейшие примеры. Создадим решение Windows приложения с одной кнопкой и следующим обработчиком ее нажатия:

```
private void button1_Click(object sender, EventArgs e)
{
 //Создаем битовую матрицу
 Bitmap bitmap = new Bitmap(ClientSize.Width,
ClientSize.Height);
 //Создаем объект класса Graphics на основе битовой
матрицы
 Graphics graph = Graphics.FromImage(bitmap);
 //Рисуем линию на поверхности, Graphics использует в
качестве нее
 //битовую матрицу
 graph.DrawLine(new Pen(Color.Red,2), 0, 0,
ClientSize.Width, ClientSize.Height);
```

```
 //Присваиваем нарисованноу свойству формы
 BackgroundImage = bitmap;
 graph.Dispose();
}
```

Одинакового эффекта можно добиться, если использовать обработчики некоторых событий, которым передается объект класса Graphics как аргумент (например, обработчик события Paint формы приложения):

```
private void Form1_Paint(object sender, PaintEventArgs
e)
{
 e.Graphics.DrawLine(new Pen(Color.Red, 2), 0, 0,
ClientSize.Width, ClientSize.Height);
}
```

Одинакового эффекта (Рис.1.), можно добиться и при непосредственном создании объекта Graphics:

```
private void button1_Click(object sender, EventArgs e)
{
 Graphics graph = CreateGraphics();
 graph.DrawLine(new Pen(Color.Red, 2), 0, 0,
ClientSize.Width, ClientSize.Height);
 graph.Dispose();
}
```

А так можно рисовать (писать) на кнопке и на других контролах, для которых может быть создан обработчик события Paint :

```
private void button1_Paint(object sender,
PaintEventArgs e)
{
 e.Graphics.DrawLine(new Pen(Color.Red, 2), 0, 0,
button1.Width, button1.Height);
}
```

Класс Graphics находятся в пространстве имен Drawing (хотя, забегая вперед, нам понадобится и пространство имен Drawing.Drawing2D, по сему, целесообразно сразу добавить их в решение).

```
using System.Drawing;
```

```
using System.Drawing.Drawing2D;
```

Для того чтобы рисовать в окне формы Windows приложения, необходимо иметь не только связанный с этим окном объект Graphics, как холст для рисования, но и инструменты рисования. Используют два основных инструмента: карандаш и кисть. Карандаш мы использовали в примерах, приведенных выше. В конструкторе класса Pen задается цвет пера и можно задать толщину (по умолчанию 1). Класс Brush определяет кисти для рисования. Это абстрактный класс. Кисти этого класса создать нельзя, хотя можно создавать кисти классов-потомков Brush:

- SolidBrush - сплошная закраска цветом кисти;
- TextureBrush - наложение картинки (image) на область закраски;
- HatchBrush - закраска области предопределенным узором;
- LinearGradientBrush - сплошная закраска с переходом цвета кисти (градиентная закраска);
- PathGradientBrush -сплошная закраска с переходом цвета кисти по специальному алгоритму.

```
private void button1_Click(object sender, EventArgs e)
{
 Graphics graph = CreateGraphics();
 graph.DrawLine(new Pen(Color.Blue, 2), 0, 0,
ClientSize.Width, ClientSize.Height);
 graph.FillEllipse(new HatchBrush(HatchStyle.Percent05,
Color.Silver), 0, 0,

ClientSize.Width, ClientSize.Height);
 graph.Dispose();
}
```

Как видно из приведенных примеров, существует несколько способов отображение графики. Основное их отличие - при использовании битовой матрицы и свойств контролов изображение рисуется один раз и не исчезает (нет необходимости его вновь рисовать) при перерисовки формы.

Здесь нет необходимости подробно описывать все методы объекта Graphics, их можно легко увидеть из контекстной подсказки, как обычно, поставив точку после написания имени объекта. По этой же причине нет необходимости перечислять и преопределенные цвета карандашей (Color.) и кистей (Brushes.)

Отметим еще ***два важных момента***, которые нам понадобятся далее:

1. Круговые фигуры рисуются при указании в качестве стартовой точки левого верхнего угла (см. последний пример);

2. Использование событий Paint для отображения графики не всегда оправдано из-за необходимости постоянной перерисовки изображения при перерисовке формы.

Рисование непосредственно на форме не всегда является и не есть хороший тон, кода Visual Studio предлагает специальный контрол, который подходит для вывода графической информации и обладает всеми преимуществами контролов: программное позиционирование и масштабирование без перерисовки формы, возможность выполнять Stretch Image и множество других полезных свойств и событий. Кроме того простой доступ к Graphics, аналогично через событии Paint и возможность использования битовых карт Bitmap для создания объекта Graphics, с последующим переносом их в контрол (аналогично, как мы делали это в первом примере):

```
pictureBox1.Image = bitmap;
```

Далее мы будем использовать именно PictureBox, как объект для отображения графической информации.

Задание:

Выполнить примеры.

Лабораторная работа № 14 (2 часа)
Создание программы построения и масштабирования графиков функций

Стоит задача создать класс для отображения графической информации, который бы мог стать базовым классом для работы с графикой, позволял бы не только выводить различные виды графиков и обладал бы гибкостью настройки форм отображения, но и оставался открытым для дальнейшего его расширения.

Конечная цель - помещение созданного графического изображения в элемент управления PictureBox.

2.3. Исходные данные

Исходные данные перед их отображением могут находиться где угодно (файл, таблица базы данных...). Однако рассматривать чтения из базы данных или из файла значений графиков - только засорять отображение материала. Мы всегда можем прочитать данные с любого источника в массив значений. Автор предпочитает работать со строковым массивом, как позволяющим хранить цифровые и текстовые значения. В примерах, приводимых ниже, используется **массив строк string[,] rgsValues**. В программе, о которой шла выше речь, этот массив использован для настройки параметров, отображаемых на графике. Заполнять массив будем с помощью датчиков случайных чисел:

```
private int viNumInRg=20;//20 - начальное значение
private string[,] rgsValues=null;

private int iCreateRg()
{
 Random rnd = new Random(DateTime.Now.Millisecond);
 Random rnd1 = new Random(DateTime.Now.Millisecond+5);
 rgsValues = new string[viNumInRg, 2];
 for (int i = 0; i < viNumInRg; i++)
 {
  rgsValues[i, 0] = Convert.ToString(((float)(rnd.Next(0, 10) *
100) +
                                    (float)rnd1.Next(0, 99)) /
(float)100);
  rgsValues[i, 1] = "I-" + Convert.ToString(i+1);
 }
```

Предполагается, что значение переменной, определяющей размерность массива, хранится в настройках и устанавливается на этапе загрузки приложения.

2.4. Проект решения

Создадим простой проект WindowsApplication решения с любым именем (у меня graph1). Поместим на форму три кнопки, в свойствах "Текст" которых напишем соответственно: "Линейная диаграмма", "Гистонрамма" и "Круговая диаграмма". Ниже кнопок поместим контрол PictureBox. Подберем удобное для себя расположение кнопок и PictureBox (в реальных программах для размещения удобнее использовать контролы TableLayoutPanel, но сейчас нас интересует графика, а не размещение).

В окне Solutation Explorer кликаем правой кнопкой мышки на узле решения (у меня graph1) и в контекстном меню выбираем Add\New Item. В окне Templates выбираем Class, даем ему имя, например PaintCl.cs и нажимаем кнопку Add. Будет создан пустой класс.

```
using System;
using System.Collections.Generic;
using System.Text;
namespace graph1
{
 class PaintCl
 {
 }
}
```

Задачей будет постепенное наполнение этого класса при минимуме добавления кода в основной файл кода приложения - Form1.cs.

Для начала создадим обработчик события нажатия кнопки "Линейный график" (клик мышкой на кнопке), а также обработчики для событий Load и FormClozed (первый можно кликом мышки на форме, второй через окно Properties формы - закладка Events - клик в окошечке против события FormClosed). Слегка преобразуем код, как показано ниже:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace graph1
{
 public partial class Form1 : Form
 {
  private int viNumButton = 0;
  private int viNumInRg=20;//20 - начальное значение
  private string[,] rgsValues=null;

  public Form1()
  {
```

```
        InitializeComponent();
    }


    private void Form1_Load(object sender, EventArgs e)
    {
        //Здесь при создании реальной программы необходимо
        //будет предусмотреть восстановление сохраненных
        //параметров для приложения и графиков
    }
    private void Form1_FormClosed(object sender,
FormClosedEventArgs e)
    {
        //Здесь при создании реальной программы необходимо
        //будет предусмотреть сохранение параметров
        //для приложения и графиков
    }


    #region Создание массива значений
    private void vCreateRg()
    {
     Random rnd = new Random(DateTime.Now.Millisecond);
     Random rnd1 = new Random(DateTime.Now.Millisecond+5);
     rgsValues = new string[viNumInRg, 2];
     for (int i = 0; i < viNumInRg; i++)
     {
      rgsValues[i, 0] = Convert.ToString(((float)(rnd.Next(0, 10)
* 100) +
                                    (float)rnd1.Next(0, 99)) /
(float)100);
      rgsValues[i, 1] = "I-" + Convert.ToString(i+1);
     }
    }
    #endregion

    #region создание линейного графика
    private void button1_Click(object sender, EventArgs e)
    {
       viNumButton = 1;
       vCreateLinGr();
    }
    private void vCreateLinGr()
    {
     //Создаем массив значений для вывода на графике
     vCreateRg();

    }
    #endregion
  }
}
```

Назначение переменной viNumButton, будет ясно далее. Массив значений у нас создан. Осталось нарисовать по значениям массива график, используя класс.

2.5. Конструкторы класса

Начало класса - конструктор и закрытые переменные. В классе лучше иметь несколько конструкторов. Например, таких как приведено в коде ниже. И, естественно, необходимо сразу определить основные объекты для графики: битовую матрицу, объект Graphiks, шрифт, кисть и перо (о чем мы говорили выше), а также переменные для сохранения размеров холста:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Drawing;
using System.Drawing.Drawing2D;
namespace graph1
{
 class PaintCl
 {
   //Основные объекты для рисования
   private Bitmap bmp = null;
   private Graphics graph = null;
   private Font objFont = new Font("Arial", 8, FontStyle.Bold);
   private Brush objBrush = Brushes.Black;
   private Pen objPenLine = new Pen(Color.Black, 1);
   //Размеры холста
   private int viX = 200;
   private int viY = 100;

   #region Конструкторы
   //Первый
   public PaintCl()
   {

   }
   //Второй
   public PaintCl(int a, int b)
   {
     bmp = new Bitmap(a, b);
     //Создаем объект Graphics на основе битовой матрицы
     graph = Graphics.FromImage(bmp);
     //Запоминаем размеры холста
     viX = a;
     viY = b;
   }
   #endregion
 }
}
```

Пустой конструктор, как правило, ничего не дает и потребует в дальнейшем введения дополнительных функций для инициализации объектов отображения графики. Для нашей цели он не удобен и мы можем его вычеркнуть (не использовать). Однако если мы, например, захотим передавать в качестве объекта для рисования некоторый рисунок, то вынуждены будем воспользоваться или новым сложным конструктором или добавить в класс всего лишь функцию передачи рисунка, так как функции инициализации объектов рисования так или иначе у нас уже будут. И, хотя, нагрузка на конструктор при инициализации основных объектов не страхует от необходимости иметь функции переопределения параметров объектов рисования, все же второй конструктом мне кажется более предпочтительным.

2.6. Создание объекта для рисования

Используем второй конструктор и создадим и инициализируем в классе сразу все объекты, которые нам необходимы:

```
private void vCreateLinGr()
{
   //Создаем массив значений для вывода на графике
   vCreateRg();
   //Создаем класс и передаем ему размер холсты
   PaintCl clPaint = new PaintCl(pictureBox1.Width,
pictureBox1.Height);
   //Передадим фон холста в класс
   clPaint.vSetBackground(Color.Red);
   //Принимаем нарисованное в pictureBox
   pictureBox1.Image = clPaint.Bmp;
}
```

Таким образом, нам понадобятся в классе еще две функции: установки фона холста и приема изображения из класса. Добавим их в класс:

```
#region Установка цвет фона диаграммы
public void vSetBackground(Color bcl)
{
 graph.Clear(bcl);
}
#endregion

#region Доступ к переменным класса
public Bitmap Bmp
{
 get {return bmp;}
}
#endregion
```

Выполним решение на данном этапе.

Описанный выше код показал сам принцип создания изображения с использованием PictureBox и создаваемого нами и претендующего на универсальность класса для рисования графиков.

2.7. Рисование осей

Добавим в классе переменные для хранения отступов от краев холста (они нам еще понадобятся не один раз).

```
//Отступы от краев холста
private int viDeltaaxisL = 50;
private int viDeltaaxisR = 50;
private int viDeltaaxisH = 20;
```

Добавим функцию рисования осей и функции запоминания цвета и толщины осей. Функция выполняет простую задачу - рисует две линии и при необходимости стрелочки на конце осей:

```
#region Рисование Осей
//Параметры вызоыва: отступы слева - deltaaxisL, справа -
deltaaxisR,
//сверху(снизу) - deltaaxisH, Цвет осей - colorpenaxis, толщина
пера - widthpen,
//нужны ли стрелки - fArrow (true - да)
public void vDravAxis(int deltaaxisL, int deltaaxisR,
             int deltaaxisH, Color colorpenaxis, int widthpen,
bool fArrow)
{
 //Запоминаем отступы
 viDeltaaxisL = deltaaxisL;
 viDeltaaxisR = deltaaxisR;
 viDeltaaxisH = deltaaxisH;
 //Запоминаем цвет осей и толщину
 vSetPenColorLine(colorpenaxis);
 if (widthpen > 0) vSetPenWidthLine(widthpen);
 //Точка начала рисования по x и y
 int x = deltaaxisL;
 int y = viY - deltaaxisH;
 int x1 = viX - deltaaxisR;
 int y1 = deltaaxisH;
 //Переменная определения длины стрелок
 int d = 0;
 if(fArrow) d = widthpen * 10;
 //Оси на d пикселей длинней для стрелок
 graph.DrawLine(objPenLine, x, y, x1 + d, y);
 graph.DrawLine(objPenLine, x, y, x, y1 - d);
 //Надо рисовать стрелки
 if (fArrow)
 {
  int a = 10 * (int)objPenLine.Width;
  int b = 2 * (int)objPenLine.Width;
  int x2 = x1 - a;
```

```
  int y2 = y + b;
  //Стрелки
  graph.DrawLine(objPenLine, x1 + 20, y, x2 + d, y2);
  y2 = y - b;
  graph.DrawLine(objPenLine, x1 + 20, y, x2 + d, y2);
  x2 = x - b;
  y2 = y1 + a;
  graph.DrawLine(objPenLine, x, y1 - d, x2, y2 - d);
  x2 = x + b;
  graph.DrawLine(objPenLine, x, y1 - d, x2, y2 - d);
 }
}
#endregion

#region Карандаш, шрифт, кисть
//Цвет карандаша
public void vSetPenColorLine(Color pcl)
{
 if (objPenLine == null)
 {
  objPenLine = new Pen(Color.Black, 1);
 }
 objPenLine.Color = pcl;
}
//Установка толщина карандаша
public void vSetPenWidthLine(int penwidth)
{
 if (objPenLine == null)
 {
   objPenLine = new Pen(Color.Black, 1);
 }
 objPenLine.Width = penwidth;
}
#endregion
```

Осталось добавить вызов функции рисования осей:

```
private void vCreateLinGr()
{
  //Создаем массив значений для вывода на графике
  vCreateRg();
  //Создаем класс и передаем ему размер холсты
  PaintCl clPaint = new PaintCl(pictureBox1.Width,
pictureBox1.Height);
  //Фон холста
  clPaint.vSetBackground(Color.White);
  //Параметры вызова: отступы слева, справа, сверху(снизу),
  //Цвет осей, толщина пера, необходимость стрелок
  clPaint.vDravAxis(50, 50, 20, Color.Red, 2,true);
  //Принимаем нарисованное в pictureBox
  pictureBox1.Image = clPaint.Bmp ;
}
```

В функции vDravAxis мы задали параметры непосредственно. Отметим еще раз, что все величины целесообразно иметь настраиваемыми и их значения хранить в реестре.

2.8. Рисоване сетки

Для рисования сетки нам потребуется: цвет и толщина пера, размер массива отображаемых значений и непосредственно функция для рисования сетки. Установку цвета и толщины пера мы уже использовали при рисовании осей, поэтому в функции vCreateLinGr() добавим вновь вызовы:

```
//Параметры линии для сетки
clPaint.vSetPenWidthLine(1);
clPaint.vSetPenColorLine(Color.Silver);
Для хранения размера массива в классе определим переменную и
определим доступ к ней через свойство, а также определим
функцию, рисующую сетку viMaxRg*viMaxRg клеток. Рисование сетки
сводится к рисованию параллельных осям линий:

//Максимальный размер массива
private int viMaxRg = 20;

public int MaxRg
{
 set { viMaxRg=value; }
}

#region Рисование сетки
public void vDravGrid()
{
 float x = viDeltaaxisL;
 float y = viY - viDeltaaxisH;
 float x1 = viX - viDeltaaxisR;
 float y1 = viDeltaaxisH;
 //Сдвиг линий сетки на один отсчет по Y
 float f = (y - y1) / (float)viMaxRg;
 //Рисуем горизонтальные линии
 for (int i = 1; i < viMaxRg + 1; i++)
 {
  graph.DrawLine(objPenLine, x, y - f * i, x1, y - f * i);
 }
 //Сдвиг линий сетки на один отсчет по X
 f = (x - x1) / (float)(viMaxRg - 1);
 //Рисуем вертикальные линии
 for (int i = 1; i < viMaxRg; i++)
 {
  graph.DrawLine(objPenLine, x - f * i, y, x - f * i, y1);
 }
}
#endregion
```

В функции vCreateLinGr() добавим код и выполним решение:

```
clPaint.MaxRg = 20;
clPaint.vDravGrid();
```

2.8. Рисование линии графика

Зададим цвет и толщину пера. Далее нам понадобятся данные из нашего массива значений непосредственно в классе. Для этого в классе определим массив и доступ к нему:

```
private string[,] rgsValues = null;

public string[,] RgValue
{
 set { rgsValues = value; }
}
```

В классе создадим функцию рисования линий графика. Линии рисуются по соседним точкам массива:

```
#region Рисование линий графика для линейного графика
public void vDrawGraphLines()
{
 string s = string.Empty;
 string s1 = string.Empty;
 string s2 = string.Empty;
 float f = 0;
 float f1 = 0;
 float x1 = 0;
 float x = viDeltaaxisL;
 float y = viY - viDeltaaxisH;
 float x2 = 0;
 float fMax = float.MinValue;
 //Ищем максимальное значение по оси Y
 for (int i = 0; i < viMaxRg; i++)
 {
  s = rgsValues[i, 0];
  if (fMax < float.Parse(s)) fMax = float.Parse(s);
 }
 //Пикселей для рисования по оси x
 float fdeltax = viX - viDeltaaxisL - viDeltaaxisR;
 //Пикселей на одну единицу массива значения по X
 fdeltax = fdeltax / (float)(viMaxRg - 1);
 //Пикселей для рисования по оси y
 float fdeltay = viY - 2 * viDeltaaxisH;
 //Пикселей на одну единицу массива значений по Y
 fdeltay = fdeltay / fMax;
 for (int i = 0; i < viMaxRg; i++)
 {
  //Первый раз запоминаем точку старта
  if (i == 0)
  {
    s = rgsValues[i, 0];
```

```
        s2 = rgsValues[i, 1];
        f = y - (float.Parse(s) * fdeltay);
        x1 = x;
      }
      else
      {
        //Здесь рисуем линии
        s1 = rgsValues[i, 0];
        f1 = y - (float.Parse(s1) * fdeltay);
        x2 = x + (int)(fdeltax * i);
        graph.DrawLine(objPenLine, x1, f, x2, f1);
        //Запоминаем координаты конечной точки, точки
        //начала следующего отрезка линии
        s = rgsValues[i, 0];
        s2 = rgsValues[i, 1];
        f = f1;
        x1 = x + (int)(i * fdeltax);
      }
    }
  }
  #endregion
```

Код vCreateLinGr() на данный момент:

```
private void vCreateLinGr()
{
 //Создаем массив значений для вывода на графике
 vCreateRg();
 //Создаем класс и передаем ему размер холста
 PaintCl clPaint = new PaintCl(pictureBox1.Width,
pictureBox1.Height);
 //Фон холста
 clPaint.vSetBackground(Color.White);
 //Параметры вызоыва: отступы слева, справа,
 //сверху(снизу),Цвет осей, толщина пера
 clPaint.vDravAxis(50, 50, 30, Color.Red, 2,true);
 //Цвет и толщина пера
 clPaint.vSetPenWidthLine(1);
 clPaint.vSetPenColorLine(Color.Silver);
 clPaint.MaxRg = 20;
 //Рисуем сетку
 clPaint.vDravGrid();
 //Цвет и толщина пера
 clPaint.vSetPenWidthLine(2);
 clPaint.vSetPenColorLine(Color.Green);
 //Передаем массив значений в класс
 clPaint.RgValue = rgsValues;
 //Рисуем линии графика
 clPaint.vDrawGraphLines();
 //Принимаем нарисованное в pictureBox
 pictureBox1.Image = clPaint.Bmp;
}
```

2.10. Надписи на графике

Надписи можно наносить по оси X, по оси Y и над точками линий графика. Причем иногда бывает целесообразно выполнять соседние надписи со сдвигом по оси Y. Кроме того - надписи выполняются не пером, а кистями и требуют задания шрифта. Таким образом, перед выполнением надписей надо установить в классе соответственно шрифт и кисть (Brush).

Для передачи шрифта и кисти создадим в классе свойства:

```
public Brush  brush
{
 set { objBrush  = value; }
}
public Font font
{
 set { objFont = value; }
}
```

В функции рисования графика запишем код:

```
Font objFont = new Font("Arial", 12, FontStyle.Bold |
FontStyle.Italic);
clPaint.font = objFont;
clPaint.brush = Brushes.Blue;
```

Для выполнения различных надписей создадим в классе несколько функций. Подробно давать пояснения нет необходимости. Здесь, как и при рисовании линий, необходимо постоянно рассчитывать дельны расстояний по осям и точки начала надписей.

```
#region Текст по оси X - Цифры отсчетов
//Параметр: false - соседние значения без сдвига по оси Y
//          true  - соседние значения со эдвигом по оси Y
public void vDrawTextAxXNumber(bool f)
{
 //Пикселей для надписей по оси x
 float fdeltax = viX - viDeltaaxisL - viDeltaaxisR;
 //Пикселей на один отсчет
 fdeltax = fdeltax / (float)(viMaxRg - 1);
 float x = viDeltaaxisL;
 float y = viY - viDeltaaxisH + objPenLine.Width;
 for (int i = 1; i < viMaxRg + 1; i++)
 {
  if (!f || i % 2 == 0)
  {
   graph.DrawString(Convert.ToString(i), objFont, objBrush, x +
(i - 1) * fdeltax, y);
  }
  else
  {
   graph.DrawString(Convert.ToString(i),
```

```
    objFont, objBrush, x + (i - 1) * fdeltax, y + objFont.Size);
   }
  }
 }
#endregion

#region Текст по оси X -  Параметр массива
//Параметр: false - соседние значения без сдвига по оси Y
//          true  - соседние значения со вдвигом по оси Y
public void vDrawTextAxXValues(bool f)
{
 string s = string.Empty;
 //Пикселей для надписей по оси x
 float fdeltax = viX - viDeltaaxisL - viDeltaaxisR;
 //Пикселей на один отсчет
 fdeltax = fdeltax / (float)(viMaxRg - 1);
 float x = viDeltaaxisL;
 float y = viY - viDeltaaxisH;// +objPenLine.Width;
 for (int i = 0; i < viMaxRg; i++)
 {
  if (!f || i % 2 == 0)
  {
   graph.DrawString(rgsValues[i, 1], objFont, objBrush, x + i *
fdeltax, y);
  }
  else
  {
   graph.DrawString(rgsValues[i, 1], objFont, objBrush, x + i *
fdeltax, y + objFont.Size);
  }
 }
}
#endregion

#region Текст по оси Y - Значения по отсчетам сетки оси Y
public void vDrawTextAxYValues()
{
 string s = string.Empty;
 float f = 0;
 float fMax = float.MinValue;
 for (int i = 0; i < viMaxRg; i++)
 {
     s = rgsValues[i, 0];
     if (fMax < float.Parse(s)) fMax = float.Parse(s);
 }
 f = fMax / (float)(viMaxRg - 1);
 //Пикселей для надписей по оси x
 float fdeltay = viY - 2 * viDeltaaxisH;
 //Пикселей на один отсчет
 fdeltay = fdeltay / (float)(viMaxRg - 1);
 float y = viY - viDeltaaxisH - objFont.Size;
 for (int i = 0; i < viMaxRg; i++)
 {
  graph.DrawString(((float)(i * f)).ToString("0.00"),
```

```
     objFont, objBrush, viDeltaaxisL - (objFont.Size) * 5 - 5, y
- i * fdeltay);
 }
}
#endregion

#region Надписи - Значения  над точкой
//1 параметр = false - без отображения процентов, true - с
отображением
//2 параметр = false - без сдвига, true - со здвигом по оси Y
public void vDrawTextAxYValuesPoint(bool a, bool b)
{
 string s = string.Empty;
 float fMax = float.MinValue;
 float fSum = 0;
 for (int i = 0; i < viMaxRg; i++)
 {
     s = rgsValues[i, 0];
     fSum += float.Parse(s);
     if (fMax < float.Parse(s)) fMax = float.Parse(s);
 }
 //Пикселей для надписей по оси x
 float fdeltax = viX - viDeltaaxisL - viDeltaaxisR;
 //Пикселей на один отсчет по x
 fdeltax = fdeltax / (float)(viMaxRg - 1);
 float x = viDeltaaxisL;
 float fdeltay = viY - 2 * viDeltaaxisH;
 float y = viY - viDeltaaxisH - objFont.Size;
 //Пикселей на одну единицу
 fdeltay = fdeltay / fMax;
 float fdelta = 0;
 for (int i = 0; i < viMaxRg; i++)
 {
   if (a)
   {
    if (i % 2 == 0) fdelta = objFont.Size;
      else fdelta = 2 * objFont.Size;
   }
   else
   {
     fdelta = objFont.Size;
   }
   if (b)
   {
    graph.DrawString(rgsValues[i, 0], objFont, objBrush, x + i *
fdeltax,
           y - (float.Parse(rgsValues[i, 0]) * fdeltay) -
fdelta);
   }
   else
   {
    float fp = float.Parse(rgsValues[i, 0]);
    fp = (fp * 100) / fSum;
```

```
    graph.DrawString(rgsValues[i, 0] + "-" + fp.ToString("0.0")
+ "%",
                     objFont, objBrush, x + i * fdeltax,
                                y - (float.Parse(rgsValues[i, 0])
* fdeltay) - fdelta);
    }
  }
}
#endregion
```

Мы создали полностью код для отображения линейного графика. Все функции для управления построением и изменения внешнего вида представлены в void vCreateLinGr():

```
private void vCreateLinGr()
{
 //Создаем массив значений для вывода на графике
 vCreateRg();
 //Создаем класс и передаем ему размер холсты
 PaintCl clPaint = new PaintCl(pictureBox1.Width,
pictureBox1.Height);
 //Фон холста
 clPaint.vSetBackground(Color.White);
 //Параметры вызова: отступы слева, справа, сверху(снизу),Цвет
осей, толщина пера
 clPaint.vDravAxis(50, 50, 30, Color.Red, 2,true);
 clPaint.vSetPenWidthLine(1);
 clPaint.vSetPenColorLine(Color.Silver);
 clPaint.MaxRg = 20;
 clPaint.vDravGrid();
 clPaint.vSetPenWidthLine(2);
 clPaint.vSetPenColorLine(Color.Green);
 clPaint.RgValue = rgsValues;
 clPaint.vDrawGraphLines();
 Font objFont = new Font("Arial", 7, FontStyle.Bold |
FontStyle.Italic);
 clPaint.font = objFont;
 clPaint.brush = Brushes.Blue;
 //Здесь необходимо поэксперементировать с
 //использованием различных надписей и изменением параметров
 clPaint.vDrawTextAxXNumber(false);
 //clPaint.vDrawTextAxXValues(true);
 clPaint.vDrawTextAxYValues();
 clPaint.vDrawTextAxYValuesPoint(true,false);
 //Принимаем нарисованное в pictureBox
 pictureBox1.Image = clPaint.Bmp;
}
```

Лабораторная работа № 15 (2 часа)
Разработка класса

Библиотека визуальных компонентов VCL и ее базовые классы

В основе всего многообразия классов и компонентов, используемых в Visual Studio, лежат всего лишь пять базовых классов (рис.). Они обеспечивают выполнение основных функций любого объекта — будь это стандартный компонент VCL или специализированный объект, выполняющий некоторые операции в приложении.

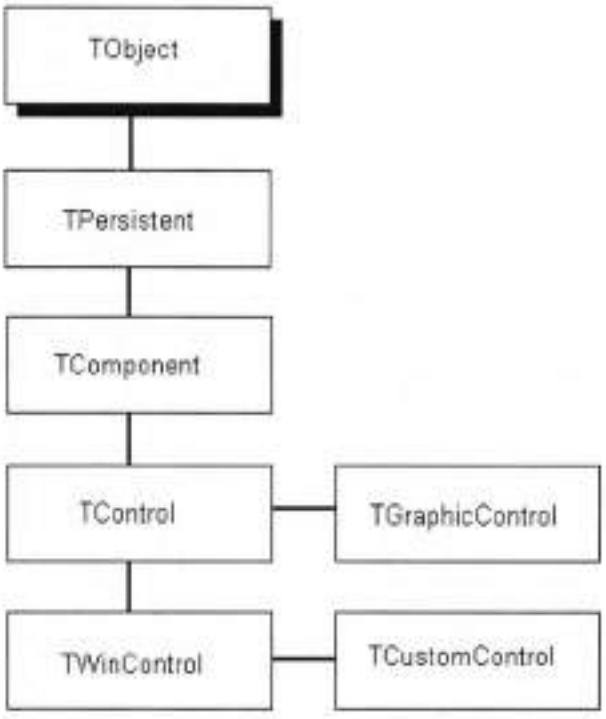


Рис.. Иерархия базовых классов VCL

Благодаря механизму наследования свойств и методов, потомки базовых классов умеют "общаться" друг с другом; работают в среде разработки, взаимодействуя с Палитрой компонентов и Инспектором объектов; распознаются операционной системой как элементы управления и окна.

В основе иерархии классов лежит класс TObject. Он обеспечивает выполнение важнейших функций "жизнедеятельности" любого объекта. Благодаря ему, каждый класс получает в наследство механизмы создания экземпляра объекта и его уничтожения.

Обычно разработчик даже не задумывается о том, как объект будет создан и что необходимо сделать для его корректного уничтожения. Компоненты VCL создаются и освобождают занимаемые ресурсы автоматически. Иногда разработчику приходится создавать и удалять объекты самостоятельно. Но даже в этом случае ему достаточно вызвать соответствующие конструктор и деструктор:

```
var SomeList: TStrings;
...
 SomeList := TStrings.Create;
...
SomeList.Free;
```

За кажущейся простотой этих операций скрывается довольно сложная реализация указанных процессов. Практически весь исходный код класса TObject написан на ассемблере для обеспечения наибольшей эффективности операций, которые будут выполняться в каждом его потомке.

Кроме этого, класс TObject обеспечивает создание и хранение информации об экземпляре объекта и обслуживание очереди сообщений.

Класс TPersistent происходит непосредственно от класса TObject. Он обеспечивает своих потомков возможностью взаимодействовать с другими объектами и процессами на уровне данных. Его методы позволяют передавать данные в потоки, а также обеспечивают взаимодействие объекта с Инспектором объектов.

Класс TComponent является важнейшим для всех компонентов. Непосредственно от него можно создавать любые невизуальные компоненты. Механизмы, реализованные в классе TComponent, обеспечивают взаимодействие компонента со средой разработки, главным образом с Палитрой компонентов и Инспектором объектов. Благодаря возможностям этого класса, компоненты начинают работать на форме проекта уже на этапе разработки.

Класс TControl происходит от класса TComponent. Его основное назначение — обеспечить функционирование визуальных компонентов. Каждый визуальный компонент, произошедший от TControl, наделяется основными признаками элемента управления. Благодаря этому, каждый визуальный компонент умеет работать с GUI (Graphic User Interface — графический интерфейс пользователя ОС) и отображать себя на экране.

Класс TWinControl расширяет возможности разработчиков по созданию элементов управления. Он наследуется от класса TControl и обеспечивает создание оконных элементов управления.

На основе класса TWinControl создан еще один дополнительный класс — TCustomControl. Он обеспечивает создаваемые на его основе компоненты возможностями по использованию канвы — специального объекта, предназначенного для отображения графики. Класс TCustomControl является общим предком для целой группы классов, обеспечивающих создание различных нестандартных типов оконных (получающих фокус) элементов управления Windows: редакторов, списков и т. д.

Для создания неоконных (не получающих фокус) элементов управления используется класс TGraphicControl, являющийся потомком класса TControl.
В целом иерархия базовых классов обеспечивает полноценную работу разработчиков в Visual Studio, позволяя проектировать любые типы приложений.
Ниже мы остановимся на основных свойствах и методах базовых классов, выделяя только те, которые могут пригодиться в реальной работе. Часть из них доступна в Инспекторе объектов, часть может быть использована в программном коде.
Класс TObject
Класс TObject является родоначальником всей иерархии использующихся в Visual Studio классов VCL. Он реализует функции, которые обязательно будет выполнять любой объект, который может быть создан в среде разработки. Учитывая гигантское разнообразие возможных областей применения объектов в процессе создания приложений, можно сказать, что круг общих для всех классов операций весьма невелик.
В первую очередь — это создание экземпляра объекта и его уничтожение. Любой объект выполняет эти две операции в обязательном порядке. Процесс создания объекта включает выделение области адресного пространства, установку указателя на экземпляр объекта, задание начальных значений свойств и выполнение установочных действий, связанных с назначением объекта. В общем случае две последние операции могут не выполняться.
Указатель на экземпляр объекта передается в переменную объектного типа, которая в дальнейшем будет идентифицировать объект в программном коде приложения. В приведенном выше фрагменте кода переменная объектного типа someList объявлена как экземпляр типа TStrings. При создании экземпляра этого типа конструктор Create возвращает в переменную SomeList указатель на выделенную для нового объекта область памяти. Для этого применяется метод Newinstance, который вызывается в конструкторе автоматически:

```
class function Newinstance: TObject; virtual;
```

Объект класса TObject обеспечивает выполнение этого процесса для любого порожденного от него объекта. А уже внутри конструктора, который унаследован от класса TObject, можно предусмотреть инициализацию переменных и выполнение дополнительных операций. Объявление конструктора выглядит следующим образом:

```
constructor Create;
```

В конструкторах потомков это объявление может перекрываться, но при необходимости вызвать конструктор предка используется оператор inherited:

```
constructor TSomeObject.Create;
begin
inherited Create;
...
end;
```

Для уничтожения экземпляра объекта в классе TObject предназначены методы Destroy и Free:

```
destructor Destroy; virtual;
procedure Free;
```

Как видно из объявления, настоящим деструктором является метод Destroy. Он обеспечивает освобождение всех занимаемых экземпляром объекта ресурсов. Обычно при создании новых классов деструктор всегда перекрывается для того, чтобы корректно завершить работу с данными.

Обратите внимание, что обычно унаследованный конструктор вызывается в первую очередь. Это необходимо для того, чтобы выполнить все нужные операции при создании объекта до инициализации его собственных свойств. При уничтожении объекта обычно сначала выполняются завершающие операции и только в самом конце вызывается унаследованный деструктор, чтобы выполнить собственно уничтожение объекта.

При уничтожении объектов рекомендуется вместо деструктора использовать метод Free, который просто вызывает деструктор, но перед этим проверяет, чтобы указатель на экземпляр объекта был не пустым (не был равен Nil). Это позволяет избежать серьезных ошибок.

Если объект является владельцем других объектов (например, форма владеет всеми размещенными на ней компонентами), то его метод Free автоматически вызовет эти же методы для всех объектов. Поэтому при закрытии формы разработчик избавлен от необходимости заботиться об уничтожении всех компонентов.

Для освобождения занимаемой объектом памяти деструктор автоматически Вызывает метод FreeInstance:

```
procedure FreeInstance; virtual;
```

Каждый объект должен содержать некоторую информацию о себе, которая используется приложением и средой разработки. Поэтому класс TObject

содержит ряд методов, обеспечивающих представление этой информации в потомках.
Метод

class function ClassInfo: Pointer;
возвращает указатель на таблицу информации времени выполнения (RTTI). Эта информация используется в среде разработки и в приложении.
Функция

class function ClassName: ShortString;
возвращает имя типа объекта, которое может быть использовано для идентификации. Например, один метод-обработчик щелчка на кнопке может работать с несколькими типами компонентов кнопок:

```
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
if Sender is TBitBtn
then TBitBtn(Sender).Enabled := False;

if Sender is TSpeedButton
then TSpeedButton(Sender).Down := True;
end;
```

Метод

class function ClassNameIs(const Name: string): Boolean;
позволяет определить, является ли данный объект того типа, имя которого передано в параметре Name. В случае положительного ответа функция возвращает True.
Как известно, программирование для Windows основано на событиях. Каждое приложение и каждый программный объект должны уметь реагировать на сообщение о событиях и, в свою очередь, рассылать сообщения. В выполнении этих операций заключается третья общая для всех объектов функция.
Метод

procedure Dispatch(var Message); virtual;
осуществляет обработку сообщений, поступающих объекту. Он определяет, сможет ли объект обработать сообщение при помощи собственных

обработчиков событий. В случае отсутствия таких методов сообщение передается аналогичному методу Dispatch класса-предка (если он есть).
Класс TObject имеет предопределенный обработчик событий:

```
procedure DefaultHandler(var Message); virtual;
```

Кроме рассмотренных здесь методов, класс TObject имеет еще несколько методов, которые в основном применяются для взаимодействия объекта со средой разработки.
В целом класс TObject может служить для создания на его основе некоторых простых классов для использования в приложениях.

Содержание работы.

1. Ознакомиться с теоретической частью методических указаний.
2. Разработать класс, позволяющий хранить и обрабатывать информацию об учебных заведениях СПО города (добавление информации, сортировку по различным критериям, поиск по различным критериям, вывод информации о реализуемых специальностях, выбор всех учебных заведений, реализующих конкретную специальность). Интерфейс класса должен включать перегруженные операторы.
3. Решить поставленную задачу
4. Представить работающую программу преподавателю.

Лабораторная работа № 16 (2 часа)

Разработка программного продукта с использованием ООП.

Объектно-ориентированная программа должна быть представлена в виде взаимосвязанных объектов, которые взаимодействуют между собой. Описанием объекта является класс, в то время как объект - экземпляр этого класса. Класс определяется с помощью ключевого слова Class:

Class Book

End Class

Всю функциональность класса обеспечивают его члены - поля, свойства, методы, конструкторы, события. Поля представляют обычные переменные? обычным образом также определяются процедуры и функции (в этом плане классы во многом похожи на структуры):

```
Class Book
    'Название книги
    Dim name As String
    'Автор
    Dim author As String
    'Год издания
    Dim year As Integer

  'Метод для вывода информации о книге

    Sub GetInformation()
        Console.WriteLine("Книга '{0}' (автор {1}) была
издана в {2} году", name, author, year)
    End Sub
End Class
```

Кроме обычных методов в классах существуют специальные методы - конструкторы. Конструкторы вызываются при создании нового объекта класса. Чтобы объявить конструктор, надо использовать ключевое слово New. Зачем нужен конструктор? Обычно конструктор выполняет инициализацию членов класса. Объявим в классе конструктор, который будет инициализировать поля нашего класса Book:

```
Public Class Book
```

```
    'Название книги
    Dim name As String
    'Автор
    Dim author As String
    'Год издания
    Dim year As Integer

    Sub New(name As String, author As String, year As
Integer)
        Me.name = name
        Me.author = author
        Me.year = year
    End Sub

End Class
```

Здесь мы создали конструктор, который принимает три параметра - название книги, ее автора и год издания. Затем в конструкторе мы присваиваем значения параметров полям класса. Поскольку параметры и поля класса называются одинаково, то нам надо использовать ключевое слово Me, которое предоставляет ссылку на данный класс. Если бы параметры имели другие имена, то использование слова Me было бы необязательным. Если мы не создадим свой конструктор, тогда будет использоваться конструктор по умолчанию:

```
Sub New()


End Sub
```

Теперь используем класс. Создайте новое консольное приложение. Затем нажмите правой кнопкой мыши на название проекта в окне Solution Explorer (Обозреватель решений) и в появившемся меню выберите пункт Add (Добавить), в другом появившемся меню выберите пункт Class (Класс). В окне создания нового класса присвойте ему имя Book и нажмите кнопку Add (Добавить). В проект будет добавлен новый класс Book, который будет находиться в файле Book.vb. Перенесите в этот класс следующий код:

```
Public Class Book
    'Название книги
    Dim name As String
    'Автор
```

```
    Dim author As String
    'Год издания
    Dim year As Integer

    Sub New(name As String, author As String, year As
Integer)
        Me.name = name
        Me.author = author
        Me.year = year
    End Sub
    'Конструктор по умолчанию
    Sub New()
        name = "Евгений Онегин"
        author = "А. С. Пушкин"
        year = 1833
    End Sub
```

'Метод для вывода информации о книге

```
    Sub GetInformation()
        Console.WriteLine("Книга '{0}' (автор {1}) была
издана в {2} году", name, author, year)
    End Sub
End Class
```

Теперь в основной файл приложения - в модуль добавим следующий код:

```
Module Module1

    Sub Main()

        Dim b1 As Book = New Book("Война и мир", "Л. Н.
Толстой", 1869)
        b1.GetInformation()

        'Используем конструктор по умолчанию
        Dim b2 As Book = New Book()
        b2.GetInformation()
```

```
        Console.ReadLine()
    End Sub
End Module
```

Если мы запустим код на выполнение, то консоль выведет нам информацию о книгах b1 и b2. Обратите внимание, что чтобы создать новый объект кроме конструктора нам надо использовать ключевое слово New. В первом случае мы используем свой конструктор, а во втором - конструктор по умолчанию.

Частичные классы

Частичные классы представляют возможность разделения одного класса на несколько файлов. Например, в одном файле может быть:

```
Partial Public Class Book
    'Название книги
    Dim name As String

End Class
```

а в другом:

```
Partial Public Class Book

    'Автор
    Dim author As String
    'Год издания
    Dim year As Integer

End Class
```

Для создания частичного класса перед его объявлением надо поставить ключевое слово Partial. В итоге в результате компиляции будет создан единый класс, который не будет отличаться от других классов.

Ключевое слово With

Предположим, что у нас есть следующий класс State:

```
Class State
    Public capital As String
    Public area As Integer
    Public population As Integer
```

```
    Sub New(cap As String)
        capital = cap
    End Sub
End Class
```

При создании объекта класса мы указываем столицу, значения для остальных полей мы должны указать отдельно, например так:

```
Dim state1 As New State("City")
state1.area = 200
state1.population = 200
```

Однако таких свойств может быть множество. И чтобы сократить объем кода, мы можем использовать конструкцию With...End With. Чтобы применить эту конструкцию, после слова With указывается объект, а затем построчно свойства объекта, которые мы хотим получить или присвоить:

```
Dim capit As String
Dim state1 As New State("City")

With state1
    'Установка остальных полей
    .area = 200
    .population = 200
    'Получение значение поля capital
    capit = .capital
End With
```

Кроме того, мы можем использовать сокращенный синтаксис инициализации объекта с помощью ключевого слова With и одновременно указать все необходимые свойства:

```
Dim state2 = New With {.capital = "City2", .area = 300,
.population = 300}
```

Лабораторная работа № 17(2 часа)

Особенности отладки программного продукта с использованием ООП

Методы и средства отладки

Контроль программы

Под этап контроля программы характеризуется как этап решения задачи, целью которого является установление наличия ошибок в составленной программе или убедительная демонстрация их отсутствия. Если будет установлено, что ошибки в программе имеются, то на следующем этапе (этап локализации) будет производиться их поиск. Поэтому в задачу контроля входит также получение еще и такой информации о характере работы программы, которая могла бы помочь в дальнейшем при поиске ошибок.

Контроль текста

Контроль текста программы можно производить как "вручную", так и с применением ЭВМ. Сначала рассмотрим "ручные" методы контроля текста программ (алгоритмов). Можно различать три способов контроля текста без применения ЭВМ: просмотр, проверка и прокрутка.

Просмотр

Текст составленной программы внимательно просматривается на предмет обнаружения ошибок, описок и смысловых расхождений с текстом алгоритма, по которому производилось программирование. Помимо сплошного просмотра применяется еще и выборочный просмотр некоторых фрагментов программы.

Проверка

При проверке программы программист по тексту программы мысленно старается восстановить тот вычислительный процесс, который определяет программа, после чего сверяет его с требуемым процессом, т. е. ТЗ, определенном в проекте.

Прокрутка

Другим способом контроля программ и алгоритмов за столом является прокрутка (иногда ее называют "сухой" прокруткой – ***dry running*** - для отличия от метода прокрутки, применяемого на этапе локализации и использующего ЭВМ). Основой прокрутки является имитация программистом выполнения программы на машине, с целью более конкретного и наглядного представления о процессе, определяемом текстом проверяемой программы. Прокрутка дает возможность приблизить последовательность проверки программы к последовательности ее выполнения, что позволяет проверять программу как бы в динамике ее работы, проверять элементы вычислительного процесса, задаваемого проверяемой программой а не только статичный текст программы. Для выполнения прокрутки обычно приходится задаваться какими-то исходными данными и производить над ними необходимые вычисления.

Печать текста

Для проверки правильности препарации программы (переноса текста на какие-либо машинные носители) после ввода в машину производится печать текста программы для последующей его сверки с исходным текстом.

Контроль результатов

Как бы ни была тщательно проверена и прокручена программа за столом, решающим этапом, устанавливающим ее пригодность для работы, является контроль программы по результатам ее выполнения на ЭВМ. Наиболее универсальным методом проверки для всех классов задач является метод контрольных тестов или тестирование. Тестом будем называть информацию, состоящую из исходных данных, специально подобранных для отлаженной программы, и из соответствующих им эталонных результатов (не только окончательных, но и промежуточных), используемых в дальнейшем для контроля правильности работы программы.

Тестирование

Под тестированием следует понимать процесс исполнения программы с целью обнаружения ошибок, в качестве которых принимается любое отклонение от эталонов. Хорошим считается тест, который имеет высокую вероятность обнаружения еще не выявленных ошибок.

Под отладкой понимается процесс, позволяющий получить программу, функционирующую с требуемыми характеристиками в заданной области входных данных. Таким образом, в результате отладки программа должна соответствовать некоторой фиксированной совокупности правил и показателей качества, принимаемой за эталонную для данной программы.

Существует 3 основных способа тестирования: алгоритмическое, аналитическое, содержательное.

Алгоритмическое тестирование

Алгоритмическое тестирование применяется программистом для контроля этапов алгоритмизации и программирования. Программисты проектируют тесты и начинают готовить эталонные результаты на этапе алгоритмизации, а используют их на этапе отладки.

Функциональное или аналитическое тестирование

Аналитическое тестирование служит для контроля выбранного метода решения задачи, правильности его работы в выбранных режимах и с установленными диапазонами данных. Тесты проектируют и начинают готовить сразу после выбора метода, а используют их на последнем этапе отладки или для анализа результатов пробного счета; в ходе тестирования, наряду со сверкой на совпадение, применяются и качественные оценки результатов.

Содержательное тестирование

Содержательное тестирование служит для проверки правильности постановки задачи. Для контроля при этом используются, как правило, качественные оценки и статистические характеристики программы, физический смысл полученных результатов и т.п. в проведении содержательного тестирования, принципы которого формулируются в

техническом задании, самое активное участие должны принимать заказчики или идущие пользователи программы.

Содержательные и аналитические тесты проверяют правильность работы программы в целом или крупных ее частей, в то время как алгоритмические тесты в первую очередь должны проверять работу отдельных блоков или операторов программы.

Классификация методов контроля

КОНТРОЛЬ

1. *По тексту.*
    1. Без ЭВМ.
        1. Просмотр.
        2. Проверка.
        3. Прокрутка.
    2. С ЭВМ.
        1. Печать.
        2. Трансляция (синтаксический контроль).
        3. Статический анализ.

2. По результатам.

1. Тестирование.
    1. Алгоритмическое.
    2. Функциональное.
    3. Содержательное.
2. Специальные методы.

3.3.4. Локализация ошибок.

Способы локализации

После того, как с помощью контрольных тестов (или каким либо другим путем) установлено, что в программе или в конкретном ее блоке имеется ошибка, возникает задача ее локализации, то есть установления точного места в программе, где находится ошибка.

Процесс локализации ошибок состоит из следующих трех компонент:

1. *Получение на машине тестовых результатов.*
2. *. Анализ тестовых результатов и сверка их с эталонными.*
3. *. Выявление ошибки или формулировка предположения о характере и месте ошибки в программе.*

По принципам работы средства локализации разделяются на 4типа:

1. *Аварийная печать.*
2. *Печать в узлах.*
3. *Слежение.*
4. *Прокрутка.*

АВАРИЙНАЯ ПЕЧАТЬ осуществляется один раз при работе отлаживаемой программы, в момент возникновения аварийной ситуации в программе, препятствующей ее нормальному выполнению. Тем самым, конкретное место включения в работу аварийной печати определяется автоматически без использования информации от программиста, который должен только определить список выдаваемых на печать переменных.

ПЕЧАТЬ В УЗЛАХ включается в работу в выбранных программистом местах программы; после осуществления печати значений данных переменных продолжается выполнение отлаживаемой программы.

СЛЕЖЕНИЕ производится или по всей программе, или на заданном про грамм истом участке. Причем слежение может осуществляться как за переменными (арифметическое слежение), так и за операторами (логическое слежение). Если обнаруживается, что происходит присваивание заданной переменной или выполнение оператора с заданной меткой, то производится печать имени переменной или метки и выполнение программы продолжается. Отличием от печати в узлах является то, что место печати может точно и не определяться программистом (для арифметического слежения); отличается также и содержание печати.

ПРОКРУТКА производится на заданных участках программы, и после выполнения каждого оператора заданного типа (например, присваивания или помеченного) происходит отладочная печать.

Классификация средств локализации ошибок

Ниже дана классификация средств локализации.

Средства локализации:

1. *Аварийная печать (арифметическая).*
    1. Специальные средства языка.
    2. Системные средства.
2. *Печать в узлах (арифметическая).*
    1. Обычные средства языка.
    2. Специальные средства языка.
3. *Слежение (специальные средства).*
    1. Арифметическое.
    2. Логическое.
4. Прокрутка (специальные средства).
    1. Арифметическая.
    2. Логическая.

3.3.5. Технология отладки программы

Рассмотрим этапы создания рассматриваемой программы, основываясь на приведенных выше методах и приемах.

Обычно рано или поздно наступает такой момент, когда возникает вопрос о необходимости создания программы. Перед нами стояла именно такая задача. То есть нам необходимо было, создать программу, рассчитанную на не специалистов в области компьютерной техники, т. е. программу с отлично проработанным интерфейсом для удобства и доступности пользователем. ТЗ выдавалось постепенно, уточнялось и изменялось. В зависимости от этого программа разрабатывалась поэтапно. Алгоритмизация в нашем случае охватывает, как программу целиком, так и поблочно, для процедур и функций.

При отладке программы использовались следующие методы контроля и локализации ошибок (обзор методов см. в теоретической части раздела):

1. Просмотр текста программы и прокрутка с целью обнаружения явных синтаксических и логических ошибок.
2. Трансляция программы (транслятор выдает сообщения об обнаруженных им ошибках в тексте программы).
3. При локализации ошибок преимущественно использовалась печать в узлах, которыми являлись в основном глобальные переменные, переменные, используемые при обмене данными основной про граммы с подпрограммами слежения. Также сверялись типы получаемых данных, что позволяет выявить ошибки несовпадения типов, которые не видны при компиляции.

Известно, что существует проблема восприятия информации с экрана монитора. Чтобы избежать этого мы использовали распечатки программы на различных этапах разработки и отладки.

В общем случае отладка программы производилась по следующему алгоритму:

1. Прогонка программы с набором тестовых входных данных и наличия ошибок.
2. Выделение области программы, в которой может находиться ошибка. Просмотр листинга программы с целью возможного обнаружения ошибок. В противном случае - установка контрольной точки примерно в середине выделенной области.
3. Новая прогонка программы. Если работа программы прервалась до обработки контрольной точки, значит, ошибка произошла раньше. Контрольная точка переносится, и процесс отладки возвращается к шагу 2.
4. Если контрольная точка программы была обработана, то далее следует изучение значений регистров, переменных и параметров программы с тем, чтобы убедиться в их правильности. При появлении ошибки - новый перенос контрольной точки и возврат к шагу 2
5. В случае не обнаружения ошибки продолжение выполнения программы покомандно, с контролем правильности выполнения переходов и содержимого регистров и памяти в контрольных точках. При локализации ошибки она исправляется, и процесс возвращается к шагу 1.

Решающим этапом, устанавливающим пригодность программы для работы, является её контроль по результатам ее выполнения на ЭВМ. Наиболее универсальным методом проверки для всех классов задач является метод контрольных тестов или тестирование.

В процессе отладки созданной программы, был выявлен ряд ошибок. На основании результатов анализа проделанной работы по их устранению была разработана инструкция по пользованию созданной Автоматизированной системой и предложены методы устранения возможных ошибок.

Содержание работы

1. Ознакомиться с теоретической частью методических указаний.
2. Получить вариан задания от преподавателя.
3. Решить поставленную задачу
4. Представить работающую программу преподавателю.

Лабораторная работа № 18 (2 часа)
Работа с информацией о файле

Работа с файлами в Visual Studio позволяет считывать, сохранять информацию, и выполнять другие действия с файлами. В Visual Studio поддерживаются все операции с файлами - создание, поиск, чтение и запись, переименование как файлов, так и к директорий. В Visual Studio существует несколько способов работы с файлами.

Вначале нужно упомянуть компоненты Visual Studio, которые умеют работать с файлами. Они читают и сохраняют своё содержимое, строки типа String, в файл текстового формата. Это компоненты ListBox, ComboBox и Memo, расположенные на первой же вкладке палитры компонентов.

Каждая строка компонентов ListBox и ComboBox является объектом Items[i], а Memo - Lines[i], где i - номер строки, который отсчитывается от нуля. Добавление строк в компоненты выполняется методами Add и Insert:

```
begin
 Memo1.Lines.Add('Первая строка');
 ComboBox1.Items.Add('Первая строка');
 ComboBox1.Items.Add('Вторая строка');
 ListBox1.Items.Add('Первая строка');
 ListBox1.Items.Add('Вторая строка');
end;
```

Метод Add добавляет новую строку в конец. Метод Insert имеет дополнительный параметр, указывающий, после какой строки разместить новую строку. Доступ к строкам осуществляется так:

```
ComboBox1.Items[0] := 'Первая строка изменилась';
ListBox1.Items[1] := 'Вторая строка изменилась' ;
```

У компонента ComboBox дополнительно есть свойство Text, где (как и у компонента Edit) находится вводимый текст:

```
ComboBox1.Text := ' Вводимый текст ';
```

На выделенную в данный момент строку компонента ComboBox указывает свойство ItemIndex типа Integer, то есть это номер выделенной строки. Следовательно, получить саму выделенную строку компонента ComboBox можно следующей конструкцией:

```
S := ComboBox1.Items[ComboBox1.ItemIndex];
```

или, пользуясь оператором присоединения

```
With ComboBox1 do
S := Items[ItemIndex];
```

Вот как по нажатию клавиши Enter можно заносить в этот компонент вводимую в строку информацию и удалять нажатием Escape:

Выделите на Форме ComboBox и перейдите в Инспектор объектов, на вкладку Events. Щёлкните дважды по обработчику OnKeyPress. Система Visual Studio создаст заготовку обработчика. Напишите:

```
begin
if Key = #13 then
ComboBox1.Items.Add(ComboBox1.Text)
if Key = #27 then;
ComboBox1.Items.Delete(ComboBox1.Items.Count-1);
end;
```

Key - определённая в этом обработчике переменная, содержащая код нажатой клавиши, #13 и #27 - коды клавиш Enter и Escape соответственно. Items.Count - количество содержащихся в компоненте строк. Так как отсчёт строк идёт от нуля, мы отнимаем единицу. После очередного удаления количество строк меняется, таким образом, Items.Count-1 всегда указывает на последнюю строку. Последовательно нажимая Escape, мы можем удалить все строки. Командой

ComboBox1.Items.Delete(0);

можно добиться того же эффекта, только удаляться будут первые строки. Впрочем, чтобы стереть всё сразу, есть метод Clear!

Ну а теперь собственно о сохранении содержимого в файл. Для этого выполните команду

ListBox1.Items.SaveToFile('Имя_файла.txt');

Впрочем, расширение можно поставить любое по желанию, не обязательно .txt, как и вообще без него обойтись. Но расширение .txt позволит легко открыть файл стандартным виндовским Блокнотом, что бывает очень удобно на этапе написания программы!

Для загрузки служит метод LoadFromFile:

ListBox1.Items.LoadFromFile('Имя_файла.txt');

Что делать, если в своей программе мы не работаем с компонентами Combobox, ListBox или Memo, а сохранять информацию нужно? Берём один из компонентов и делаем его невидимым, для этого в Инспекторе Объектов ставим в свойство Visible значение False. Функциональность его от этого не изменится!

Последний штрих - создадим программу, сохраняющую своё положение на экране в момент закрытия и там же открывающуюся. Нам нужно сохранить два параметра - значения свойств Формы Left и Top, это расстояние от левого и верхнего краёв экрана соответственно. Значения эти выражаются в

пикселах и имеют тип Integer. Необходимо перевести эти числа в строку (тип String) с помощью оператора IntToStr. Итак, разместите на Форме невидимый ListBox, щёлкните по пустому месту Формы, чтобы её свйства появились в Инспекторе объектов, и перейдите на вкладку Events. Щёлкните по обработчику OnClose и в созданной заготовке напишите:

```
begin
ListBox1.Items.Clear;
ListBox1.Items.Add(IntToStr(Form1.Left));
ListBox1.Items.Add(IntToStr(Form1.Top));
ListBox1.Items.SaveToFile('MyFormPos.txt');
end;
```

Этот обработчик сохраняет положение Формы на экране. Теперь напишем обработчик, помещающий Формуна прежнее место при старте программы. Создайте заготовку обработчика события OnCreate. Это событие происходит в момент "создания" Формы операционной системой. В этот момент и нужно присваивать ей необходимые свойства. Пишите:

```
begin
if FileExists('MyFormPos.txt') then
begin
ListBox1.Items.LoadFromFile('MyFormPos.txt');
Form1.Left := StrToInt(ListBox1.Items[0]);
Form1.Top := StrToInt(ListBox1.Items[1]);
end;
end;
```

В первой строке происходит проверка на существование файла, ведь если его не будет, произойдёт ошибка. Впрочем, программа после выдачи предупреждения откроется в том месте, где была на этапе проектирования, а при закрытии нужный файл будет воссоздан!

Затем в логических скобках begin/end содержится сам код, который будет выполнен только приналичии файла MyFormPos.txt в папке с программой, так как используется относительный путь. Можно указать конкретное местоположение, например, C:\Program Files\MyProg\MyFormPos.txt.

Проверку на существование файла можно выполнить также с помощью контроля исключительных ситуаций. Если файл не существует, то произойдёт исключительная ситуация. Перехватив её с помощью специального оператора, мы сможем избежать ошибок в программе.

Содержание работы

1. Ознакомиться с теоретической частью методических указаний.
2. Получить вариан задания от преподавателя.
3. Решить поставленную задачу
4. Представить работающую программу преподавателю.

Лабораторная работа № 19 (2 часа)
Использование основных операций с файлами

Технология **работы с файлами** в системе **Visual Studio** требует определённого порядка действий:

1. Прежде всего **файл** должен быть **открыт**. Система следит, чтобы другие приложения не мешали **работе с файлом**. При этом определяется, в каком режиме открывается **файл** - для изменения или только считывания информации. После открытия **файла** в программу возвращается его идентификатор, который будет использоваться для указания на этот **файл** во всех процедурах обработки.
2. Начинается **работа с файлом**. Это могут быть запись, считывание, поиск и другие операции.
3. **Файл** закрывается. Теперь он опять доступен другим приложениям без ограничений. Закрытие **файла** гарантирует, что все внесённые изменения будут сохранены, так как для увеличения скорости работы изменения предварительно сохраняются в специальных буферах операционной системы.

В **Visual Studio** реализовано несколько способов **работы с файлами**. Познакомимся со классическим способом, связанным с использованием **файловых переменных**. Файловая переменная вводится для указания на файл. Делается это с помощью ключевого слова **File** :

```
var F: File ;
```

Описанная таким образом файловая переменная считается ***нетипизированной***, и позволяет работать с файлами с неизвестной структурой. Данные считываются и записываются побайтно блоками, размер которых указывается при открытии файла, вплоть от 1 байт.

Но чаще используются файлы, состоящие из последовательности одинаковых записей. Для описания такого файла к предыдущему описанию добавляется указание типа записи:

```
var F: File of тип_записи ;
```

В качестве типа могут использоваться базовае типы, или создаваться свои. Важно только, чтобы для типа был точно известен фиксированный размер в байтах, поэтому, например, тип **String** в чистом виде применяться не может, а только в виде **String[N]**, как указывалось в уроке Visual Studio 5.

Данные, считанные из файла или записываемые в файл, содержатся в обычной переменной, которая должна быть того же типа, что и файловая. Поэтому сначала в программе лично я описываю нужный тип, а затем ввожу две переменные этого типа - файловую и обычную:

Для открытия файла нужно указать, где он расположен. Для этого файловая переменная должна быть ассоциирована с нужным файлом, который определяется его адресом. Адрес файла может быть абсолютным, с указанием диска и каталогов ('C:\Мои документы\Мои рисунки\FileName.ini'), или относительным, тогда он создаётся в папке с **.exe** файлом программы. Для задания относительного адреса достаточно указать имя файла с нужным расширением. Делается это оператором **AssignFile :**

```
AssignFile(SaveF, 'C:\Мои документы\Мои рисунки\FileName.ini');
AssignFile(SaveF, 'FileName.ini');
```

Теперь файл должен быть открыт.

Открытие файла оператором **Rewrite** приведёт воссозданию файла заново, т.е. существующий файл будет ***без предупреждения*** уничтожен, и на его месте будет создан новый пустой файл заданного типа, готовый к записи данных. Если же файла не было, то он будет создан.

Открытие файла оператором **Reset** откроет существующий файл к считыванию или записи данных, и его указатель будет установлен на начало файла **:**

```
Rewrite(SaveF);
Reset(SaveF);
```

Каждый из этих операторов может иметь второй необязательный параметр, имеющий смысл для нетипизированных файлов, и указывающий длину записи нетипизированного файла в байтах:

```
Rewrite(SaveF, 1);
Reset(SaveF, 1);
```

Чтение файла производится оператором **Read :**

```
Read(SaveF, SaveV);
```

Запись в файл производится оператором **Write :**

```
Write(SaveF, SaveV);
```

При этом чтение и запись производится с текущей позиции указателя, затем указатель устанавливается на следующую запись. Можно проверить, существует ли нужный файл, оператором **FileExists :**

```
if FileExists('FileName.ini')
  then Read(SaveF, SaveV);
```

Принудительно установить указатель на нужную запись можно оператором **Seek(SaveF, N)**, где N - номер нужной записи, который, как и почти всё в программировании, отсчитывается от нуля:

```
Seek(SaveF, 49); - установка указателя на 50-ю запись.
```

При последовательном чтении из файла рано или поздно будет достигнут конец файла, и при дальнейшем чтении произойдёт ошибка. Проверить, не достигнут ли конец файла, можно оператором **EOF** (аббревиатура **E**nd **O**f **F**ile), который равен **true**, если прочитана последняя запись и указатель находится в конце файла:

```
while (not EOF(SaveF)) do
  Read(SaveF, SaveV);
```

Для текстовых файлов вместо **Read** и **Write** используются операторы **Readln** и **Writeln**, умеющие определять конец строки. В коментариях приведена процедура чтения текстового файла.

Оператор **Truncate(SaveF)** позволяет отсечь (стереть или, если хотите, удалить!) все записи файла, начиная от текущей позиции указателя, и до конца файла.

В конце работы с файлом его необходимо закрыть. Это делается оператором **CloseFile(SaveF) ;**

Создаём обработчик события Формы **OnCreate** со следующим содержимым:

```
procedure TForm1.FormCreate(Sender: TObject) ;
begin
  AssignFile(SaveF, 'Init.ini') ;
  if FileExists('Init.ini') then
    begin
      Reset(SaveF) ;
      Read(SaveF, SaveV) ;
      Form1.Left := SaveV.X ;
      Form1.Top := SaveV.Y ;
      Form1.Caption:=SaveV.Caption ;     //Наши переменные дополнительно
сохраняют заголовок Формы!
    end ;
end ;
```

Теперь необходимо создать обработчик события **OnClose :**

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction) ;
begin
  Rewrite(SaveF) ;     //Нет необходимости проверять наличие файла,
создадим его заново!
  SaveV.X := Form1.Left ;
  SaveV.Y := Form1.Top ;
  SaveV.Caption := Form1.Caption ;
  Write(SaveF, SaveV) ;
  CloseFile(SaveF) ;
end ;
```

В данном случае файл считывается и записывается туда, куда мы ему указали. Но необходимо также уметь выбрать нужный файл в работающей программе.

Содержание работы

1. Ознакомиться с теоретической частью методических указаний.
2. Выполнить примеры
3. Получить вариант задания от преподавателя.
4. Решить поставленную задачу
5. Представить работающую программу преподавателю.

Лабораторная работа № 20 (2 часа)
Чтение и запись текстовых файлов

Чтение данных из файла

Чтение из файла выполняется при помощи инструкций **read** и **readln**, которые в общем виде записываются следующим образом:

read(ФайловаяПеременная, СписокПеременных);
readln (ФайловаяЛеременшя, СписокПеременных) ;

где:

- ФайловаяПеременная — переменная типа Textfile;
- списокпеременных — имена переменных, разделенные запятыми.

Чтение чисел

Следует понимать, что в текстовом файле находятся не числа, а их изображения. Действие, выполняемое инструкциями read или readin, фактически состоит из двух: сначала из файла читаются символы до появления разделителя (пробела или конца строки), затем прочитанные символы, являющиеся изображением числа, преобразуются в число, и полученное значение присваивается переменной, имя которой указано в качестве параметра инструкции read ИЛИ readln.

Например, если текстовый файл a:\data.txt содержит следующие строки:

23 15
45 28
56 71

то в результате выполнения инструкций:

значения переменных будут следующими: a = 23, b = 15, c = 45, d = 23.

Отличие инструкции readin от read состоит в том, что после считывания из файла очередного числа и присвоения полученного значения переменной, имя которой стоит последним в списке параметров инструкции readin, указатель чтения из файла автоматически перемешается в начало следующей строки файла, даже в том случае, если за прочитанным числом есть еше числа.

Поэтому в результате выполнения инструкций

значения переменных будут следующими: a = 23, b = 45, c = 28, d = 56.

Если при чтении значения численной переменной в файле вместо изображения числа будет какая-то другая последовательность символов, то произойдет ошибка.

Чтение строк

В программе строковая переменная может быть объявлена с указанием длины или без нее.

Например:

При чтении из файла значения строковой переменной, длина которой явно задана в ее объявлении, считывается столько символов, сколько указано в объявлении, но не больше, чем в текущей строке.

При чтении из файла значения строковой переменной, длина которой явно не задана в объявлении переменной, значением переменной становится оставшаяся после последнего чтения часть текущей строки. Другими словами, если надо прочитать из файла всю строку, то объявите строковую переменную, длина которой заведомо больше самой длинной строки файла, и считывайте строки в ату переменную.

мер, двух переменных, то первая переменная будет содержать столько символов, сколько указано в ее объявлении или, если длина не указана, всю строку файла. Вторая переменная будет содержать оставшиеся символы текущей строки или, если таких символов нет, не будет содержать ни одного символа (длина строки равна нулю).
Пусть, например, текстовый файл freinds.txt содержит строки:

Косичкина Маша
Васильев Антон
Цой Лариса

Конец файла

Пусть на диске есть некоторый текстовый файл. Нужно в диалоговое окно вывести содержимое этого файла. Решение задачи довольно очевидно: надо открыть файл, прочитать первую строку, затем вторую, третью и т. д. до тех пор, пока не будет достигнут конец файла. Но как определить, что прочитана последняя строка, достигнут конец файла? Для определения конца файла можно воспользоваться функцией EOF (End Of File — конец файла).

У функции **EOF** один параметр — файловая переменная. Значение функции EOF равно False, если прочитанный элемент данных не является последним в файле, т. е. возможно дальнейшее чтение. Если прочитанный элемент данных является последним, то значение EOF равно True.
Значение функции EOF можно проверить сразу после открытия файла. Если при этом оно окажется равным True, то это значит, что файл не содержит ни одного элемента данных, т. е. является пустым (размер такого файла равен нулю).

В листинге 5 приведена процедура, которая выполняет поставленную задачу. Она читает строки из файла, имя которого ввел пользователь во время работы программы, и выводит эти строки в поле Memo. Окно программы приведено на рисунке

Листинг. Чтение из файла

Содержание работы

1. Ознакомиться с теоретической частью методических указаний.
2. Выполнить примеры
3. Получить вариант задания от преподавателя.
4. Решить поставленную задачу
5. Представить работающую программу преподавателю.

Лабораторная работа № 21 (2 часа)
Работа с информацией о файле

Большинство компьютерных программ работают с файлами, и поэтому возникает необходимость создавать, удалять, записывать читать, открывать файлы. Что же такое файл? Файл – именованный набор байтов, который может быть сохранен на некотором накопителе. Ну, теперь ясно, что под файлом понимается некоторая последовательность байтов, которая имеет своё, уникальное имя, например файл.txt. В одной директории не могут находиться файлы с одинаковыми именами. Под именем файла понимается не только его название, но и расширение, например: file.txt и file.dat - разные файлы, хоть и имеют одинаковые названия. Существует такое понятие, как полное имя файлов – это полный адрес к директории файла с указанием имени файла, например: D:\docs\file.txt. Важно понимать эти базовые понятия, иначе сложно будет работать с файлами.

Для работы с файлами необходимо подключить заголовочный файл <fstream>. В <fstream> определены несколько классов и подключены заголовочные файлы <ifstream> - файловый ввод и <ofstream> - файловый вывод.

Файловый ввод/вывод аналогичен стандартному вводу/выводу, единственное отличие – это то, что ввод/вывод выполнятся не на экран, а в файл. Если ввод/вывод на стандартные устройства выполняется с помощью объектов cin и cout, то для организации файлового ввода/вывода достаточно создать собственные объекты, которые можно использовать аналогично операторам cin и cout.

Например, необходимо создать текстовый файл и записать в него строку Работа с файлами в C++. Для этого необходимо проделать следующие шаги:

1. создать объект класса ofstream;
2. связать объект класса с файлом, в который будет производиться запись;
3. записать строку в файл;
4. закрыть файл.

Почему необходимо создавать объект класса ofstream, а не класса ifstream? Потому, что нужно сделать запись в файл, а если бы нужно было считать данные из файла, то создавался бы объект класса ifstream.

```
// создаём объект для записи в файл
ofstream /*имя объекта*/; // объект класса ofstream
```

Назовём объект – fout, Вот что получится:

```
ofstream fout;
```

Для чего нам объект? Объект необходим, чтобы можно было выполнять запись в файл. Уже объект создан, но не связан с файлом, в который нужно записать строку.

```
fout.open("cppstudio.txt"); // связываем объект с файлом
```

Через операцию точка получаем доступ к методу класса open(), в круглых скобочках которого указываем имя файла. Указанный файл будет создан в текущей директории с программой. Если файл с таким именем существует, то существующий файл будет заменен новым. Итак, файл открыт, осталось записать в него нужную строку. Делается это так:

```
fout << "Работа с файлами в C++"; // запись строки в файл
```

Используя операцию передачи в поток совместно с объектом fout строка Работа с файлами в C++ записывается в файл. Так как больше нет необходимости изменять содержимое файла, его нужно закрыть, то есть отделить объект от файла.

```
fout.close(); // закрываем файл
```

Итог – создан файл со строкой Работа с файлами в C++.

Шаги 1 и 2 можно объединить, то есть в одной строке создать объект и связать его с файлом. Делается это так:

```
ofstream fout("cppstudio.txt"); // создаём объект класса ofstream и связываем его с файлом cppstudio.txt
```

Объединим весь код и получим следующую программу.

```
// file.cpp: определяет точку входа для консольного приложения.

#include "stdafx.h"
#include <fstream>
using namespace std;

int main(int argc, char* argv[])
```

```
{
   ofstream fout("cppstudio.txt"); // создаём объект класса ofstream для записи и связываем его с файлом cppstudio.txt
   fout << "Работа с файлами в C++"; // запись строки в файл
   fout.close(); // закрываем файл
   system("pause");
   return 0;
}
```

Осталось проверить правильность работы программы, а для этого открываем файл cppstudio.txt и смотрим его содержимое, должно быть - Работа с файлами в C++.

Для того чтобы прочитать файл понадобится выполнить те же шаги, что и при записи в файл с небольшими изменениями:

1. создать объект класса ifstream и связать его с файлом, в который будет производиться запись;
2. прочитать файл;
3. закрыть файл.

```
// file_read.cpp: определяет точку входа для консольного приложения.

#include "stdafx.h"
#include <fstream>
#include <iostream>
using namespace std;

int main(int argc, char* argv[])
{
   setlocale(LC_ALL, "rus"); // корректное отображение Кириллицы
   char buff[50]; // буфер промежуточного хранения считываемого из файла текста
   ifstream fin("cppstudio.txt"); // открыли файл для чтения

   fin >> buff; // считали первое слово из файла
   cout << buff << endl; // напечатали это слово
```

```
    fin.getline(buff, 50); // считали строку из файла
    fin.close(); // закрываем файл
    cout << buff << endl; // напечатали эту строку

    system("pause");
    return 0;
}
```

В программе показаны два способа чтения из файла, первый – используя операцию передачи в поток, второй – используя функцию getline(). В первом случае считывается только первое слово, а во втором случае считывается строка, длиной 50 символов. Но так как в файле осталось меньше 50 символов, то считываются символы включительно до последнего. Обратите внимание на то, что считывание во второй раз (**строка 17**) продолжилось, после первого слова, а не с начала, так как первое слово было прочитано в **строке 14**.

Программа сработала правильно, но не всегда так бывает, даже в том случае, если с кодом всё впорядке. Например, в программу передано имя несуществующего файла или в имени допущена ошибка. Что тогда? В этом случае ничего не произойдёт вообще. Файл не будет найден, а значит и прочитать его не возможно. Поэтому компилятор проигнорирует строки, где выполняется работа с файлом. В результате корректно завершится работа программы, но ничего, на экране показано не будет. Казалось бы это вполне нормальная реакции на такую ситуацию. Но простому пользователю не будет понятно, в чём дело и почему на экране не появилась строка из файла. Так вот, чтобы всё было предельно понятно в C++ предусмотрена такая функция - is_open(), которая возвращает целые значения: 1 — если файл был успешно открыт, 0 — если файл открыт не был. Доработаем программу с открытием файла, таким образом, что если файл не открыт выводилось соответствующее сообщение.

```
// file_read.cpp: определяет точку входа для консольного приложения.

#include "stdafx.h"
#include <fstream>
#include <iostream>
using namespace std;

int main(int argc, char* argv[])
```

```
{
   setlocale(LC_ALL, "rus"); // корректное отображение Кириллицы
   char buff[50]; // буфер промежуточного хранения считываемого из файла текста
   ifstream fin("cppstudio.doc"); // (ВВЕЛИ НЕ КОРРЕКТНОЕ ИМЯ ФАЙЛА)

   if (!fin.is_open()) // если файл не открыт
      cout << "Файл не может быть открыт!\n"; // сообщить об этом
   else
   {
   fin >> buff; // считали первое слово из файла
   cout << buff << endl; // напечатали это слово

   fin.getline(buff, 50); // считали строку из файла
   fin.close(); // закрываем файл
   cout << buff << endl; // напечатали эту строку
   }
   system("pause");
   return 0;
}
```

Программа сообщит о невозможности открыть файл. Поэтому, если программа работает с файлами, рекомендуется всегда использовать эту функцию, is_open(), даже, если уверены, что файл существует.

Режимы открытия *Режимы открытия файлов* файлов устанавливают характер использования файлов. Для установки режима в классе ios_base предусмотрены константы, которые определяют режим открытия файлов (см. Таблица 1).

Таблица 1 – режим открытия файлов

Константа	Описание
ios_base::in	открыть файл для чтения
ios_base::out	открыть файл для записи

Таблица 1 — режимы открытия файлов	
Константа	**Описание**
ios_base::ate	при открытии переместить указатель в конец файла
ios_base::app	открыть файл для записи в конец файла
ios_base::trunc	удалить содержимое файла, если он существует
ios_base::binary	открытие файла в двоичном режиме

Режимы открытия файлов можно устанавливать непосредственно при создании объекта или при вызове функции open().

```
ofstream fout("cppstudio.txt", ios_base::app); // открываем файл для добавления информации к концу файла
fout.open("cppstudio.txt", ios_base::app); // открываем файл для добавления информации к концу файла
```

Режимы открытия файлов можно комбинировать с помощью поразрядной логической операции **или** |, например: ios_base::out | ios_base::trunc - открытие файла для записи, предварительно очистив его.

Объекты класса ofstream, при связке с файлами по умолчанию содержат режимы открытия файлов ios_base::out | ios_base::trunc. То есть файл будет создан, если не существует. Если же файл существует, то его содержимое будет удалено, а сам файл будет готов к записи. Объекты класса ifstream связываясь с файлом, имеют по умолчанию режим открытия файла ios_base::in - файл открыт только для чтения. Режим открытия файла ещё называют — флаг, для удобочитаемости в дальнейшем будем использовать именно этот термин. В таблице 1 перечислены далеко не все флаги, но для начала этих должно хватить.

Обратите внимание на то, что флаги ate и app по описанию очень похожи, они оба перемещают указатель в конец файла, но флаг app позволяет производить запись, только в конец файла, а флаг ate просто переставляет флаг в конец файла и не ограничивает места записи.

Разработаем программу, которая, используя операцию sizeof(), будет вычислять характеристики основных типов данных в C++ и записывать их в файл. Характеристики:

1. число байт, отводимое под тип данных

2. максимальное значение, которое может хранить определённый тип данных.

Запись в файл должна выполняться в таком формате:

```
/*  data type      byte        max value
bool              = 1        255.00
char              = 1        255.00
short int         = 2        32767.00
unsigned short int = 2       65535.00
int               = 4        2147483647.00
unsigned int      = 4        4294967295.00
long int          = 4        2147483647.00
unsigned long int = 4        4294967295.00
float             = 4        2147483647.00
long float        = 8        92233720368547758O0.00
double            = 8        92233720368547758O0.00  */
```

Такая программа уже разрабатывалась ранее в разделе Типы данных C++ , но там вся информация о типах данных выводилась на стандартное устройство вывода, а нам необходимо программу переделать так, чтобы информация записывалась в файл. Для этого необходимо открыть файл в режиме записи, с предварительным усечением текущей информации файла (**строка 14**). Как только файл создан и успешно открыт (строки 16 — 20), вместо оператора cout, в **строке 22** используем объект fout. таким образом, вместо экрана информация о типах данных запишется в файл.

```
// write_file.cpp: определяет точку входа для консольного приложения.

#include "stdafx.h"
#include <iostream>
#include <fstream> // работа с файлами
#include <iomanip> // манипуляторы ввода/вывода
using namespace std;

int main(int argc, char* argv[])
{
   setlocale(LC_ALL, "rus");
```

```
// связываем объект с файлом, при этом файл открываем в режиме записи, предварительно удаляя все данные из него
ofstream fout("data_types.txt", ios_base::out | ios_base::trunc);

if (!fout.is_open()) // если файл небыл открыт
{
cout << "Файл не может быть открыт или создан\n"; // напечатать соответствующее сообщение
return 1; // выполнить выход из программы
}

    fout << "    data type     " << "byte"                << "    "   << "   max value  " << endl // заголовки столбцов
        << "bool          = " << sizeof(bool)          << "       " << fixed << setprecision(2)
/*вычисляем максимальное значение для типа данных bool*/
<< (pow(2,sizeof(bool) * 8.0) - 1)          << endl
      << "char          = " << sizeof(char)          << "       " << fixed << setprecision(2)
/*вычисляем максимальное значение для типа данных char*/
<< (pow(2,sizeof(char) * 8.0) - 1)          << endl
      << "short int     = " << sizeof(short int)     << "       " << fixed << setprecision(2)
/*вычисляем максимальное значение для типа данных short int*/
<< (pow(2,sizeof(short int) * 8.0 - 1) - 1)    << endl
        << "unsigned short int = " << sizeof(unsigned short int) << "       " << fixed << setprecision(2)
/*вычисляем максимальное значение для типа данных unsigned short int*/
<< (pow(2,sizeof(unsigned short int) * 8.0) - 1) << endl
        << "int           = " << sizeof(int)           << "       " << fixed << setprecision(2)
/*вычисляем максимальное значение для типа данных int*/                    << (pow(2,sizeof(int) * 8.0 - 1) - 1)        << endl
        << "unsigned int  = " << sizeof(unsigned int)  << "       " << fixed << setprecision(2)
/*вычисляем максимальное значение для типа данных unsigned int*/
<< (pow(2,sizeof(unsigned int) * 8.0) - 1)     << endl
```

```
        << "long int        = " << sizeof(long int)        << "       " << fixed <<
setprecision(2)
/*вычисляем максимальное значение для типа данных long int*/
<< (pow(2,sizeof(long int) * 8.0 - 1) - 1)     << endl
        << "unsigned long int  = " << sizeof(unsigned long int)  << "       " <<
fixed << setprecision(2)
/*вычисляем максимальное значение для типа данных undigned long int*/
<< (pow(2,sizeof(unsigned long int) * 8.0) - 1)  << endl
        << "float           = " << sizeof(float)           << "       " << fixed <<
setprecision(2)
/*вычисляем максимальное значение для типа данных float*/
<< (pow(2,sizeof(float) * 8.0 - 1) - 1)        << endl
        << "long float      = " << sizeof(long float)      << "       " << fixed <<
setprecision(2)
/*вычисляем максимальное значение для типа данных long float*/
<< (pow(2,sizeof(long float) * 8.0 - 1) - 1)    << endl
        << "double          = " << sizeof(double)          << "       " << fixed <<
setprecision(2)
/*вычисляем максимальное значение для типа данных double*/
<< (pow(2,sizeof(double) * 8.0 - 1) - 1)       << endl;
    fout.close(); // программа больше не использует файл, поэтому его нужно
закрыть
  cout << "Данные успешно записаны в файл data_types.txt\n";
  system("pause");
  return 0;
}
```

Содержание работы

1. Ознакомиться с теоретической частью методических указаний.
2. Выполнить примеры
3. Представить работающую программу преподавателю.

Лабораторная работа № 22 (2 часа)
Настройка Web-сервера для разработки ASP приложений.

Сервер приложений представляет собой базовую технологию, обеспечивающую инфраструктуру ключа и службы для приложений, находящихся в системе. Обычно серверы приложений содержат перечисленные ниже службы.

- Группировка ресурсов в пул (например, группировка в пул соединения базы данных и объекта).
- Управление распределенными транзакциями.
- Асинхронная связь программ, в основном при помощи очереди сообщений.
- Модель оперативной активации объекта.
- Интерфейсы автоматических веб-служб XML для доступа к рабочим объектам.
- Службы перемещения при сбое и определения работоспособности приложений.
- Интегрированная безопасность.

Операционные системы семейства Windows Server 2003 включают в себя все эти функции, помимо служб разработки, развертывания и управления во время работы веб-службами XML, веб-приложениями и распределенными приложениями.

В этом разделе объясняются основные шаги, которые необходимо проделать при настройке сервера приложений. Этот процесс включает в себя использование мастера настройки сервера для настройки сервера как сервера приложений. После завершения установки основного сервера приложений, дополнительные задачи можно выполнять при помощи программы "Управление данным сервером".

Предварительная подготовка

Прежде чем компьютер будет настроен как сервер приложений, необходимо убедиться в следующем.

- Во всех существующих томах диска используется файловая система NTFS. Тома FAT32 не безопасны и не поддерживают сжатие файлов и папок, дисковые квоты, шифрование файлов и разрешения специального доступа к файлам. Чтобы узнать тип файловой системы, в папке **Мой компьютер** щелкните правой кнопкой мыши том диска и выберите команду **Свойства**. Для получения сведений о преобразовании раздела с файловой системой FAT в раздел с файловой системой NTFS см. раздел "Как преобразовать том в NTFS из командной строки" в центре справки и поддержки.
- Компьютер подключен к сети и имеет статический или динамический IP-адрес.
- Включен брандмауэр Windows. Для получения дополнительных сведений см. раздел
- Мастер настройки безопасности установлен и активен. Для получения дополнительной информации о мастере настройки безопасности см. раздел

- Следующая таблица содержит сведения, необходимые для добавления роли сервера приложений.

Действия, которые следует выполнить перед добавлением роли сервера приложений	Комментарий
Ознакомьтесь со следующими технологиями, которые устанавливаются автоматически во время настройки сервера приложений. • службы IIS • Консоль сервера приложений • COM+ • Координатор распределенных транзакций (DTC)	• IIS 6.0 представляет собой полнофункциональный веб-сервер, обеспечивающий работу инфраструктуры .NET, а также существующих веб-приложений и веб-служб. • COM+ – расширение модели объектных компонентов (COM). COM+ основано на интегрированных службах и свойствах COM, облегчая разработчикам создание и использование компонентов программного обеспечения на любом языке и используя любые средства. • Консоль сервера приложений предоставляет возможность администрирования веб-приложений. Чтобы открыть консоль сервера приложений, в программе "Управление сервером" выберите пункт **Управление этим сервером приложений**. • Координатор распределенных транзакций (DTC) координирует транзакции COM+.
Определите, нужно ли устанавливать серверные расширения FrontPage.	Серверные расширения FrontPage позволяют пользователю на клиентском компьютере удаленно публиковать и администрировать веб-узлы на сервере.
Определите, нужно ли запускать на сервере приложения ASP.NET.	ASP.NET — это единая платформа веб-разработки, которая предоставляет необходимые службы для создания разработчиками веб-приложений корпоративного уровня. ASP.NET можно активировать для разработки веб-приложений.

Настройка сервера приложений

Для настройки сервера приложений необходимо запустить мастер настройки сервера, выполнив следующие действия.

- В программе "Управление данным сервером" выберите ссылку **Добавить или удалить роль**. По умолчанию программа "Управление данным сервером" автоматически загружается при входе в систему. Чтобы открыть оснастку "Управление сервером", нажмите кнопку **Пуск**, выберите **Панель управления**, дважды щелкните **Администрирование**, а затем выберите **Управление сервером**.
- Чтобы открыть мастер настройки сервера, нажмите кнопку **Пуск**, выберите **Панель управления**, дважды щелкните **Администрирование**, а затем выберите **Мастер настройки сервера**.

На странице **Роль сервера** выберите пункт **Сервер приложений (IIS, ASP.NET)** и нажмите кнопку **Далее**.

Параметры сервера приложений

На странице **Параметры сервера приложений** по желанию можно выбрать для установки следующие компоненты сервера приложений.

- **Серверные расширения FrontPage**. Серверные расширения FrontPage позволяют нескольким пользователям с клиентского компьютера удаленно администрировать и публиковать данные на веб-узле. Это свойство следует выбирать, чтобы предоставить возможность нескольким пользователям с клиентских компьютеров создавать веб-приложения или одновременно через Интернет создавать веб-узлы.

- **Включить ASP.NET**. ASP.NET – единая платформа веб-приложений, предоставляющая службы, необходимые для создания и развертывания веб-приложений корпоративного уровня. ASP.NET предлагает новую модель программирования и инфраструктуру для более безопасных, масштабируемых и устойчивых приложений, предназначенных для любых программ обозревателей или устройств. Если на веб-узле имеются приложения, разработанные с помощью ASP.NET, выберите это свойство. Если вы не уверены в этом, ASP.NET можно включить позднее с помощью диспетчера IIS. Это свойство недоступно в версиях операционной системы Windows для компьютеров с процессорами Itanium. После включения ASP.NET на сервере приложений можно размещать приложения ASP.NET. Ниже перечислены некоторые свойства ASP.NET.

    - ASP.NET может работать одновременно с кодом ASP в службах IIS. Если код ASP уже запущен, не обязательно обновлять страницы ASP, кроме того, можно добавлять страницы ASP.NET в приложения.

    - ASP.NET имеет улучшенное быстродействие.

    - ASP.NET поддерживает множество языков, в том числе Visual Basic .NET, C# и JScript .NET.

Для продолжения нажмите кнопку **Далее**.

Сводка выбранных параметров

На странице **Сводка выбранных параметров** посмотрите и подтвердите выбранные параметры. Если на странице **Роль сервера** был выбран пункт **Сервер приложений (IIS, ASP.NET)**, в сводке будут отображены следующие параметры.

- **Установка IIS**

- **Включение COM+ для удаленных транзакций**

- **Включение координатора распределенных транзакций Microsoft (DTC) для удаленного доступа**

Если были выбраны пункты "Серверные расширения FrontPage" или "ASP.NET", отобразятся следующие параметры.

- **Установка серверных расширений FrontPage**

- **Включить ASP.NET**

Для применения параметров, выбранных на странице **Сводка выбранных параметров**, нажмите кнопку **Далее**. После нажатия кнопки **Далее** появится, а затем автоматически закроется страница **Настройка компонентов** мастера компонентов Windows. На этой странице невозможно нажать кнопки **Назад** или **Далее**.

Завершение работы мастера настройки сервера

После настройки компонентов мастер настройки сервера отобразит страницу **Данный сервер теперь является сервером приложений**. Для просмотра всех изменений, сделанных на сервере мастером настройки сервера, или для проверки успешной установки новой роли нажмите кнопку **Просмотр сведений о настройке журнала сервера**. Журнал мастера настройки сервера находится в файле *системный_корневой_каталог*\Debug\Configure Your Server.log. Чтобы завершить работу мастера настройки сервера, нажмите кнопку **Готово**.

Если установка завершилась неудачно, отобразится страница **Продолжение невозможно** и служба IIS не будет установлена. Для устранения неполадок щелкните ссылку **Просмотр сведений о настройке журнала сервера**.

Запуск центра обновления Windows. Для получения дополнительных сведений см. раздел

Запустите мастер настройки безопасности. Для получения дополнительных сведений см. раздел.

Удаление роли сервера приложений

Если требуется перенастроить сервер для другой роли, можно удалить существующую роль. Вместе с удалением роли сервера приложений удаляются все его компоненты, такие как IIS. После этого сервер прекращает поддержку в качестве сервера веб-страниц, веб-приложений или распределенных приложений.

Для удаления роли сервера приложений необходимо перезапустить мастер настройки сервера, выполнив следующие действия.

- В программе "Управление данным сервером" выберите ссылку **Добавить или удалить роль**. По умолчанию программа "Управление данным сервером" автоматически загружается при входе в систему. Чтобы открыть оснастку "Управление сервером", нажмите кнопку **Пуск**, выберите **Панель управления**, дважды щелкните **Администрирование**, а затем выберите **Управление сервером**.

- Чтобы открыть мастер настройки сервера, нажмите кнопку **Пуск**, выберите **Панель управления**, дважды щелкните **Администрирование**, а затем выберите **Мастер настройки сервера**.

На странице **Роль сервера** выберите пункт **Сервер приложений (IIS, ASP.NET)** и нажмите кнопку **Далее**. На странице **Подтверждение удаления роли** просмотрите список, отображенный под заголовком **Сводка**, установите флажок **Удалить роль сервера приложений** и нажмите кнопку **Далее**. После нажатия кнопки **Далее** появится, а затем автоматически закроется страница **Настройка компонентов** мастера компонентов

Windows. На этой странице невозможно нажать кнопки **Назад** или **Далее**. На странице **Роль сервера приложений удалена** нажмите кнопку **Готово**.

Дальнейшие действия: выполнение дополнительных задач

После завершения работы мастера настройки сервера и включения возможностей, необходимых для работы приложений, компьютер готов к использованию в качестве основного сервера приложений. К данному моменту завершены следующие задачи:

- установка IIS, ASP.NET, и COM+;
- включение серверных расширений FrontPage, если требуется;
- включение ASP.NET, если требуется.

Следующая таблица содержит некоторые дополнительные задачи, которые можно выполнять на сервере приложений.

Задача	Назначение задачи	Ссылка
Защита сервера приложений	Для гарантии безопасности этих серверов рекомендуется принять меры предосторожности, такие как брандмауэры и безопасность протокола IP (IPSEC), перед использованием их в производственной среде. Сервер приложений может стать объектом атак злоумышленников, так как он будет доступен через Интернет и другие сети. Защитить приложения можно с помощью протоколов проверки подлинности, управления доступом, протокола SSL и шифрования.	Безопасность протокола IP (IPSec);Основной брандмауэр; Система безопасности служб Microsoft Internet Information Services
Защита файлов с помощью NTFS	Чтобы обеспечить безопасность веб-узла, приложений, баз данных и файлов, используйте разрешения NTFS. Это чрезвычайно важно для защиты узла.	Задание, просмотр, смена или удаление разрешений для доступа к файлам и папкам
Настройка веб-интерфейса для удаленного администрирования	Позволяет управлять сервером приложений через веб-обозреватель удаленного компьютера.	Использование веб-интерфейса для удаленного администрирования
Создание веб-узла	Веб-узел необходим для размещения веб-приложений.	Установка веб-узла в службах Microsoft Internet Information Services

Создание приложений с помощью новейших средств разработки	Узнайте о новейших средствах разработки от корпорации Майкрософт, позволяющие быстрее и эффективнее разрабатывать новые приложения.	Веб-узел корпорации Майкрософт
Создание веб-приложений ASP.NET	Создание приложений ASP.NET.	Создание веб-приложений ASP.NET на веб-узле корпорации Майкрософт
Защита веб-приложений ASP.NET	Обеспечение безопасности приложений ASP.NET.	Безопасность веб-приложений ASP.NET на веб-узле корпорации Майкрософт
Настройте порты для разрешения удаленного администрирования.	Управление сервером приложений с других компьютеров сети.	Параметры брандмауэра Windows

Содержание работы

1. Ознакомиться с теоретической частью методических указаний.
2. Выполнить примеры
3. Представить работу преподавателю.

Лабораторная работа № 23 (2 часа)
Создание Web-приложений на базе ASP

Почти все крупномасштабные веб-сайты ASP.NET создаются с помощью Visual Studio.

В состав этого профессионального средства для разработки входит развитый набор инструментов для проектирования, в том числе легендарные инструменты для отладки и **механизм IntelliSense**, способный перехватывать ошибки и предлагать варианты по мере ввода. Кроме того, в Visual Studio поддерживается мощная модель отделенного кода, которая позволяет разделять создаваемый код .NET и дескрипторы разметки веб-страницы. И, наконец, в Visual Studio имеет встроенный тестовый вебсервер, который значительно упрощает процесс отладки веб-сайтов.

Большая часть изменений относится к таким деталям, как меньшая степень загромождения экрана и улучшенная функция IntelliSense. Но разработчики, имеющие дело с WPF или Silverlight, получили в свое распоряжение столь долгожданный визуальный конструктор, который позволяет строить пользовательский интерфейс за счет перетаскивания нужных элементов управления из панели Toolbox точно так же, как при работе со страницами ASP.NET.

Написание и компиляция кода вручную было бы утомительным делом для любого разработчика. К счастью, IDE-среда Visual Studio предлагает набор высокоуровневых функциональных возможностей, которые выходят за рамки базового управления кодом.

Ниже перечислены основные преимущества IDE-среды Visual Studio:

Наличие встроенного веб-сервера

Для обслуживания веб-приложения ASP.NET необходим веб-сервер, подобный IIS, который будет ожидать веб-запросы и обрабатывать соответствующие страницы.

Установить свой веб-сервер несложно, но может быть не совсем удобно. Наличие в Visual Studio интегрированного веб-сервера позволяет запускать веб-сайт прямо из среды проектирования, а также повышает безопасность, исключая вероятность получения доступа к тестовому вебсайту с какого-нибудь внешнего компьютера, поскольку тестовый сервер может принимать соединения только с локального компьютера.

Поддержка использования множества языков для разработки

Visual Studio позволяет писать код на своем языке или любых других предпочитаемых языках, используя все время один и тот же интерфейс (IDE).

Более того, Visual Studio также еще позволяет создавать веб-страницы на разных языках, но помещать их все в один и тот же веб-сайт. Существуют лишь два ограничения, о которых следует помнить: не использовать более одного языка в одной и той же веб-странице (что может вызвать очевидные проблемы на этапе компиляции) и обязательно применять модель веб-сайта без проекта (а не веб-проект).

Меньше кода для написания

Для создания большинства приложений требуется приличное количество стандартного стереотипного кода, и веб-страницы ASP.NET тому не исключение.

Например, добавление веб-элемента управления, присоединение обработчиков событий и корректировка форматирования требует установки в разметке страницы ряда деталей. В Visual Studio такие детали устанавливаются автоматически.

Интуитивно понятный стиль кодирования

По умолчанию Visual Studio форматирует код по мере его ввода, автоматически вставляя необходимые отступы и применяя цветовое кодирование для выделения элементов типа комментариев. Такие незначительные отличия делают код более удобным для чтения и менее подверженным ошибкам.

Применяемые Visual Studio автоматически параметры форматирования можно даже настраивать, что очень удобно в случаях, когда разработчик предпочитает другой стиль размещения скобок (например, стиль Кернигана и Ритчи, при котором открывающая скобка размещается на той же строке, что и объявление, которому она предшествует).

Изменить параметры форматирования в Visual Studio можно, выбрав в меню Tools (Сервис) пункт Options (Параметры) и затем просмотреть группы параметров Text Editor --> C# --> Formatting (Текстовый редактор --> C# --> Форматирование). Среди них будут и параметры, отвечающие за место размещения фигурных скобок.

Более высокая скорость разработки

Многие из функциональных возможностей Visual Studio направлены на то, чтобы помочь разработчику делать свою работу как можно быстрее.

Удобные функции, такие как IntelliSense (которая умеет перехватывать ошибки и предлагать правильные варианты), поиск и замена (которая позволяет отыскивать ключевые слова как в одном файле, так и во всем проекте) и автоматическое добавление и удаление комментариев (которая может временно скрывать блоки кода), позволяют разработчику работать быстро и эффективно.

Возможности отладки

Предлагаемые в Visual Studio инструменты отладки являются наилучшим средством для отслеживания загадочных ошибок и диагностирования странного поведения. Разработчик может выполнять свой код по строке за раз, устанавливать интеллектуальные точки останова, при желании сохраняя их для использования в будущем, и в любое время просматривать текущую информацию из памяти.

Веб-сайты и веб-проекты

В Visual Studio поддерживаются два пути создания веб-приложений на базе ASP.NET,

Разработка на основе проекта

При создании веб-проекта в Visual Studio генерируется файл проекта с расширением .csproj (при условии, что код пишется на языке C#), в котором фиксируется информация обо всех включаемых в состав проекта файлах и сохраняются кое-какие отладочные параметры. При запуске веб-проекта перед открытием веб-браузера Visual Studio сначала компилирует весь написанный код в одну сборку.

Разработка без использования проекта

Это альтернативный подход, который подразумевает создание просто веб-сайта без файла проекта.

При таком подходе Visual Studio предполагает, что каждый файл в каталоге веб-сайта (и всех его подкаталогах) является частью веб-приложения. В этом случае Visual Studio не требуется предварительно компилировать код. Вместо этого ASP.NET компилирует уже сам веб-сайт при первом запросе какой-нибудь входящей в его состав страницы.

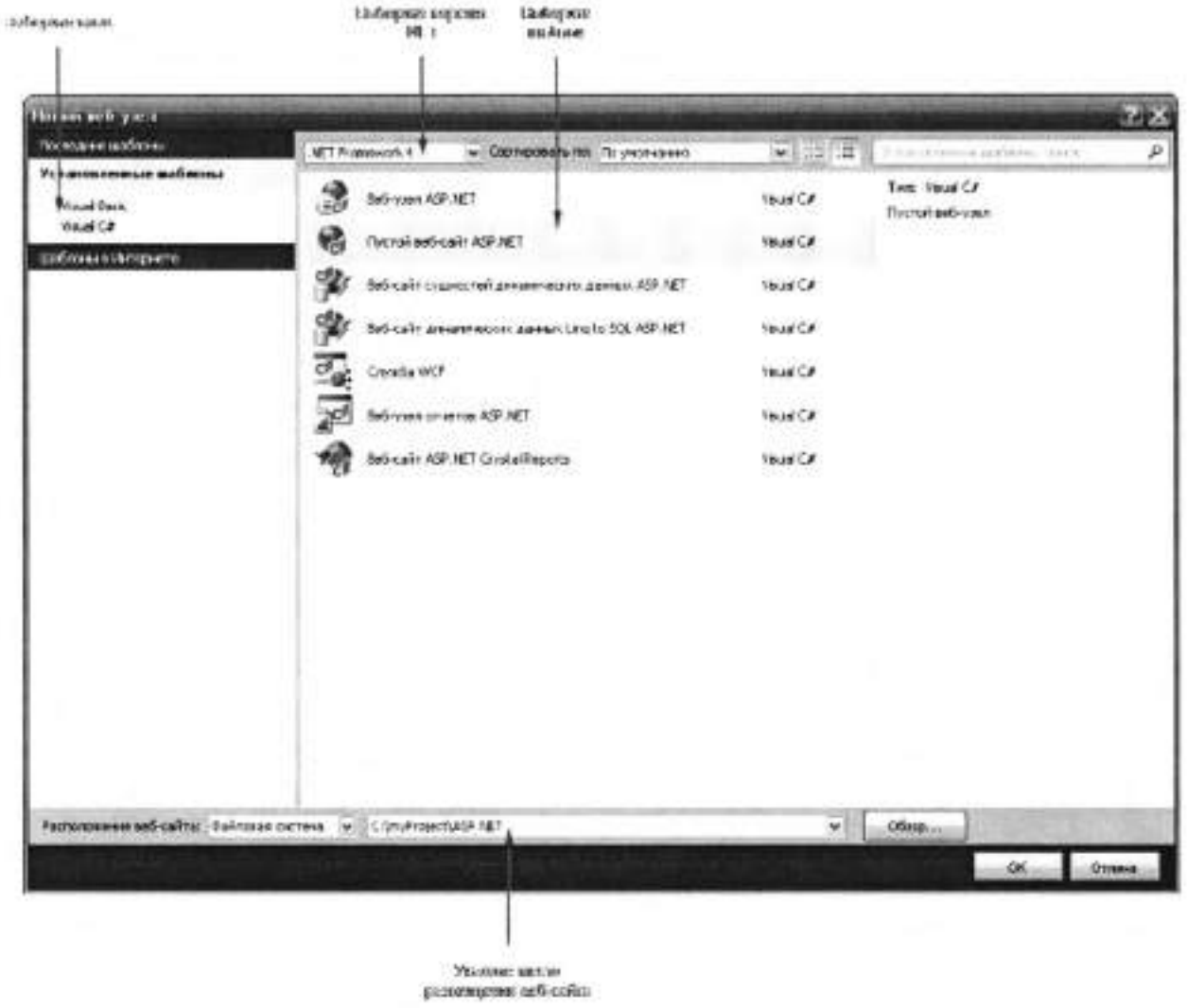
Разумеется, разработчик может применить предварительную компиляцию и устранить связанные с первым запросом непроизводительные издержки для развертываемого веб-приложения.

В первой версии Visual Studio для .NET использовалась модель разработки на основе проекта. В Visual Studio 2005 эта модель была удалена и заменена разработкой без использования проекта. Однако это вызвало возмущение среди разработчиков.

Осознав, что для определенных сценариев все-таки больше подходила проектная разработка, Microsoft выпустила доступный для загрузки пакет, который возвращал в Visual Studio 2005 опцию проектной модели. В версии Visual Studio 2010 поддерживаются оба подхода.

Создание веб-сайта без использования проекта

Чтобы сразу же приступить к работе и создать новое веб-приложение, выберите в меню File (Файл) пункт New --> Website (Создать Веб-узел). Откроется диалоговое окно New Web Site, показанное на рисунке:



Для создания нового веб-сайта необходимо выбирать язык разработки (слева), версию .NET (посередине вверху), шаблон веб-сайта (в середине) и место размещения (внизу).

Каждая деталь более подробно описана в следующих разделах.

Язык разработки

Под этим подразумевается язык программирования .NET, который будет использоваться для написания кода веб-сайта. Выбранный язык просто назначается для проекта языком по умолчанию. Это значит, что можно явно добавлять веб-страницы на Visual Basic к веб-сайту на C# и наоборот.

Версия .NET

Прежние версии Visual Studio были тесно связаны с конкретными версиями .NET. Так, например, Visual Studio .NET можно было применять только для создания приложений .NET 1.0, Visual Studio 2003 — только для приложений .NET 1.1, а Visual Studio 2005 — для приложений .NET 2.O.

В версии Visual Studio 2008 это ограничение было устранено и стало возможно выбирать в качестве целевой любую версию .NET. В Visual Studio 2010 эта тенденция была

продолжена. В этой последней версии теперь можно создавать веб-приложения, ориентированные на работу с такими версиями, как NET 2.0, .NET 3.0, .NET 3.5 и .NET 4.

Обычно следует выбирать версию, которая поддерживается на используемом веб-сервере. Более поздние версии предоставляют доступ к новейшим функциональным возможностями.

Разумеется, на одном веб-сервере можно установить сразу несколько версий .NET и сконфигурировать разные виртуальные каталоги на использование различных версий ASP.NET.

Для поддержки множества целевых платформ в Visual Studio 2010 включены ссылочные сборки для каждой версии .NET. Такие сборки содержат метаданные для каждого типа, но не код их реализации.

Это означает, что Visual Studio 2010 может использовать упомянутые сборки для подстройки списка IntelliSense и проверки на предмет ошибок, не давая возможности пользоваться элементами управления, классами или членами, которые не доступны в выбранной целевой версии .NET. Эти метаданные также могут применяться для определения того, какие элементы управления должны присутствовать в панели Toolbox (Элементы управления), какие члены должны отображаться в окне Properties (Свойства) и Object Browser (Браузер объектов) и т.д. В результате вся IDE-среда будет ограничена выбранной версией .NET.

Изменить целевую версию .NET можно даже после создания веб-сайта. Для этого выполните следующие шаги:

- Выберите в меню Website (Веб-сайт) пункт Start Options (Параметры запуска).
- В списке слева выберите категорию Build (Сборка).
- В списке Target Framework (Целевая платформа) выберите желаемую версию .NET.

В случае веб-проекта этот процесс выглядит несколько иначе. В окне Solution Explorer дважды щелкните на узле Properties (Свойства) и перейдите на вкладку Application (Приложение). Эта вкладка содержит список Target Framework (Целевая платформа), в котором можно выбрать желаемую версию .NET.

При изменении версии .NET среда Visual Studio существенно модифицирует соответствующий файл web.config. Например, этот файл для приложения .NET 4 имеет довольно короткий и простой вид, поскольку все необходимые низкоуровневые детали устанавливаются в корневом файле web.config самого компьютера. Однако для приложения .NET 3.5 файл web.config содержит приличный объем дополнительного кода, который явно включает поддержку Ajax и функциональных средств C# 3.5.

Шаблон

После выбора языка (в списке слева) появляется список всех шаблонов, предоставляемых Visual Studio для этого языка (в большой области в центре). Каждый шаблон определяет то, с каких файлов будет начинаться создание веб-сайта.

В Visual Studio поддерживается несколько типов приложений ASP.NET, но все они компилируются и выполняются одинаково. Единственное, чем они отличаются, так это тем, какие файлы для них создаются в Visual Studio по умолчанию. Например, в случае

создания приложения типа WCF Service (Служба WCF) в Visual Studio генерируется веб-сайт, в котором изначально содержится только одна служба WCF, а не веб-страница ASP.NET.

Ниже приведено краткое описание всех доступных шаблонов:

ASP.NET Web Site (Веб-узел ASP.NET)

Создается полнофункциональный веб-сайт ASP.NET с готовой базовой инфраструктурой. Этот веб-сайт включает главную страницу с общей компоновкой (верхним колонтитулом, нижними колонтитулом и строкой меню) и готовые страницы default.aspx и about.aspx.

Также он содержит папку Accounts (Учетные записи) со страницами для регистрации, входа в систему и смены пароля и папку Scripts (Сценарии) с библиотекой jQuery для клиентских сценариев JavaScript.

ASP.NET Empty Web Site (Пустой веб-сайт ASP.NET)

Создается почти пустой веб-сайт. Этот веб-сайт включает только сокращенную версию конфигурационного файла web.config, и ничего больше. Разумеется, с началом кодирования в него можно легко добавлять необходимые фрагменты.

Если вы — новичок в мире ASP.NET, лучше начинайте с выбора шаблона ASP.NET Empty Web Site. Освоив функциональные средства, подобные мастер-страницам и членству, можно будет перейти на применение несколько более сложного шаблона ASP.NET Web Site, если он отвечает существующим потребностям.

ASP.NET Dynamic Data Entities Web Site (Веб-сайт сущностей динамических данных ASP.NET)

Создается веб-сайт ASP.NET, использующий компонент ASP.NET Dynamic Data. Этот веб-сайт ориентирован на применение сущностной модели (Entity Model) для доступа к серверной базе данных, в то время как шаблон со схожим именем ASP.NET Dynamic Data LINQ to SQL Web Site предусматривает использование для этого более старого подхода LINQ to SQL.

WCF Service (Служба WCF)

Создается служба WCF — библиотека серверных методов, которые удаленные клиенты (например, Windows-приложения) могут вызывать.

ASP.NET Reports Web Site (Веб-узел отчетов ASP.NET)

Создается веб-сайт ASP.NET с элементом управления ReportView и компонентом *SQL Server Reporting Services - инструмент для генерации отчетов по базам данных*, которые можно просматривать и обрабатывать через Интернет. Шаблон ASP.NET Crystal Reports Web Site (Веб-сайт ASP.NET с Crystal Reports) обеспечивает похожие возможности, но с использованием программного обеспечения Crystal Reports.

Хотя большинство разработчиков предпочитает выбирать сначала шаблон ASP.NET Empty Web Site или ASP.NET Web Site и приступать к написанию кода, доступно множество специализированных шаблонов, предназначенных для создания веб-приложений конкретных типов.

Чтобы просмотреть их, щелкните на заголовке Online Templates (Шаблоны в Интернете) в левом крайнем углу диалогового окна New Web Site. Через некоторое время, необходимое для установления связи с веб-серверами Microsoft, добавляется список различных подкатегорий, каждая с собственной группой готовых шаблонов.

Например, разработчики ASP.NET могут загрузить из этого списка шаблон для создания веб-сайта DotNetNuke (который использует популярную платформу для создания порталов DotNetNuke) или веб-сайта ASP.NET MVC (который применяет OpenID для аутентификации пользователей).

Место размещения

Место размещения отвечает за то, где будут храниться файлы веб-сайта. Обычно выбирается вариант File System (Файловая система) и затем указывается либо папка на локальном компьютере, либо сетевой путь.

Однако веб-сайт также допускается редактировать и непосредственно через HTTP или **FTP (File Transfer Protocol — протокол передачи файлов)**. Такой подход иногда удобен, когда требуется "вживую" выполнять редактирование веб-сайта на удаленном веб-сервере. С другой стороны, он влечет за собой дополнительные накладные расходы. Конечно, производить редактирование напрямую на производственном сервере не следует никогда, поскольку такие изменения являются автоматическими и необратимыми. Вместо этого лучше ограничивать свои изменения только тестовыми серверами.

Если просто нужно создать свой проект в какой-то папке внутри файловой системы, можно ввести имя этой папки в поле Location (Размещение) вручную. Но если интересуют все возможные варианты для определения подходящего места, щелкните на кнопке Browse (Обзор). Откроется диалоговое окно Choose Location (Выбор размещения). В левой части этого окна есть четыре кнопки, позволяющие выбирать различные варианты для размещения файлов.

Содержание работы

1. Ознакомиться с теоретической частью методических указаний.
2. Выполнить примеры
3. Получить вариант задания от преподавателя.
4. Решить поставленную задачу
5. Представить работающую программу преподавателю.

Лабораторная работа № 24 (2 часа)
Создание Web сервисов XML на базе ASP.NET

Для создания web-приложений на основе .NET Framework средствами Microsoft Visual Studio используется технология Active Server Pages .NET (ASP .NET), которая представляет собой унифицированную модель создания web-приложений, поддерживающую разработку на всех языках программирования, совместимых с Common Language Runtime (CLR), включая такие языки, как Microsoft Visual Basic, C#, JScript .NET и J#. В состав ASP .NET входят модель описания страниц и компонентов, располагаемых на этих страницах, специальный компилятор, инфраструктура обеспечения безопасности, средства управления состоянием, механизмы конфигурации приложений, средства мониторинга производительности приложений, средства отладки, поддержка создания web-сервисов на базе XML, поддержка хостинга и управления жизненным циклом приложения, а также профессиональные средства дизайна и разработки web-приложений. Рассмотрим каждый из перечисленных компонентов подробнее.

Модель описания страниц и компонентов

Модель описания страниц и компонентов представляет собой набор программных средств, выполняемых на web-сервере и динамически генерирующих и отрисовывающих web-страницы (называемые ASP .NET web-страницами). Такие страницы могут использоваться в любом браузере или мобильном устройстве — ASP .NET возвращает запросившему браузеру код страницы на языке HTML. ASP .NET web-страницы полностью объектно-ориентированы. Это означает, что в рамках страницы вы можете обращаться к HTML-элементам через их свойства, методы и событийную модель. ASP .NET предоставляет в распоряжение разработчиков унифицированную модель реакции на клиентские события в коде, который выполняется на сервере. Программная модель автоматически поддерживает состояния для всей страницы и ее отдельных элементов во время цикла обработки страницы.

Функциональность пользовательского интерфейса реализована в ASP .NET на уровне набора интерфейсных компонентов, которые представляют собой как основные элементы, поддерживаемые на уровне языка HTML, так и дополнительные компоненты, расширяющие интерфейсные возможности, предоставляемые языком HTML.

В состав ASP .NET входят следующие типы компонентов:

- **серверные HTML-компоненты** — они представляют собой программную реализацию стандартных HTML-элементов и позволяют управлять различными аспектами их поведения из кода, выполняемого на сервере.

По умолчанию HTML-элементы, включенные в состав файла ASP .NET, представляют собой обычный текст и не могут использоваться из серверного кода (они доступны только из клиентского JavaScript-кода через DOM-модель). Для того чтобы такие элементы стали программно доступны, они должны представлять собой серверные HTML-компоненты — для этого в HTML-текст добавляется атрибут runat="server" (рис. 1).

Рис. 1. HTML-элементы в галерее элементов

Помимо этого для программного обращения к таким элементам можно применять атрибут id, задающий идентификатор соответствующего элемента. Атрибуты также используются для задания свойств элементов и связи их с обработчиками событий. Например, HTML-элемент <button> соответствует серверному компоненту HtmlButton, который имеет следующие атрибуты:

<button
CausesValidation="False|True"
Disabled="Disabled"
EnableViewState="False|True"
Id="string"
ValidationGroup="String"
Visible="False|True"
OnDataBinding="OnDataBinding event handler"
OnDisposed="OnDisposed event handler"
OnInit="OnInit event handler"
OnLoad="OnLoad event handler"
OnPreRender="OnPreRender event handler"
OnServerClick="OnServerClick event handler"
OnUnload="OnUnload event handler"

```
runat="server"
>
<!-- Текст, отображаемый на кнопке, —>
</button>
```

Связь с обработчиком события, означающего нажатие клавиши, задается следующим образом:

```
<%@ Page Language="C#" AutoEventWireup="True" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>HtmlButton Control</title>
<script runat="server">
void Button1_OnClick(object Source, EventArgs e)
{
Span1.InnerHtml="You clicked Button1";
}
</script>
</head>
<body>
<h3>HtmlButton </h3>
<form id="Form1" runat="server">
<p />
<button id="Button1"
onserverclick="Button1_OnClick"
style="font: 8pt verdana;
border-color:black;
height:30;
width:100"
runat="server">
Click me!
</button>
</form>
</body>
</html>
```

- **серверные web-компоненты** — такие компоненты расширяют набор стандартных компонентов, поддерживаемых на уровне языка HTML: календарь, меню, средства отрисовки древовидных структур и т.п., а также предоставляют в распоряжение разработчиков программную модель, позволяющую управлять различными аспектами их поведения из кода, выполняемого на сервере (рис. 2).

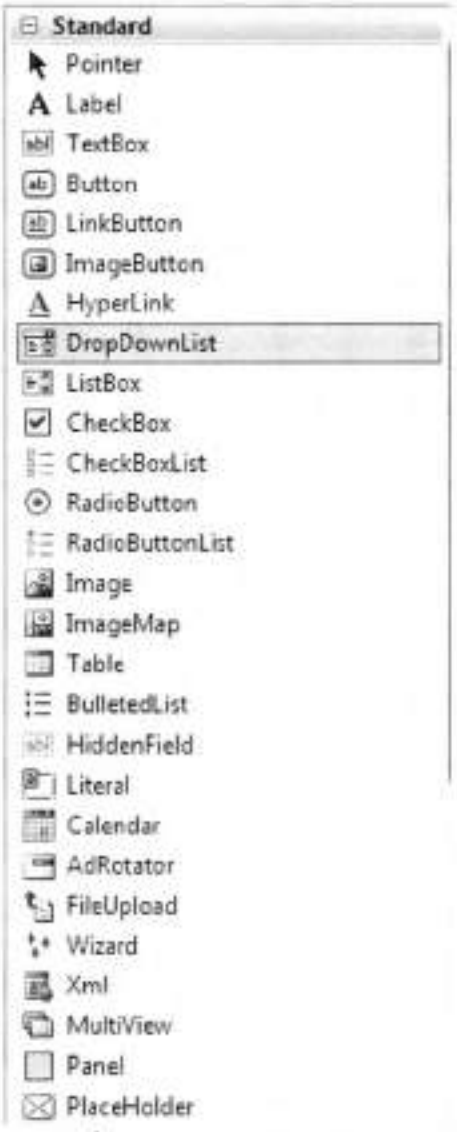


Рис. 2. Серверные web-компоненты в галерее компонентов

В состав ASP .NET входят web-компоненты, представленные в <u>табл. 2</u>.

Web-компоненты описываются атрибутом asp:. Например, вот как описывается кнопка в виде серверного web-компонента:

```
<asp:Button id="SubmitButton"
Text="Submit"
CommandName="Submit"
OnCommand="CommandBtn_Click"
runat="server"/>
```

и обработчик события для нее:

```
void CommandBtn_Click(Object sender, CommandEventArgs e)
{
Message.Text = "You clicked the " + e.CommandName +
" — " + e.CommandArgument + " button.";
}
```

- **компоненты проверки ввода** — эти компоненты позволяют описывать логику проверки данных, вводимых пользователями в

компоненте TextBox. Компоненты проверки ввода позволяют задавать поля, значения которых не могут быть пустыми, указывать шаблоны вводимой информации, диапазоны допустимых значений и т.п. В ASP .NET входят следующие компоненты проверки ввода (**табл. 3**).

Компоненты проверки ввода используются совместно с компонентом **ValidationSummary**, который позволяет отображать сообщения об ошибках, выдаваемые данными компонентами (рис. 3);

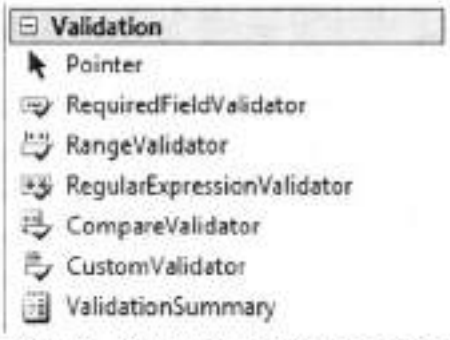


Рис. 3. Компоненты проверки ввода в галерее компонентов

- **пользовательские компоненты** — это компоненты, которые разработчики создают для расширения набора стандартных интерфейсных элементов, входящих в состав ASP .NET.

Модель описания страниц и компонентов также позволяет управлять внешним видом интерфейсных элементов — для этого применяются темы и скины. Возможно использование тем и скинов как на уровне страниц, так и на уровне отдельных компонентов.

Помимо тем, в ASP .NET поддерживаются так называемые мастер-страницы, позволяющие задавать единое расположение элементов для всех страниц (или группы страниц), входящих в состав web-приложения.

Специальный компилятор

Код ASP .NET-страниц является компилируемым, что, помимо всего прочего, обеспечивает поддержку строгой типизации, оптимизацию производительности и раннее связывание. После того как код откомпилирован, ядро исполнения .NET — Common Language Runtime — преобразует его в машинный код для обеспечения максимальной производительности. В состав ASP .NET входит компилятор для обработки компонентов web-приложения, включая страницы и визуальные компоненты, — они преобразуются в сборки, которые выполняются под управлением среды ASP .NET.

Инфраструктура обеспечения безопасности

В дополнение к механизмам безопасности, реализованным на уровне .NET, ASP .NET включает расширенную инфраструктуру безопасности, обеспечивающую сервисы аутентификации и авторизации пользователей, а также решение ряда связанных с обеспечением безопасности задач. Поддерживается возможность использования аутентификации Windows, поддерживаемой на уровне Internet Information Services, либо собственных механизмов аутентификации на основе форм ASP .NET и ASP .NET Membership с применением собственной базы данных пользователей. Помимо этого имеется возможность управления авторизацией web-приложений путем использования группы Windows или роли на уровне ASP .NET с помощью собственной базы данных. Поддерживается возможность удаления, добавления или замены описанных выше механизмов в зависимости от требований к конкретному web-приложению.

Отметим, что ASP .NET всегда выполняется под определенной учетной записью Windows — таким образом, вы можете дополнительно защитить приложение, используя такие возможности Windows, как NTFS Access Control Lists (ACLs), разрешения на уровне баз данных и т.п.

Средства управления состоянием

В ASP .NET входят встроенные средства управления состоянием, позволяющие сохранять информацию между запросами к странице. Наиболее часто такие сервисы требуются для хранения данных о пользователях или о содержимом корзины покупателя в сценариях электронной коммерции. Поддерживается возможность сохранять и управлять информацией на уровне приложения, сессии, страницы, пользователя и т.п. Разработчики могут расширять эти механизмы для предоставления дополнительной функциональности по хранению данных. Такая информация может быть полностью независимой от компонентов, располагаемых на странице. В ASP .NET поддерживаются распределенные средства сохранения информации — например для нескольких экземпляров приложения, выполняющихся на одном или нескольких серверах.

Механизмы конфигурации приложений

Приложения, создаваемые средствами ASP .NET, используют систему конфигурации, которая позволяет задавать конфигурационные настройки для web-сервера, web-сайта или определенного приложения. Эти настройки можно выполнять в процессе развертывания ASP .NET-приложений — это минимально влияет на работу как других web-приложений, так и самого сервера. Настройки ASP .NET хранятся в XML-файлах, благодаря чему вносить изменения в конфигурацию можно с помощью различных редакторов. При необходимости конфигурационная схема может быть расширена: ASP .NET не устанавливает никаких ограничений на расширения содержимого конфигурационных файлов.

Средства мониторинга производительности приложений

В ASP .NET входят средства, позволяющие выполнять мониторинг производительности и других ключевых характеристик приложений. Средства слежения за «здоровьем» приложений (health monitoring) дают возможность получить отчеты о ключевых событиях, связанных с жизненным циклом приложения, и различных состояниях, в которых оно может находиться, включая ошибки. Эти события объединяют ключевые характеристики мониторинга и диагностики и обеспечивают максимальную гибкость при определении, какие данные и когда должны заноситься в протокол. В ASP .NET поддерживается две группы счетчиков производительности, которые доступны из приложений: группа системных счетчиков, собирающих данные о работе всей подсистемы, и группа прикладных счетчиков, отвечающих за сбор данных о конкретном приложении.

Средства отладки

В ASP .NET полностью используется инфраструктура времени исполнения, предоставляемая .NET CRL для обеспечения функций отладки. Поддерживается отладка как объектов, написанных на неуправляемом коде, так и объектов, созданных средствами Microsoft .NET с применением любых языков программирования, поддерживаемых на уровне CLR, а также скриптовых языков. Помимо этого модель описания страниц и компонентов поддерживает режим трассировки, позволяющий разработчикам использовать механизмы управления приложениями (instrumentation) непосредственно в web-страницах, создаваемых средствами ASP .NET.

Поддержка создания web-сервисов на базе XML

В ASP .NET полностью поддерживаются web-сервисы на базе XML. Такой web-сервис представляет собой компонент, содержащий бизнес-функциональность и позволяющий приложениям обмениваться информацией через межсетевые экраны с помощью таких стандартов, как протоколы HTTP и XML Messaging. Web-сервисы не связаны с какой-то конкретной компонентной технологией или соглашением о вызове объектов.
В результате приложения, написанные на любом языке программирования и использующие любую компонентную модель и любую программную платформу, могут обращаться к web-сервисам.

Поддержка хостинга и управления жизненным циклом приложения

В состав ASP .NET входит среда хостинга с возможностью расширения, которая позволяет управлять жизненным циклом приложений — от первого доступа к определенному ресурсу (например, к web-странице) приложения до завершения работы приложения (остановке web-сервера). ASP .NET

использует web-сервер (Internet Information Services, IIS) в качестве сервиса для хостинга среды выполнения и самих приложений, но при этом самостоятельно обеспечивает расширенные возможности хостинга. Например, архитектура ASP .NET позволяет обрабатывать события на уровне приложений и создавать собственные обработчики HTTP-запросов и HTTP-модули.

Профессиональные средства дизайна и разработки web-приложений

Все описанные выше ключевые возможности технологии ASP .NET полностью поддерживаются в средствах визуального дизайна и разработки web-приложений, входящих в состав Visual Studio.

Создание ASP .NET-приложений в Microsoft Visual Studio строится на использовании шаблонов, выбираемых из ветви Web для поддерживаемых языков программирования — Visual Basic .NET и Visual C#. Доступные шаблоны показаны на рис. 4.

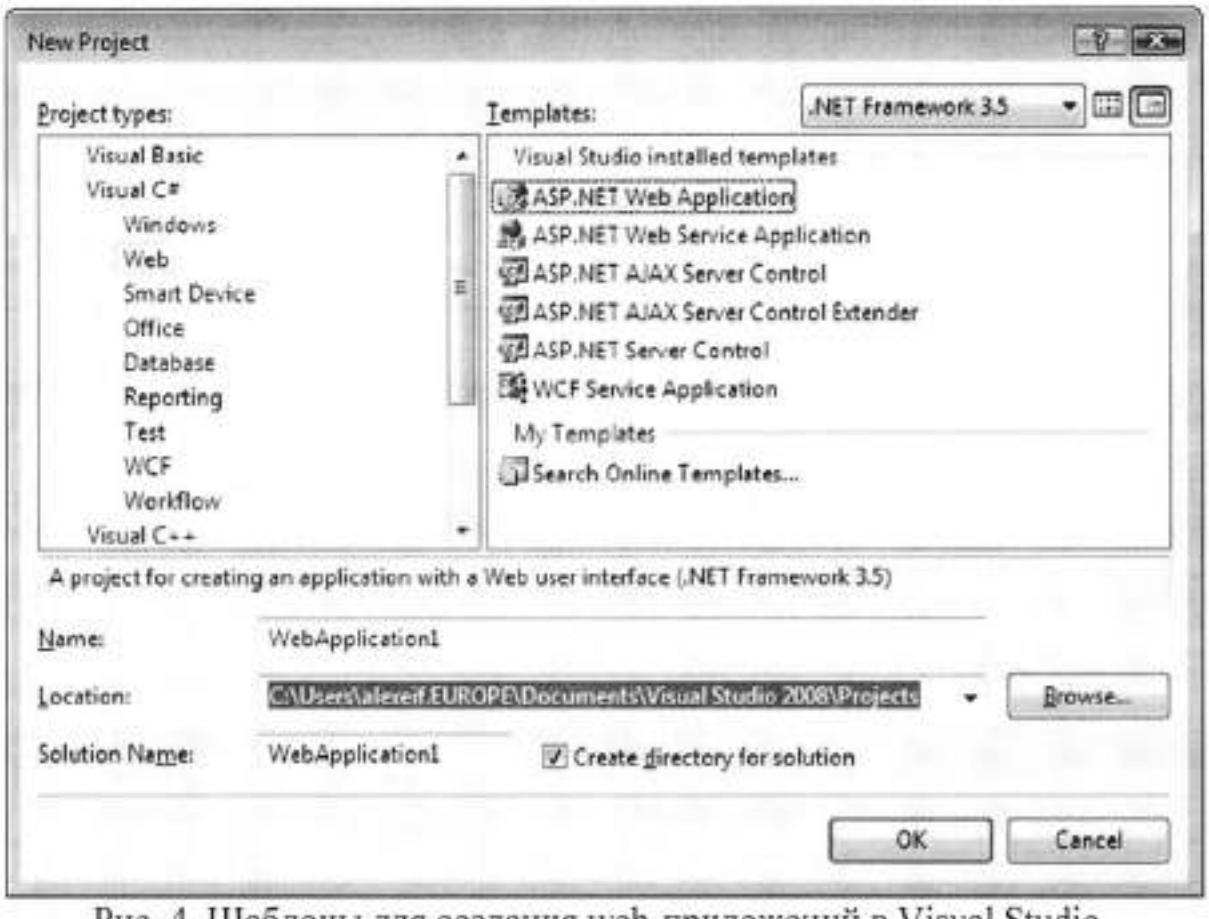


Рис. 4. Шаблоны для создания web-приложений в Visual Studio

После выбора шаблона (в большинстве случаев для web-приложений используется шаблон ASP .NET Web Application) мы попадаем в дизайнер — среду визуального создания web-приложений. В данной среде можно работать с web-страницами на уровне исходного кода (вкладка *Source*), на уровне дизайна (вкладка *Design*) или в совмещенном режиме (вкладка *Split*). Последний из них является новинкой в Visual Studio 2008. Галерея

компонентов содержит различные компоненты — HTML и Web — и может использоваться для их перетаскивания на разрабатываемую страницу. Панель управления свойствами позволяет настраивать разные атрибуты самой страницы и размещенных на ней компонентов, а двойной щелчок мышью по компоненту приводит к появлению редактора для написания кода для обработки соответствующего события.

Поддерживается автоматическая проверка кода на соответствие различным стандартам, включая Internet Explorer 6.0, Internet Explorer 3.02/Netscape Navigator 3.0, Netscape Navigator 4.0, HTML 4.01, XHTML 1.0 и XHTML 1.1. По умолчанию применяется режим совместимости с XHTML 1.0 Transitional, поддерживаемым в Netscape 7, Opera 7 и Internet Explorer 6 (рис. 5).

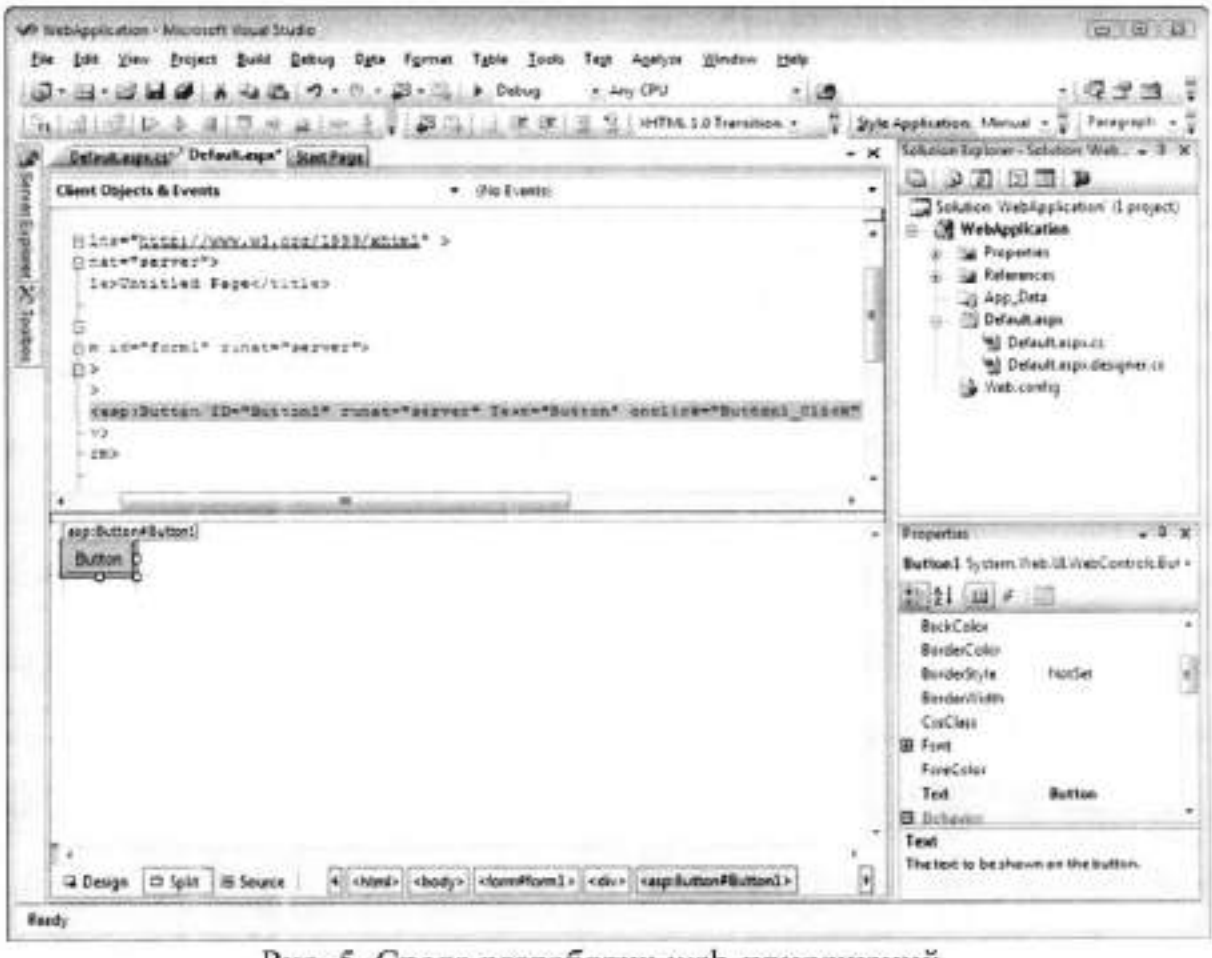


Рис. 5. Среда разработки web-приложений

Интегрированная поддержка ASP .NET AJAX 1.0

Технология AJAX — это технология создания web-приложений, обладающих повышенной интерактивностью, скоростью и производительностью. AJAX базируется на комбинации таких технологий, как XHTML (или HTML) и CSS, клиентский скриптинг (JavaScript, Jscript) и объект **XMLHttpRequest**. Это позволяет клиенту (браузеру) обмениваться с сервером небольшими объемами данных, необходимыми для обновления

определенных частей страницы, а не всей страницы, как при использовании традиционных технологий создания web-приложений. Применение AJAX позволяет создавать более богатые интерфейсы для web-приложений, что делает пользователей более продуктивными и обеспечивает реализацию сценариев, ранее недоступных для тонких клиентов.

Технологический набор, обеспечивающий функционирование технологии AJAX, был разработан компанией Microsoft и в настоящее время поддерживается всеми ведущими производителями. Первым шагом на пути реализации данной технологии стало появление HTML-элемента **IFRAME** в Microsoft Internet Explorer 3.0, выпущенном в 1996 году. Использование данного элемента позволяло обновлять часть web-страницы без необходимости в полной перезагрузке всей страницы. В 1998 году Microsoft реализовала концепцию удаленного скриптинга (remote scripting), которая стала альтернативой элементу **IFRAME**, а в браузере Internet Explorer 5.0 появился объект **XMLHttpRequest**, который очень эффективно применялся для обеспечения функционирования тонкого клиента почтовой программы Outlook — Outlook Web Access.

В Microsoft Visual Studio 2008 входят шаблоны для создания серверных компонентов на базе технологии AJAX (ASP .NET AJAX Server Control) и расширений для таких компонентов (ASP .NET AJAX Server Control Extender).

Поддержка ASP .NET AJAX реализована на основе следующих ключевых компонентов:

- Microsoft AJAX Library:
  - клиентская библиотека на JavaScript,
  - работает на любом браузере и поддерживается любым web-сервером (включая PHP, ColdFusion и т.п.);
- ASP .NET 2.0 AJAX Extensions — серверные расширения для интеграции с ASP .NET 2.0;
- ASP .NET AJAX Control Toolkit — бесплатный набор компонентов с полным исходным текстом, доступный для загрузки по адресу: http://ajax.asp.net.

Ключевыми компонентами, обеспечивающими поддержку технологии AJAX в ASP .NET-приложениях, являются **ScriptManager** (и **ScriptManagerProxy**) и **UpdatePanel**. Компонент **UpdatePanel** используется для задания регионов страницы, которые должны быть обновляемыми без обновления содержимого всей страницы:

```
<asp:UpdatePanel id="updatepanel1" runat="server">
<ContentTemplate>
<!—Содержимое будет автоматически обновляться! —>
<asp:Calendar id="calndr1" runat="server"/>
<ContentTemplate>
</asp:UpdatePanel>
```

Помимо этого можно расширять функциональность существующих компонентов, добавляя к ним поддержку технологии AJAX.

Новые шаблоны проектов Web Application

Проект Web Application, появившийся в Microsoft Visual Studio 2008, является альтернативой проекту Web Site в предыдущих версиях Visual Studio. Проектная модель web-приложения полностью поддерживает все возможности Visual Studio 2008 и ASP .NET 2.0.

По умолчанию проектная модель Web Site использует структуру каталогов, которые отражают содержимое проекта. В этой модели нет файлов проекта — все файлы в каталоге являются частью проекта. В проектной модели Web Application только файлы, непосредственно включенные в проект, являются частью этого проекта, и только они отображаются в Solution Explorer и компилируются в процессе сборки проекта. Применение проектного файла в модели Web Application позволяет проще реализовать ряд сценариев. Например, у разработчиков появляется возможность разделить ASP .NET-приложение на несколько проектов в рамках Visual Studio, ссылаться на разные проектные файлы и при необходимости легко исключать файлы из проекта. Проектную модель Web Application следует использовать в следующих случаях:

- миграция больших по объему приложений, созданных средствами Visual Studio .NET 2003, в Visual Studio 2008;
- управление именами сборок, создаваемых при компиляции проекта;
- применение отдельных классов для ссылки на классы страниц и пользовательские компоненты;
- создание web-приложений с помощью нескольких web-проектов;
- добавление шагов, выполняемых перед и после основной компиляцией проекта.

Поддержка JavaScript

Поддержка JavaScript важна для обеспечения создания приложений с использованием таких технологий, как AJAX и Microsoft Silverlight. В Microsoft Visual Studio 2008 реализована полная поддержка написания кода на JavaScript, включая:

- поддержку на уровне технологии Intellisense в редакторе кода;
- полноценную поддержку в средствах отладки приложений.

Поддержка JavaScript на уровне технологии Intellisense позволяет автоматически определять тип переменных (Type inference), обеспечивать корректную работу с внешними библиотеками на JavaScript (файлы с расширением *.js), на которые ссылается текущий код, а также использовать

комментарии, добавляемые в описаниях функций для определения типов (рис. 6).

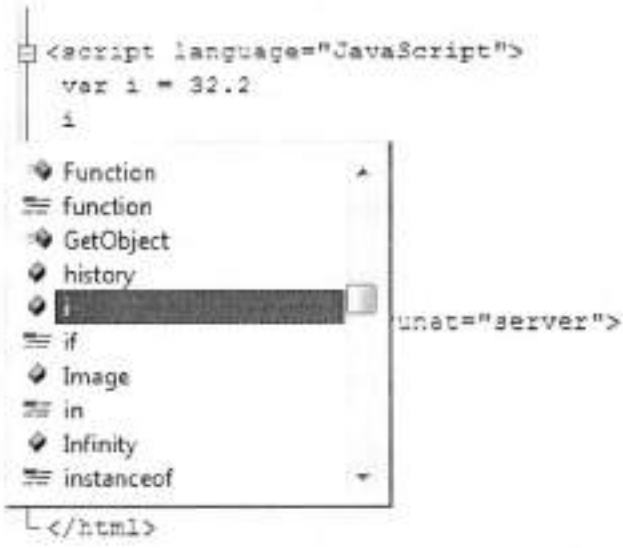


Рис. 6. JavaScript Intellisense

Поддержка отладки JavaScript-кода позволяет устанавливать точки останова непосредственно в файлах с расширениями *1ext11, *.js, *.aspx и *.master — для этого не требуется сначала генерировать клиентский код. Помимо этого поддерживается присоединение к Internet Explorer для отладки кода на JavaScript в рамках любой страницы (HTML, PHP, JSP и т.д.), но для этого требуется корректная конфигурация Internet Explorer: на вкладке *Advanced* панели *Internet Options* (команда *Tools Internet Options*) следует отключить опцию *Disable Script Debugging*, которая по умолчанию включена (рис. 7).

Рис. 7. Панель настроек Internet Explorer

В целом поддержка отладки JavaScript в Visual Studio 2008 обеспечивает возможность использования следующих функций: установка точек остановки по условию (*Conditional Breakpoints*), просмотр значений локальных переменных (*Locals Window*), а также использование окон *Immediate Window* и *Watch Windows*.

Расширенная поддержка HTML/CSS в дизайнерах

Дизайнер HTML-кода в Visual Studio 2008 пополнился возможностью отображения как кода, так и его визуального представления с двунаправленной синхронизацией вносимых изменений — этот режим называется Split-View. Переключение между тремя способами отображения — исходный текст, дизайн и Split-View — стало существенно быстрее.

Средства дизайна CSS теперь единые для Visual Studio и семейства продуктов Expression. В новой версии появилось окно *Manage Styles*, которое позволяет задавать CSS-правила и таблицы стилей для редактируемой страницы. Поддерживается возможность перетаскивания стилей — вместо встроенных стилей их можно сделать внешними файлами. Можно создавать новые стили или редактировать существующие — двойной щелчок мышью по стилю переводит его в режим редактирования. Окно свойств CSS позволяет задавать значения любых свойств стиля и поддерживает режим просмотра, позволяющий определить примененные стили и уровень их вложенности и наследования (рис. 8 и 9).

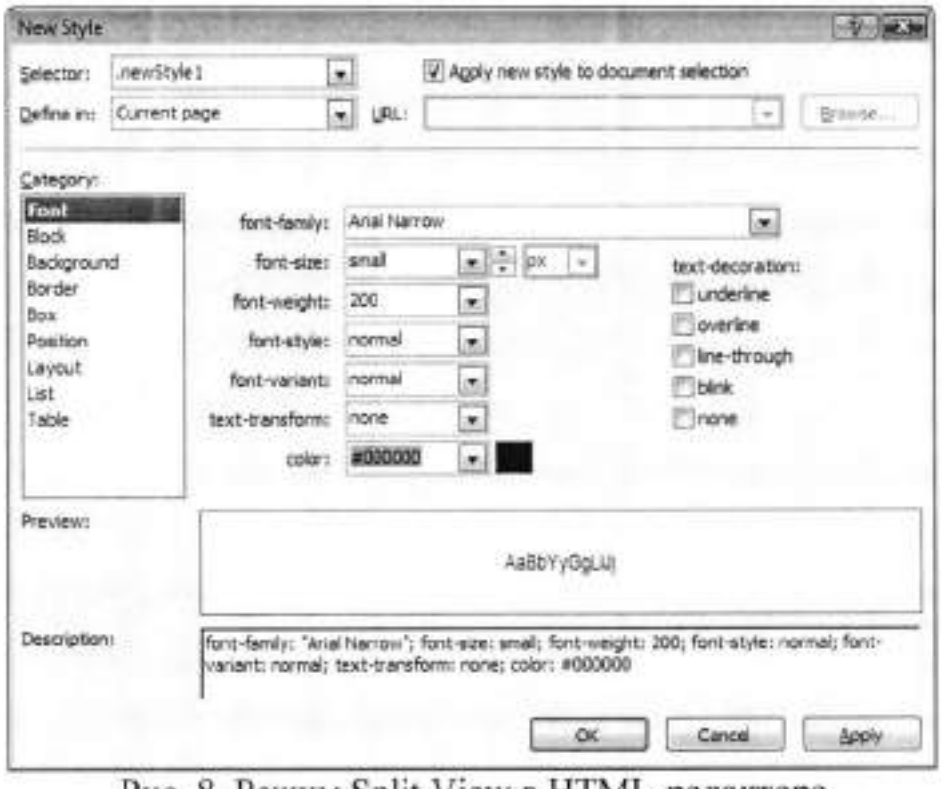


Рис. 8. Режим Split View в HTML-редакторе

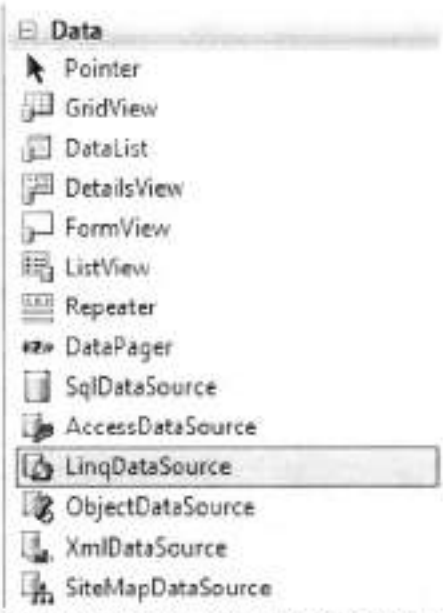


Рис. 9. Редактор стилей в Visual Studio 2008

Новые компоненты для данных

В .NET Framework 3.5 появился ряд новых ASP .NET-компонентов для работы с данными — далее мы кратко рассмотрим эти компоненты (рис. 10).

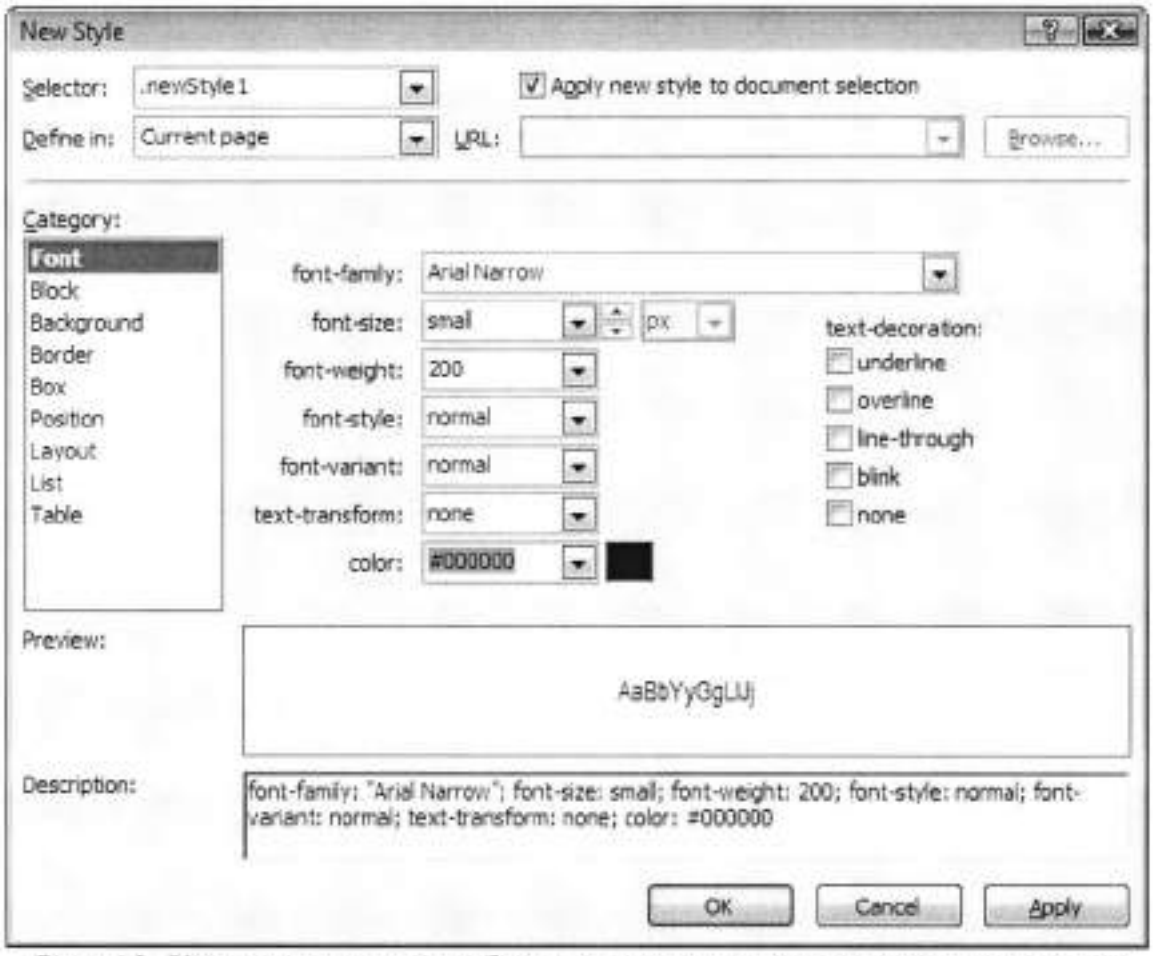


Рис. 10. Компоненты для работы с данными в галерее компонентов

Компонент LinqDataSource

Компонент **LinqDataSource** предназначен для доступа к технологии Language Integrated Query (LINQ) через стандартную архитектуру источников данных ASP .NET. Этот компонент используется на web-страницах, где требуется получение или модификация данных с применением программной модели LINQ. Компонент поддерживает автоматическое создание команд для работы с данными и позволяет существенно сократить код, требуемый для реализации операций с данными по сравнению с использованием таких компонентов, как **SqlDataSource** и **ObjectDataSource**. Помимо этого LinqDataSource позволяет разработчикам применять одну программную модель для доступа к различным типам данных.

Разработчики могут использовать декларативное описание для создания компонента **LinqDataSource**, связанного либо с базой данных, либо со структурой в памяти (например, коллекцией). В этом описании указываются критерии для отображения, фильтрации, сортировки и группировки данных, а также разбиения результатов на страницы. Когда источником данных является таблица в базе данных, можно сконфигурировать компонент таким образом, что он будет обновлять, вставлять и удалять данные — разработчикам не требуется писать SQL-команды для выполнения этих задач. Компонент **LinqDataSource** поддерживает событийную модель, которая дает возможность создавать обработчики событий, возникающих при работе с источниками данных, а также набор свойств, позволяющих управлять внешним видом и поведением компонента.

Компонент ListView

Компонент **ListView** пришел на смену компонентам **DataList** и **Repeater**. Он позволяет, например, генерировать выпадающие списки, таблицы и неотсортированные списки и предназначен для совместной работы с компонентами типа **LinqDataSource** и **DataPager**. Компонент может легко настраиваться, например используя CSS-стили, и связываться практически с любыми элементами (например, с <select>).

Компонент DataPager

Компонент **DataPager** применяется для страничного доступа к данным, отображаемым в компонентах, поддерживающих интерфейс **IPageableItemContainer**, например в компонент **ListView**. Компонент DataPager поддерживает встроенные средства навигации по страницам, указываемые объектом **NumericPagerField**, который позволяет пользователям выбирать страницы по их номеру. Помимо этого можно применять обьект **NextPreviousPagerField**, который позволяет пользователям перемещаться по страницам или переходить на первую либо последнюю страницу. Также можно создавать собственные механизмы навигации — для этого служит объект **TemplatePagerField**.

Утилита ASP .NET Merge

Утилита ASP .NET Merge (**Aspnet_merge.exe**) позволяет комбинировать сборки, создаваемые предкомпилятором ASP .NET (**Aspnet_compiler.exe**), и управлять ими. Используя данную утилиту, можно создавать единые сборки для всего сайта — объединять все сборки в одну, сборки для каждого каталога web-сайта или только для файлов, отвечающих за визуальные элементы сайта — страницы и компоненты.

Содержание работы

1. Ознакомиться с теоретической частью методических указаний.
2. Получить вариант задания от преподавателя.
3. Решить поставленную задачу
4. Представить работающую программу преподавателю.

Лабораторная работа № 25 (2 часа)
Средства отладки Web-приложений

Большинство современных программных систем имеют сложную многоуровневую архитектуру, часть компонентов которой вообще не может работать на машине разработчика, так как требует установки внушительного списка программ или подключения специального оборудования. Многие разработчики сталкиваются с массой неудобств при тестировании своего кода на разных операционных системах. Современный рынок требует от программных продуктов поддержки как можно большего числа ОС, а выпуск работоспособного и качественного ПО без тщательного тестирования на всех поддерживаемых ОС невозможен. Существенно упростить жизнь разработчика при поиске и исправлении ошибок может удаленная отладка. Предположим, мы разрабатываем обычное Windows- или Web-приложение, которое должно работать на ОС Windows 98/Me/NT/2000/XP. Например, под Windows Me наш код ну никак не хочет работать. Мы можем приблизительно определить место, где возникает ошибка, и даже набор условий, при которых она возникает, но этой информации во многих случаях недостаточно. К счастью, существуют программы, позволяющие создавать виртуальные машины.

ДОСТУПНЫЕ СРЕДСТВА УДАЛЕННОЙ ОТЛАДКИ

Используются следующие способы удаленной отладки:

- Отладка через DCOM и Machine Debug Manager. Этот способ может использоваться для отладки всех типов проектов без ограничений.
- Отладка через Remote Debug Monitor. Этот способ пригоден для отладки только неуправляемого C/C++ кода. В VS .NET 2003, в отличие от VS 6.0, к каналу связи TCP/IP добавился канал связи Pipe.

При удаленной отладке запуск отладчика может осуществляться двумя способами. Первый способ – это настроить свойства проекта. Тогда отладчик VS по команде Start Debug будет автоматически запускать процесс на удаленной машине. В этом случае работу процесса можно целиком отследить с момента создания до момента завершения. Второй способ – это подключиться к уже запущенному процессу, поставить Breakpoint и отладить нужные участки кода. Для присоединения отладчика VS к процессу предназначена команда Debug->Processes (рисунок 1). В поле Transport можно выбрать способ взаимодействия с удаленной машиной: TCP/IP, Pipe или Default. Под Default здесь подразумевается DCOM-соединение. Для использования TCP/IP и Pipe необходимо запустить msvcmon на выбранном компьютере. В поле Name задается имя или IP-адрес компьютера. По команде Attach происходит запуск отладчика и подключение к выбранному процессу. При этом заданные свойства удаленной отладки активного проекта игнорируются.

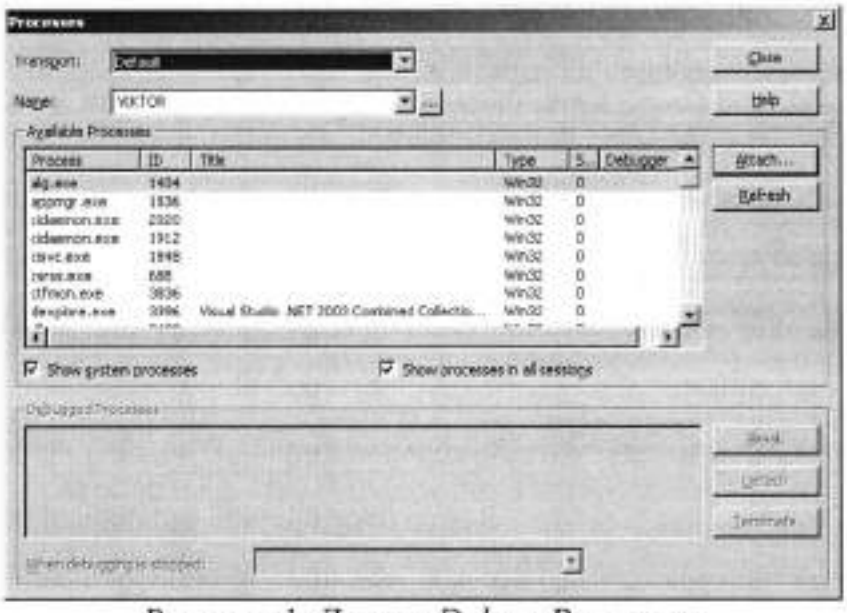


Рисунок 1. Диалог Debug-Processes.

Можно открыть окно Server Explorer, открыть соединение, выбрать БД, хранимую процедуру и запустить ее на выполнение, предварительно задав входные параметры. Выполнять отладку T-SQL можно как на локальном, так и на удаленном компьютере. В случае удаленной отладки взаимодействие осуществляется через тот же Machine Debug Manager через DCOM. В VS .NET 2003 можно выполнять отладку управляемого кода и T-SQL в одной отладочной сессии. При этом среда VS позволяет устанавливать точки останова внутри хранимых процедур и функций. Для отладки T-SQL на удаленный компьютер необходимо установить пакет SQL Debugging Components. Этот пакет входит в Remote Components Setup и в Full Debugging Setup. Пример отладки T-SQL будет рассмотрен далее в этой статье.

Отладка через Remote Debug Monitor

Для отладки через Remote Debug Monitor необходимо установить и вручную запустить на удаленной машине отладочный монитор (Remote Debug Monitor). Remote Debug Monitor – это небольшое консольное приложение msvcmon, которое взаимодействует с отладчиком VS по TCP/IP или каналы (Pipes). Установка отладочного монитора выполняется при установке Remote Components или Full Remote Debugging. Программа установки копирует его в подкаталог <VS.NET 2003>\Common7\Packages\Debugger.

Можно установить Remote Debug Monitor вручную. Для этого необходимо на удаленную машину скопировать следующие файлы:

- msvcmon.exe,
- msvcr71.dll,
- msvcp71.dll,
- natdbgtlnet.dll,
- natdbgdm.dll,

- psapi.dll (только для Windows NT 4.0) ,
- dbghelp.dll (для поддержки dumps) .

Можно также запустить msvcmon из доступного по сети каталога:

По умолчанию msvcmon использует каналы. Параметры сессии msvcmon задаются из командной строки. В таблице 1 приведены основные ключи msvcmon.

Таблица 1. Основные ключи Remote Debug Monitor.

Ключ	Назначение	Примечание
-tcpip	Использовать протокол TCP/IP.	
-register	Регистрация отладочного монитора. Отладочный монитор будет автоматически запускаться при запуске отладчика VS.	
-anyuser	Позволяет выполнять отладку всем пользователям.	Только для TCP/IP
-u domain/username	Учетная запись, которой разрешено подключение.	Только для TCP/IP
-maxsessions	Задает максимальное число одновременных соединений с монитором.	Только для TCP/IP
-timeout n	n – задает таймаут для соединений.	Только для TCP/IP

Отладка через DCOM и Machine Debug Manager

Machine Debug Manager (MDM) – это утилита удаленной отладки, использующая DCOM. MDM работает как служба и может быть установлена на удаленный компьютер следующими способами:

- установить полностью VS.NET 2003 (для самых стойких :));
- установить Remote Components Setup или Full Remote Debugging;
- установить вручную необходимые файлы.

Последние два варианта позволяют выполнять только отладку неуправляемого C/C++ кода. Для отладки управляемого кода и

всевозможных скриптов (VBScript, JScript, T-SQL) необходимо либо целиком установить VS .NET 2003 (только зачем тогда вообще удаленная отладка?), либо установить Full Remote Debugging, что мы и сделаем далее.

При отладке неуправляемого кода Machine Debug Manager автоматически запускает Remote Debug Monitor.

Полигон для испытаний

Опишем состав программных средств, которые будут использоваться для демонстрации.

Это рабочий компьютер, на котором я буду запускать VS и выполнять отладку.. В программе VM Ware Workstation я создал виртуальную машину, на которой буду отлаживаться.

На виртуальной машине я установил только Full Remote Debugging и IIS для возможности отладки ASP.NET Web Application и ASP.NET Web Service. Для тестовых проектов я создал каталог C:\RDebug, в котором будут размещаться исполняемые файлы.

Для установки Full Remote Debugging необходимо запустить программу установки VS.NET 2003 следующим образом:

Примечание

msiexec /qb+ /i *n:\<Visual Studio .NET CD1>*\vs_setup.msi NOVSUI=1 TRANSFORMS="*n:\<Visual Studio .NET CD1>*\Setup\Rmt9x.mst" SERVER_SETUP=1 ADDLOCAL=Full_Remote_Debugging

Настройка системы безопасности

Инфраструктура удаленной отладки накладывает несколько существенных ограничений по безопасности:

1. Пользователь, выполняющий отладку, не может запустить или присоединиться к процессу, который был запущен под другой сессией Terminal Server.
2. Если пользователь, выполняющий отладку, не является администратором, он не может присоединиться к процессу, который был запущен под системной или иной учетной записью.
3. Пользователь, выполняющий отладку, не может запускать процессы на удаленной машине до тех пор, пока он не выполнил вход локально или через Terminal Server.
4. Пользователь, выполняющий отладку, не может запускать процессы на удаленной машине под чужим именем.

5. Пользователь вообще не может осуществлять отладку, если он не является администратором и не входит в группу **Debugger Users**.
6. Члены группы **Debugger Users** могут отлаживать только процессы, запущенные под принадлежащими им учетными записями.
7. Для отладки T-SQL необходимо иметь права на запуск процедуры **sp_sdidebug**.

Ограничения 1, 2 и 3 могут быть сняты с помощью установки ключа (тип DWORD)

[HKLM\Software\Microsoft\Machine Debug Manager\AllowLaunchAsOtherUser]

в значение 1, но, тем не менее, в процессе отладки на удаленной машине обязательно должен быть выполнен вход под учетной записью администратора.

Доступ к Machine Debug Manager настраивается средствами DCOM. По умолчанию в ОС Windows 2000/XP Pro, Windows 2003 Server доступ разрешен группам Administrators (Администраторы в русской версии) и Debugger Users. В ОС Windows 98/95/Me по умолчанию доступ запрещен.

Если зайти на локальную и удаленную машины под разными учетными записями, то из VS нельзя будет осуществить запуск процесса. При попытке это сделать будет выдана ошибка (рисунок 2). Это происходит в связи с ограничением 4. Произвести отладку в этом случае можно только путем ручного запуска и последующим присоединением к процессу. При использовании одной и той же учетной записи можно запускать процессы для отладки на удаленной машине прямо из VS.

Рисунок 2. Ошибка при запуске отладчика, когда на удаленной машине выполнен вход под другой учетной записью.

Отладка неуправляемого C/C++ кода

Начнем поиски правды с отладки простого консольного приложения. Создадим новый Solution RDebug, а в нем новый проект RNative на основе

шаблона Visual C++->Win32 Console Application. Ниже приведен листинг файла RNative.cpp.

```
#include "stdafx.h"
#include <conio.h>
#include <windows.h>

int _tmain(int argc, _TCHAR* argv[])
{
 // получаем имя компьютера
 TCHAR szCompName[MAX_COMPUTERNAME_LENGTH + 1];
 DWORD dwSize = MAX_COMPUTERNAME_LENGTH + 1;
 BOOL bSuccess = GetComputerName(szCompName, &dwSize);
 if (!bSuccess)
 {
  printf("Error getting computer name: %d", GetLastError());
  return -1;
 }
 // формируем сообщение
 TCHAR szMessage[256];
 sprintf(szMessage, "Hello from %s !", szCompName);
 // выводим сообщение
 OutputDebugString(szMessage);
 printf(szMessage);
 getch();

 return 0;
```

Для проверки запустим программу на локальной машине. Убедимся, что появляются сообщения в окне Output и на консоли. Теперь перейдем к настройке проекта для отладки на удаленной машине. Настройки удаленной отладки задаются на закладке Debugging в Project Properties (рисунок 3).

В группе Remote Settings задаются параметры Connection, Remote Machine и Remote Command. В поле Connection - тип соединения: "Remote via DCOM", "Remote via Pipe" или "Remote via TCP/IP". В данном случае будет использоваться "Remote via Pipe". В поле Remote Machine нужно задать имя компьютера, на котором будет производиться отладка. Имя моей виртуальной машины – w2k. В поле Remote Command необходимо ввести путь и имя исполняемого файла на удаленной машине, который будет

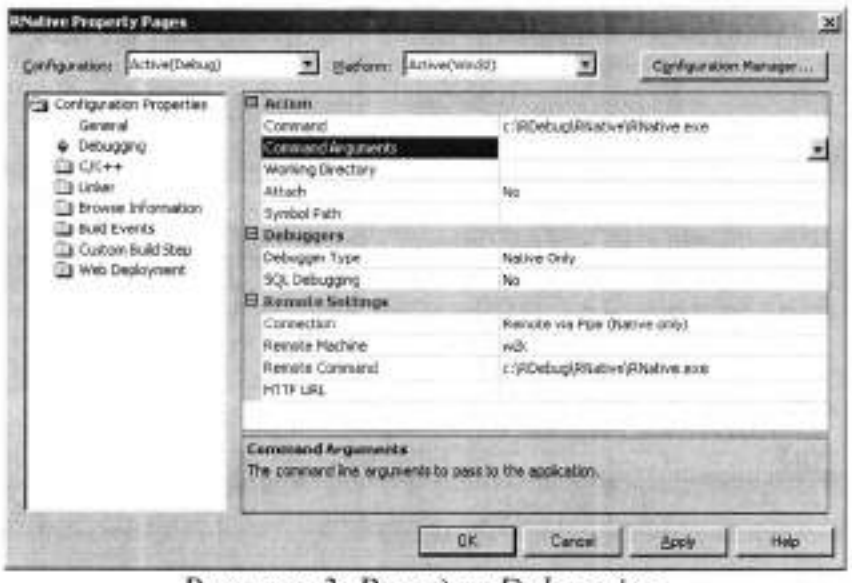


Рисунок 3. Вкладка Debugging.

запускаться при запуске отладчика VS. Путь к файлу RNative.exe на машине w2k – C:\RDebug\RNative\RNative.exe. Для того чтобы исполняемый файл по команде Build помещался в каталог C:\RDebug\RNative\, необходимо параметру Output File на закладке Linker->General присвоить значение \\w2k\RDebug\RNative\RNative.exe. Также необходимо, чтобы файл pdb находился в этом же каталоге, поэтому параметру Linker->Debugging необходимо присвоить значение \\w2k\RDebug\RNative\RNative.pdb.

Проект готов. Приступаем к запуску. На удаленной машине нужно войти в систему под той же учетной записью, что и на локальной, и запустить отладочный монитор msvcmon. Теперь нужно произвести сборку проекта, после чего запустить отладчик. В отладочном окне и на консоли появится сообщение “Hello from W2K!”.

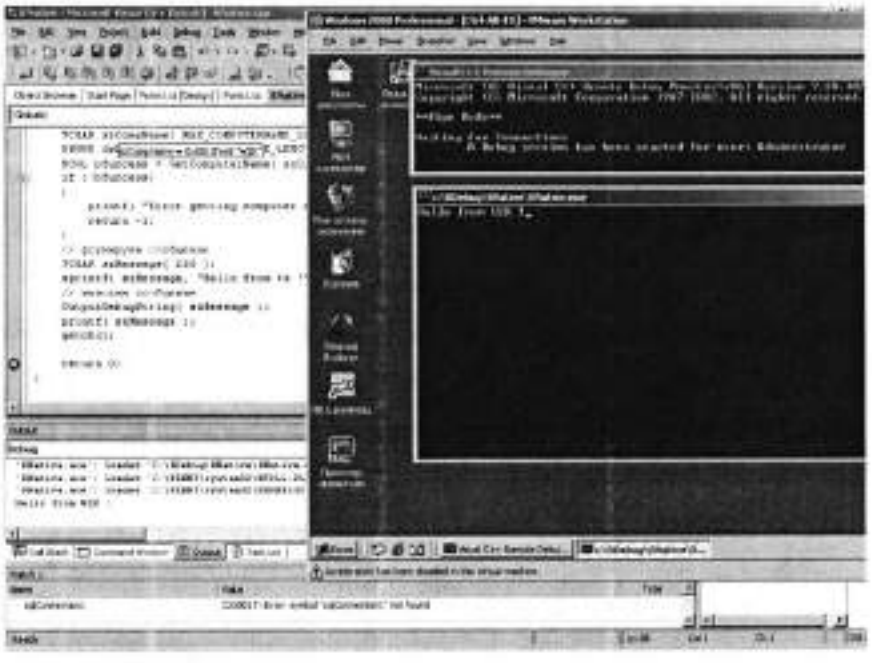


Рисунок 4. Удаленная отладка Win32 Console Application.

При отладке через Pipe Remote Debug Monitor предоставляет возможность задавать разрешения на отладку либо всем пользователям, либо определенному пользователю (см. ключи –u и –anyuser).

Отладка GUI-приложений

Создадим новый Solution RDebug, а в нем новый проект RWinApp из шаблона Visual C#->Windows Application. Затем добавим компонент Perfomance Counter и настроим его на счетчик объема доступной памяти.

Рисунок 5. Проект Windows Application.

Добавим обработчик нажатия кнопки "Debug me":

```
private void button1_Click(object sender, System.EventArgs e)
{
 float val = pcMemory.NextValue();
 string text = string.Format("Aviable memory: {0} Mb", val);
 Console.WriteLine(text);
 MessageBox.Show(text);
 System.Diagnostics.Trace.WriteLine(text);
 System.Diagnostics.Trace.Assert(false);
}
```

Итак, приложение готово. Теперь необходимо подготовить его к удаленной отладке. Для этого нужно открыть Project Properties и выбрать ветку **Debugging** (рисунок 6).

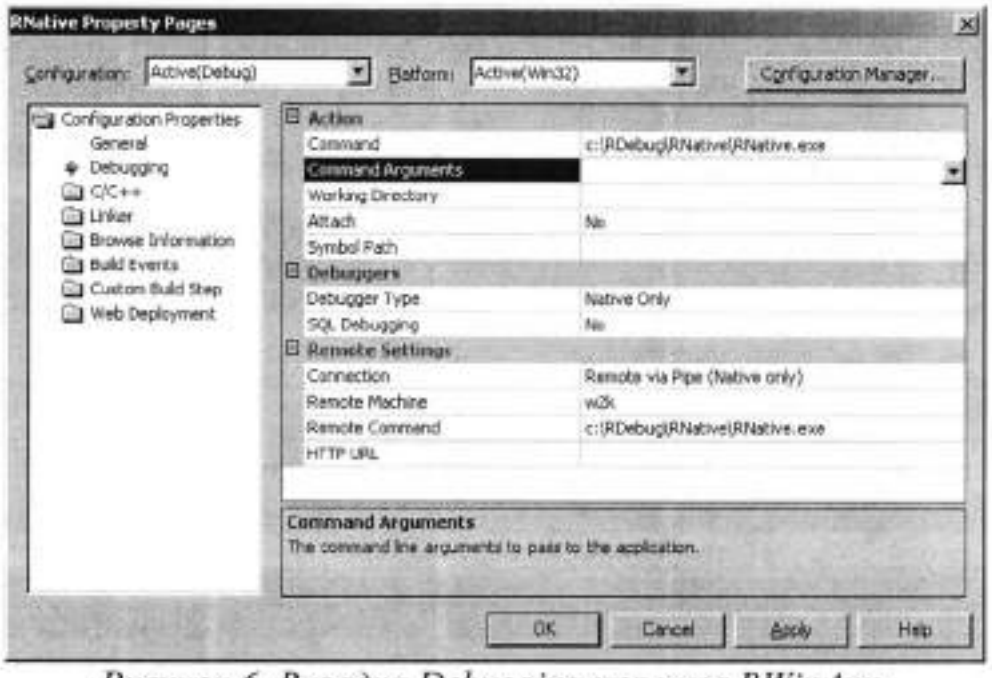


Рисунок 6. Вкладка Debugging проекта RWinApp.

Для удаленной отладки нужно параметр **Enable Remote Debugging** установить в **True**, а параметру **Remote Debug Machine** присвоить имя удаленной машины. Также необходимо изменить параметры запуска приложения: параметру **Debug Mode** присвоить значение **Program**, а параметру **Start Application** – полный путь к исполняемому файлу на удаленной машине. На машине w2k полный путь к программе RWinApp - C:\RDebug\RWinApp\RWinApp.exe. Для того чтобы после каждой сборки не копировать файлы RWinApp.exe и RWinApp.pdb, можно изменить параметр **Output Path** в ветке Build. На машине w2k я открыл доступ к каталогу RDebug, а параметру Output Path присвоил значение \\w2k\RDebug\RWinApp\. Проект Windows Application на VB .NET настраивается аналогично.

Проект готов к удаленной отладке. Можно запустить программу. Console.WriteLine при этом ничего не пишет (рисунке 7), System.Diagnostics.Trace.WriteLine выдал текст в окно Output, а System.Diagnostics.Trace.Assert дал о себе знать везде.

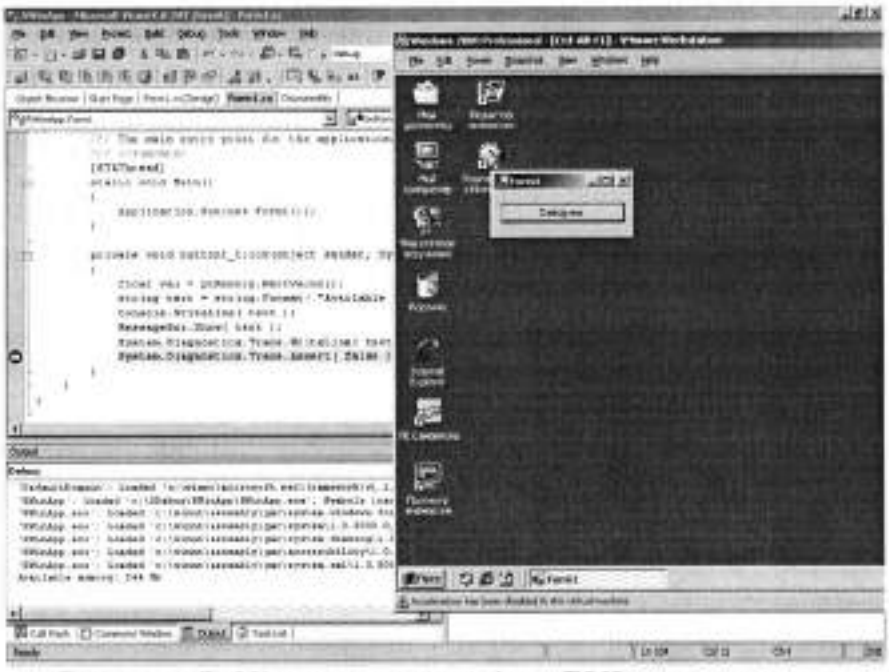


Рисунок 7. Удаленная отладка GUI-приложения.

Отладка Web-приложения ASP.NET

Теперь попробуем отладить Web-приложение на удаленной машине. Здесь все гораздо легче. При запуске Web-приложения отладчик выполняет auto-attach, поэтому входить на локальную и удаленную машины под одной учетной записью вовсе не обязательно. Однако пользователь, выполняющий отладку, должен быть администратором на удаленной машине. Как дать возможность удаленной отладки ASP.NET-приложений пользователям из Debugger Users, описано в статье "ASP Remote Debugging Setup" в MSDN.

Приступим к созданию простого Web-приложения. Добавим в Solution новый проект из шаблона Visual C#->ASP.NET Web Application. В поле **Location** укажем **http://w2k/RWebApp**.

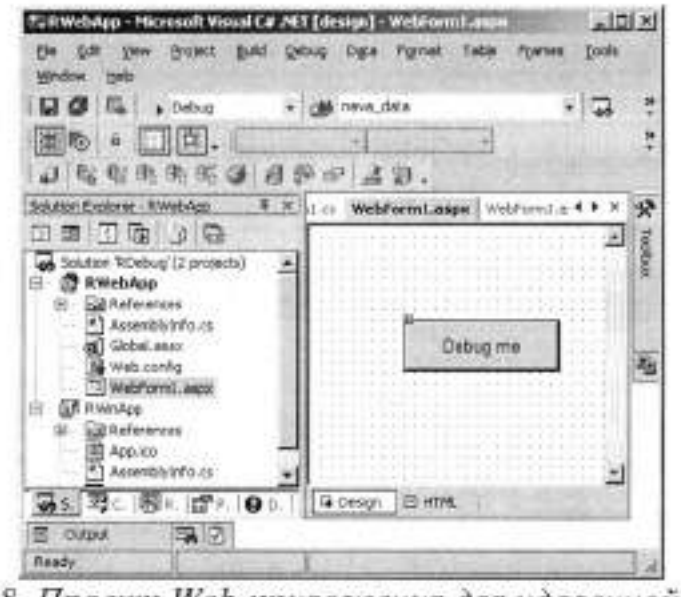


Рисунок 8. Проект Web-приложения для удаленной отладки

Добавим к форме любимую кнопку “Debug me” и в обработчике нажатия напишем:

```
private void Button1_Click(object sender, System.EventArgs e)
{
 string text = string.Format("Hello from {0}", Server.MachineName);
 System.Diagnostics.Trace.WriteLine(text);
 System.Diagnostics.Trace.Assert(false);
}
```

В случае ASP.NET-приложений задание каких-то свойств проекта для выполнения удаленной отладки не требуется. Запускаем приложение. При нажатии на кнопку в окне Output появится сообщение “Hello from W2K!” и информация о месте возникновения Assert.

Рисунок 9. Удаленная отладка Web-приложения ASP.NET.

Отладка T-SQL

Как уже было сказано выше, VS .NET 2003 позволяет выполнять одновременно отладку T-SQL и кода приложения в одной отладочной сессии. Приложение, вызывающее хранимые процедуры или функции, может работать локально или удаленно, при этом местонахождение SQL-сервера не играет особой роли. Для отладки T-SQL на сервере необходима установка пакета SQL Debugging Components. Отладчик T-SQL имеет следующие ограничения:

- Нельзя изменять последовательность выполнения инструкций T-SQL.
- Нельзя на лету изменять исполняемый код и продолжать выполнение.
- Нельзя переходить командой Step Into из T-SQL в управляемый или неуправляемый код и обратно.
- Нельзя прервать выполнение T-SQL-инструкции.

- Нельзя производить отладку триггеров напрямую. Триггер должен быть вызван. Если триггер вызывается хранимой процедурой, то по команде Step Into можно шагнуть внутрь триггера.

Взаимодействие сервера и клиента при отладке осуществляется через DCOM. В связи с этим задание прав на отладку может осуществляться средствами DCOM и SQL Server. Отладка T-SQL производится с помощью расширенной хранимой процедуры sp_sdidebug. Соответственно, пользователь, выполняющий отладку, должен иметь право запуска этой процедуры. Если отладка T-SQL осуществляется удаленно, то необходимо, чтобы служба MSSQLSERVER работала не под учетной записью LocalSystem. Если при работе отладчика на сервере возникают какие-то ошибки, они фиксируются в журнале приложений.

Перейдем к рассмотрению типовых задач отладки T-SQL. Для начала рассмотрим самую простую задачу – отладку хранимых процедур из VS. Для этого нужно открыть Server Explorer и добавить новое соединение в Data Connections, затем выбрать Step Into Stored Procedure. Я в качестве примера выбрал процедуру SalesByCategory из БД Northwind. Перед запуском отладчик запрашивает необходимые параметры. Для процедуры SalesByCategory я ввел @CategoryName='Confections', @OrdYear=<DEFAULT>. После запуска происходит остановка на первой же инструкции (рисунок 10).

Если нужно отладить T-SQL, вызываемый из программы, можно поставить в нем точку останова. Для иллюстрации этого откроем проект RWebApp, в нем форму WebForm1.aspx. Перетащим на нее хранимую процедуру SalesByCategory. В результате на форме появятся объекты sqlConnection1 и sqlCommand1.

Содержание работы

1. Ознакомиться с теоретической частью методических указаний.
2. Выполнить примеры
3. Получить вариант задания от преподавателя.
4. Решить поставленную задачу
5. Представить работающую программу преподавателю.

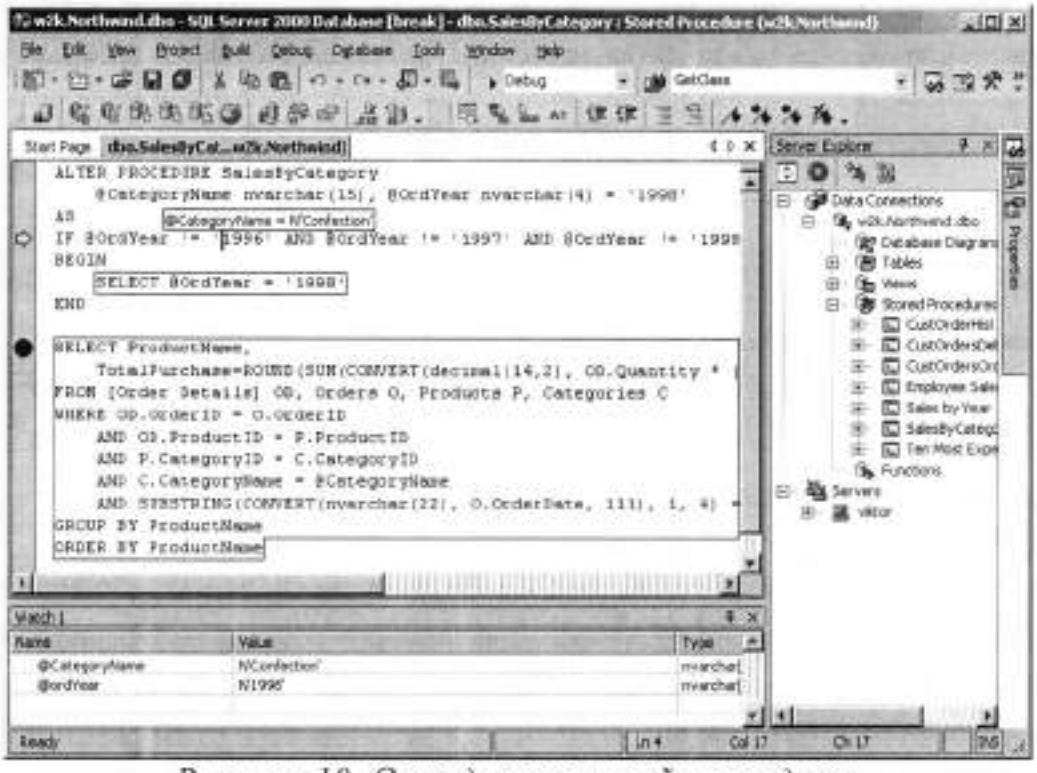


Рисунок 10. Отладка хранимой процедуры.

После этого можно в обработчик Page_Load добавить следующий код:

```
private void Page_Load(object sender, System.EventArgs e)
{
 // Put user code to initialize the page here
 sqlConnection1.Open();
}
```

Теперь добавим на форму еще одну кнопку “Debug SQL” и в обработчике ее нажатия напишем:

```
private void Button2_Click(object sender, System.EventArgs e)
{
 try
 {
  sqlCommand1.Parameters["@CategoryName"].Value = "Confections";
  int retCode = sqlCommand1.ExecuteNonQuery();
 }
 catch (System.Data.SqlClient.SqlException ex)
 {
  System.Diagnostics.Trace.WriteLine(ex.Message);
 }
}
```

После этого соберем проект, поставим точку останова в процедуре SalesByCategory и запустим приложение. По нажатию на кнопку “Debug SQL” происходит остановка на этой точке останова.

Лабораторная работа № 26 (2 часа)
Использование Web сервисов XML в консольных приложениях и приложениях Windows Forms

Отладка и трассировка страниц ASP.NET

Вообще говоря, при создании Web-проекта ASP.NET вы можете использовать те же средства отладки, что и при создании любого другого проекта в Visual Studio 2005. Так, вы можете устанавливать контрольные точки в файле внешнего кода поддержки (и в блоках script файла *.aspx), запускать сеанс отладки (по умолчанию для этого используется клавиша ‹F5›) и использовать режим пошагового выполнения программного кода. Но, чтобы выполнять отладку Web-приложения ASP.NET, ваш узел должен содержать правильно скомпонованный файл Web.config. В главе 24 структура файлов Web.config рассматривается подробнее, но, по существу, эти XML-файлы служат той же цели, что и файл app.config выполняемого компоновочного блока. Если ваш проект еще не содержит файла Web.config, Visual Studio 2005 это обнаружит и добавит такой файл в ваш проект. Соответствующим элементом является ‹compilation›.

```
‹configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0"›
 ...
 ‹system.web›
  ‹compilation debug="true"/›
 ‹/system.web›
‹/configuration›
```

Вы также можете разрешить поддержку трассировки для файла *.aspx, установив для атрибута Trace значение true (истина) в рамках директивы ‹%@Page%›.

```
‹%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" Trace="true" %›
```

В результате генерируемый HTML-код будет содержать многочисленные подробности, касающиеся предыдущего цикла запроса/ответа HTTP (переменные сервера, сеанса, приложения и т.д.). Чтобы добавить к ним свои собственные сообщения трассировки, можете использовать свойство Trace типа System.Web.UI.Page. В любое время, когда вы пожелаете записать пользовательское сообщение (из блока сценария или файла исходного вода C#), просто вызовите метод Write().

```
protected void btnFillGrid_Click(object Sender, EventArgs e) {
 ...
 // Генерирование пользовательского сообщения трассировки.
 Trace.Write("Моя категория", "Конец заполнения таблицы");
}
```

Если теперь запустить проект и направить вторичное обращение Web-серверу, вы увидите свою пользовательскую категорию и пользовательское сообщение ближе к концу раздела трассировки перед разделом Control Tree:

Рис. 1. Запись пользовательских сообщений трассировки

Настройка Web-приложения ASP.NET с помощью Web.config

При изучении компоновочных блоков .NET мы с вами выяснили, что приложения клиента могут использовать XML-файл конфигурации, содержащий инструкции CLR о том, как обрабатывать связанные запросы, где искать необходимые компоновочные блоки и что еще нужно учесть в среде выполнения. То же можно сказать и в случае Web-приложений ASP.NET, но в данном случае файлы конфигурации (впервые упомянутые в главе 23) всегда называются Web.config (в отличие от файлов конфигурации *.exe, имена которых зависят от имен соответствующих выполняемых файлов клиента).

При добавлении файла Web.config к файлам узла с помощью выбора WebSite>Add New Item из меню создаваемая по умолчанию структура выглядит примерно так, как показано ниже (чтобы не загромождать структуру, комментарии здесь были исключены).

```
‹?xml version="1.0"?›
 ‹configuration
xmlns="http://schemas.microsoft.com/.NetConfiguration/v
2.0"›
  ‹appSettings/›
  ‹connectionStrings/›
  ‹system.web›
   ‹compilation debug="false"/›
   ‹authentication mode="Windows"/›
  ‹/system.web›
‹/configuration›
```

Подобно любому файлу *.config, в файле Web.config определяется корневой элемент ‹configuration›. В его контекст вкладывается элемент ‹system.web›, который может содержать множество дочерних элементов, с помощью

которых осуществляется управление поведением Web-приложения в среде выполнения. В ASP.NET файл Web.config можно модифицировать с помощью любого текстового редактора. Некоторые элементы, которым позволено присутствовать в файле Web.config, описаны в таблице 1.

Замечание. Чтобы выяснить подробности формата файла Web.config, выполните поиск разделов документации .NET Framework 2.0 SDK, соответствующих ключу поиска "ASP.NET Settings Schema".

Таблица 1. Подборка элементов файла

Элемент	Описание
‹appSettings›	Используется для создания пользовательских пар имен и значений, которые можно программно считывать в память для использования на страницах в дальнейшем
‹authentication›	Связанный с безопасностью элемент, используемый для определения режима аутентификации данного Web-приложения
‹authorization›	Еще один связанный с безопасностью элемент, используемый для определения прав пользователей при доступе к ресурсам сервера
‹compilation›	Используется для разрешения (или запрета) отладки и определения языка .NET, используемого данным Web-приложением по умолчанию, а также (необязательно) для определения множества внешних компоновочных блоков .NET ссылки на которые должны использоваться автоматически
<connectionStrings>	Используется для хранения строк внешних соединений данного Web-узла
‹customErrors›	Используется для инструкций среде выполнения по поводу того, как сообщать об ошибках, происходящих в процессе работы Web-приложения
‹globalization›	Используется для настройки параметров глобализации данного Web-приложения
‹sessionState›	Используется для контроля того, как и где среда выполнения .NET должна хранить данные состояния сеанса
<trace>	Используется для разрешения (или отключения) трассировки данного Web-приложения

Файл Web.config может содержать дополнительные элементы, размещенные как до, так и после элементов, представленных в табл. Большинство этих элементов связано с безопасностью, а остальные оказываются полезными только для построении достаточно сложных сценариев ASP.NET, предполагающих, например, создание пользовательских HTTP-заголовков или пользовательских HTTP-модулей (эти вопросы здесь обсуждать не планируется). Если вам нужен полный комплект элементов, допустимых для

использования в файле Web.config, поищите по ключу "ASP.NET Settings Schema" в системе оперативной справки.

Разрешение трассировки с помощью ‹trace›

Первым элементом файла Web.config, который мы собираемся здесь рассмотреть, будет элемент ‹trace›. Этот XML-дескриптор может иметь любое число атрибутов, задающих особенности его поведения, как показано в следующем примере.

```
‹trace
 enabled="true|false"
 localOnly= "true|false"
 pageOutput="true|false"
 requestLimit="integer"
 traceMode="SortByTima|SortByCategory"/›
```

Описания этих атрибутов предлагаются в табл. 2.

Таблица 2. Атрибуты элемента ‹trace›

Атрибут	Описание
enabled	Индикатор разрешения трассировки для приложения в целом (значением по умолчанию является false – ложь). Как было показано в предыдущей главе, можно разрешить трассировку селективно для данного файла *.aspx, используя директиву @Page
localOnly	Индикатор отображения информации трассировки только на Web-сервере, но не в системах удаленных клиентов (значением по умолчанию является true – истина)
pageOutput	Указывает вид представления результатов трассировки
requestLimit	Указывает число запросов трассировки, сохраняемых на сервере. Значением по умолчанию является 10. Если достигается предел, трассировка автоматически отключается
traceMode	Указывает соответствие порядка отображения информации трассировки порядку ее обработки. Значением по умолчанию является SortByTime (сортировка по времени), но можно также указать сортировку по категории

Вспомните из предыдущей главы, что с помощью директивы ‹%@Page%› можно разрешить трассировку отдельных страниц. Но если вы хотите разрешить трассировку для всех страниц Web-приложения, измените ‹trace› в файле Web.config, как показано ниже.

```
‹trace enabled="true" requestLimit="10" pageOutput="false"
traceMode="SortByTime" localOnly="true" /›
```

Настройка вывода сообщений об ошибках с помощью ‹customErrors›

Элемент ‹customErrors› может использоваться для автоматического перенаправления всех ошибок в пользовательский набор файлов *.htm. Это может оказаться полезным тогда, когда вы хотите построить более понятную для пользователя страницу информирования об ошибках по сравнению с той,

которая предлагается средой CLR по умолчанию. В общем виде элемент ‹customErrors› выглядит так.

```
‹customErrors defaultRedirect="url" mode="On|Off|RemoteOnly"›
 ‹error statusCode="код_состояния" redirect="url"/›
‹/customErrors›
```

Чтобы предъявить пример применения элемента ‹customErrors›, предположим, что Web-приложение ASP.NET имеет два файла *.htm. Первый файл (genericError.htm) функционирует, как страница перехвата ошибок. Эта страница может содержать логотип вашей компании, ссылку на адрес электронной почты администратора системы и, например, сообщение с извинениями за доставленные пользователю неудобства. Второй файл (Error404.htm) – это пользовательская страница ошибки, которая должна появляться только тогда, когда среда выполнения обнаруживает ошибку с номером 404 (грозная ошибка "необнаруженного ресурса"). Если вы хотите, чтобы все ошибки обрабатывались этими пользовательскими страницами, вы можете изменить файл Web.config так, как предлагается ниже.

```
‹?xml version="1.0"?›
‹configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0"›
 ‹appSettings/›
 ‹connectionStrings /›
 ‹system.web›
  ‹compilation debug= "false" /›
  ‹authentication mode="Windows"/›
  ‹customErrors defaultRedirect="genericError.htm" mode="On"›
   ‹error statusCodes="404" redirect="Error404.htm"/›
  ‹/customErrors›
 ‹/system.web›
‹/configuration›
```

Обратите внимание на то, что корневой элемент ‹customErrors› используется для указания имени общей страницы для всех необработанных ошибок. Одним из атрибутов, которые могут присутствовать в открывающем дескрипторе, является атрибут mode. Значением, устанавливаемым для этого атрибута по умолчанию, является RemoteOnly, дающее указание среде выполнения *не отображать* пользовательские страницы ошибок, если HTTP-запрос поступает с той же машины, где находится Web-сервер (это очень удобно для разработчиков, которым требуется видеть все подробности). Если установить для атрибута mode значение "on", это позволит видеть пользовательские ошибки на всех машинах (включая машину разработки). Также заметьте, что элемент ‹customErrors› может поддерживать любое число вложенных элементов ‹error›, с помощью которых можно указать, какая страница должна использоваться для ошибки с тем или иным кодом.

Чтобы проверить работу пользовательского перенаправления ошибок, создайте страницу *.aspx с двумя элементами управления Button и обработайте их события Click так, как предлагается ниже.

```
private void btnGeneralError_Click(object sender, EventArgs e) {
 // Это генерирует ошибку общего вида.
 throw new Exception("Ошибка общего вида…");
}
private void btn404Error_Click(object sender, EventArgs e) {
 // Это генерирует ошибку 404 (при отсутствии файла MyPage.aspx).
 Response.Redirect("MyPage.aspx");
}
```

Сохранение данных состояния с помощью ‹sessionState›

Наиболее мощным элементом файла Web.config является ‹sessionState›. По умолчанию ASP.NET запоминает данные сеансового состояния с помощью *.dll в рамках рабочего процесса ASP.NET (aspnet_wp.exe). Подобно любому файлу *.dll. положительным моментом его использования является то, что доступ к информации оказывается настолько быстрым, насколько это возможно. Однако недостатком оказывается то, что в случае аварийного завершения работы этого домена приложения (по любой причине) будут потеряны все данные пользователя. К тому же, когда вы храните данные в *.dll внутри процесса, вы не можете взаимодействовать с сетевой Web-группой. По умолчанию элемент ‹sessionState› файла Web.config выглядит примерно так.

```
‹sessionState
mode="InProc" stateConnectionString="tcpip=127.0.0.1:42424" sqlConnectionS
tring="data source=127.0.0.1;Trusted_Connection=yes" cookieless="false"
timeout="20" /›
```

Принятый по умолчанию режим хранения оказывается вполне подходящим только в том случае, когда ваше приложение обслуживается в рамках одного Web-сервера. Но в ASP.NET вы можете дать указание среде выполнения обслуживать *.dll сеансового состояния в суррогатном процессе, называемом сервером сеансового состояния ASP.NET (aspnet_state.exe). Тем самым вы можете вывести *.dll из aspnet_wp.exe в отдельный *.exe. Первым шагом при этом должен быть запуск службы aspnet_state.exe Windows. С этой целью введите в командной строке

```
net start aspnet_state
```

Запустить aspnet_state.exe можно и по-другому, с помощью оснастки Службы, доступной из папки Администрирование панели управления Windows (рис. 2.).

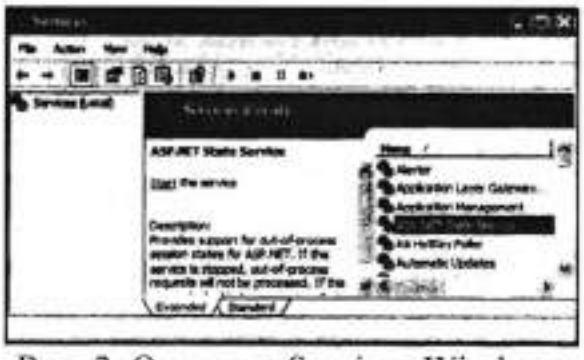


Рис. 2. Оснастка Services Windows

Преимуществом этого подхода является то, что с помощью окна свойств вы можете настроить aspnet_state.exe на автоматический старт при загрузке машины. В любом случае, запустив сервер состояния сеанса, измените элемент ‹sessionState› в файле Web.config так, как показано ниже.

‹sessionState **mode="StateServer"** stateConnectionString="tcpip=127.0.0.1:42424" sqlConnectionString="data source=127.0.0.1;Trusted_Connection=yes" cookieless="false" timeout="20"/›

Здесь атрибут mode устанавливается равным StateServer. Это именно то, что требуется. Теперь CLR обслуживает данные сеанса в рамках aspnet_state.exe. В этом случае, если домен приложения, содержащий само Web-приложение, завершится аварийно, сеансовые данные сохранятся. Также заметьте, что элемент ‹sessionState› может содержать атрибут stateConnectionString. Принятое по умолчанию значение (127.0.0.1) для адреса TCP/IP указывает на локальную машину. Если вместо этого вы хотите, чтобы среда выполнения .NET использовала сервис aspnet_state.exe на другой машине в сети (снова подумайте о Web-группе), вы имеете возможность изменить это значение.

Наконец, если требуется наивысшая степень изолированности и устойчивости дли Web-приложения, вы можете "заставить" среду выполнения сохранять все данные состояния сеанса в Microsoft SQL Server. Соответствующая модификация файла Web.config снова очень проста.

‹sessionState **mode="SQLServer"** stateConnectionString="tcpip=l27.0.0.1:42424" sqlConnectionString="data sourse=127.0.0.1;Trusted_Connection=yes" cookieless="false" timeout="20" /›

Однако перед тем как вы попытаетесь выполнять соответствующее Web-приложение, вы должны обеспечить правильную настройку целевой машины (указанной атрибутом sqlConnectionString). При установке .NET Framework 2.0 SDK (или Visual Studio 2005) создаются два файла, InstallSqlState.sql и UninstallSqlScate.sql, которые по умолчанию помещаются в папку ‹%windir%›\Microsoft.NET\Framework\‹версия›. На целевой машине вы должны выполнить файл InstallSqlState.sql, используя, например, SQL Server Query Analyzer (который входит в поставку Microsoft SQL Server).

После выполнения указанного SQL-сценария, вы обнаружите новую базу данных SQL Server (с именем ASPState), содержащую набор хранимых процедур, вызываемых средой выполнения ASP.NET, и множество таблиц, используемых для хранения сеансовых данных (кроме того, в базу данных tempdb будет добавлено множество таблиц для обмена данными). Вы должны понимать, что настройка Web-приложения на запоминание сеансовых данных рамках SQL Server является самым медленным из всех возможных вариантов. Преимущество этого варианта в том, что пользовательские данные при этом сохраняются наиболее надежно (даже при перезапуске Web-сервера).

Замечание. Если для хранения сеансовых данных используется сервер состояния сеанса ASP.NET или SQL Server, то любой пользовательский тип,

размещаемый в объекте HttpSessionState, должен быть обозначен атрибутом [Serializable].

Тестирование Web-сервиса XML с помощью WebDev.WebServer.exe

Напомним (снова см. главу 23), что WebDev.WebServer.exe является сервером Web-разработки ASP.NET, поставляемым в составе дистрибутива .NET Framework 2.0 SDK. И хотя WebDev.WebServer.exe не предполагается использовать для обслуживания Web-сервисов XML производственного уровня, этот инструмент позволяет запустить Web-содержимое непосредственно из локального каталога при отладке. Для проверки своего сервиса с помощью этого инструмента откройте окно командной строки Visual Studio 2005 и выполните следующую команду, указав свободный номер порта и физический путь к каталогу, содержащему ваш файл *.asmx.

WebDev.Webserver /port:4000 /path:"C:\HelloWorldWebService"

После запуска Web-сервера откройте любой браузер и укажите в его окне имя своего файла *.asmx, используя соответствующий номер порта.

http://localhost:4000/HelloWorldWebService.asmx

Вам будет показан список всех Web-методов, доступных по этому адресу URL (рис. 3).

Рис. 3. Тестирование Web-сервиса XML

Если в окне браузера вы щелкнете на ссылке HelloWorld, откроется другая страница, которая позволит вызвать [WebMethod], только что выбранный вами. В результате вызова HelloWorld() будет возвращена не буквальная строка .NET System.String, а XML-представление текстовых данных, возвращаемых Web-методом HelloWorld().

‹?xml version="1.0" encoding="utf-8"?›

‹string xmlns="bttp://tempuri.org/"›**Hello!**‹/string›

Тестирование Web-сервиса XML с помощью IIS

Теперь, когда вы проверили свой Web-сервис XML с помощью WebDev. WebServer.exe, перенесите файл *.asmx в виртуальный каталог IIS. Используя инструкции, предложенные в главе 23, создайте новый виртуальный каталог с именем HelloWS, который будет отображаться в физическую папку, содержащую файл HelloWorldWebService.asmx. После этого вы получите возможность проверить свой Web-сервис с помощью ввода следующего значения URL в строке Web-браузера.

Просмотр WSDL-документа

Как уже упоминалось, WSDL является метаязыком, описывающим многочисленные особенности Web-методов, доступных по данному адресу URL. Обратите внимание на то, что при проверке Web-сервиса XML автоматически генерируемая страница тестирования предлагает ссылку Service Description (Описание сервиса). В результате щелчка на этой ссылке к текущему запросу присоединяются символы ?wsdl. Когда среда выполнения ASP.NET получает запрос для файла *.asmx с таким прикрепленным суффиксом, она автоматически возвращает соответствующий WSDL-код, открывающий каждый доступный Web-метод.

В настоящий момент вам не следует беспокоиться о природе WSDL-кода или формате WSDL-документа. Пока что важно только понимать, что WSDL-код описывает то, как Web-методы могут вызываться с помощью имеющихся протоколов связи Web-сервиса XML.

Автоматически генерируемая страница тестирования

Работоспособность Web-сервисов XML можно проверить с помощью автоматически генерируемой HTML-страницы в браузере. Когда обнаруживается HTTP-запрос, указывающий на данный файл *.asmx, среда выполнения ASP.NET использует файл с именем DefaultWsdlHelpGenerator.aspx, чтобы создать HTML-страницу, позволяющую вызвать Web-методы, доступные по данному URL. Этот файл *.aspx можно найти в следующем каталоге (здесь, конечно, блок ‹версия› следует заменить на номер вашей текущей версии .NET Framework).

C:\Windows\Microsoft.NET\Framework\‹версия›\CONFIG

Создание пользовательской страницы тестирования

Если вы хотите, чтобы среда выполнения ASP.NET применяла пользовательский файл *.aspx для проверки ваших Web-сервисов XML, вы можете встроить в эту страницу дополнительную информацию (например, фирменный знак компании, дополнительные описания сервиса, ссылки на файлы справки и т.д.). Чтобы упростить себе задачу, большинство разработчиков сначала копируют существующий файл DefaultWsdlHelpGenerator.aspx в проект, а затем, используя этот файл в качестве исходного, нужным образом изменяют оригинальный HTML-документ и программный код C#.

Скопируйте файл DefaultWsdlHelpGenerator.aspx в каталог, содержащий HelloWorldWebService.asmx (например, C:\HelloWorldWebService). Переименуйте полученную копию в MyCustomWsdlHelpGenerator.aspx и измените какой-нибудь фрагмент HTML-кода, скажем, в области дескриптора ‹title›. Например, измените имеющийся код разметки

‹title›‹%#ServiceName + " " + GetLocalizedText("WebService")%›‹/title›

на следующий.

‹title›**Мой собственный**
‹%#ServiceName + " " + GetLocalizedText("WebService") %›
‹/title›

После изменения HTML-содержимого создайте файл Web.config и сохраните его в текущем каталоге. Следующие XML-элементы дают указание среде выполнения использовать ваш пользовательский файл *.aspx, а не DefaultWsdlHelpGenerator.aspx.

```
<!--Здесь указывается пользовательский файл *.aspx -->
<configuration>
 <system.web>
  <webServices>
   <wsdlHelpGenerator href="MyCustomWsdlHelpGenerator.aspx" />
  </webServices>
 </system.web>
</configuration>
```

При запросе своего Web-сервиса вы увидите, что строка заголовка браузера изменится в соответствии с указанным вами пользовательским содержимым. Кстати, если вы захотите отключить генерирование страницы помощи для данного Web-сервиса, вы можете сделать это с помощью элемента <remove> в файле Web.config.

```
<!-- Отмена генерирования страницы помощи -->
<configuration>
 <system.web>
  <webServices>
   <protocols>
    <!-- Этот элемент отменяет генерирование WSDL-документа -->
    <remove name="Documentation"/>
   </protocols>
  </webServices>
 </system.web>
</configuration>
```

Создание Web-сервиса XML

Создав Web-сервис XML вручную, выбрав File>New>Web Site из меню, создайте новый C#-проект Web-сервиса XML с именем MagicEightBallWebService и сохраните этот проект на своем локальном диске (рис.4).

Рис. 4. Проект Web-сервиса XML в Visual Studio

После щелчка на кнопке OK в окне создания сервиса будет сгенерирован файл Service.asmx, определяющий следующую директиву ‹%@WebService%›.

```
‹%@ WebService Language="C#" CodeBehind="~/App_Code/Service.cs"
Class="Service" %›
```

Обратите внимание на то, что здесь используется атрибут CodeBehind, чтобы указать имя файла с программным кодом C#, определяющим соответствующий тип класса (этот файл по умолчанию размещается в каталоге App_Code проекта). По умолчанию Service.cs определяется так.

```
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
[WebService(Namespace="http://tempuri.org/")]
[WebServiceBinding(ConformsTo=WsiProfiles.BasicProfile1_1)]
public class Service: System.Web.Services.WebService {
 public Service() { }
 public string HelloWorld() {
  return "Hello World";
 }
}
```

Здесь класс Service получается из базового класса System.Web.Services.WebService. Члены, определенные этим типом, будут рассмотрены чуть позже, а здесь достаточно подчеркнуть, что получать класс Service именно из этого базового класса совсем не обязательно.

Также обратите внимание на то, что класс Service имеет два (также необязательных) атрибута, [WebService] и [WebServiceBinding]. Роль этих атрибутов тоже будет рассмотрена немного позже.

Содержание работы

1. Ознакомиться с теоретической частью методических указаний.
2. Выполнить примеры
3. Получить вариант задания от преподавателя.
4. Решить поставленную задачу
5. Представить работающую программу преподавателю.

Лабораторная работа № 27 (2 часа)
Использование простейших графических объектов

В приведенном ниже примере создаются объект Graphics и контейнер внутри этого объекта Graphics. Объемным преобразованием объекта Graphics является сдвиг на 100 единиц в направлении оси x и на 80 единиц в направлении оси y. Объемным преобразованием контейнера является поворот на 30 градусов. Код осуществляет вызов метода DrawRectangle(pen, -60, -30, 120, 60) дважды. Первый вызов DrawRectangle осуществляется внутри контейнера. Это значит, что он происходит между вызовами BeginContainer и EndContainer. Второй вызов DrawRectangle происходит после вызова EndContainer.

```
Graphics graphics = e.Graphics;
Pen pen = new Pen(Color.Red);
GraphicsContainer graphicsContainer;
graphics.FillRectangle(Brushes.Black, 100, 80, 3, 3);

graphics.TranslateTransform(100, 80);

graphicsContainer = graphics.BeginContainer();
graphics.RotateTransform(30);
graphics.DrawRectangle(pen, -60, -30, 120, 60);
graphics.EndContainer(graphicsContainer);

graphics.DrawRectangle(pen, -60, -30, 120, 60);
```

В приведенном выше коде к прямоугольнику, рисуемому внутри контейнера, применяется сначала объемное преобразование контейнера (поворот), а затем объемное преобразование объекта Graphics (сдвиг). К прямоугольнику, рисуемому вне контейнера, применяется только объемное преобразование объекта Graphics (сдвиг).

Обрезка во вложенных контейнерах

В приведенном ниже примере демонстрируется, каким образом вложенные контейнеры работают с областями обрезки. В приведенном примере создаются объект Graphics и контейнер внутри этого объекта Graphics. Областью обрезки объекта Graphics является прямоугольник, а областью обрезки контейнера — эллипс. В коде осуществляются два вызова метода DrawLine. Первый вызов DrawLine осуществляется внутри контейнера, а второй вызов DrawLine — вне контейнера (после вызова метода EndContainer). Первая линия обрезается по области, являющейся

пересечением двух указанных областей отсечения. Вторая линия обрезается по прямоугольнику, являющемуся областью обрезки объекта Graphics.

```
Graphics graphics = e.Graphics;
GraphicsContainer graphicsContainer;
Pen redPen = new Pen(Color.Red, 2);
Pen bluePen = new Pen(Color.Blue, 2);
SolidBrush aquaBrush = new
SolidBrush(Color.FromArgb(255, 180, 255, 255));
SolidBrush greenBrush = new
SolidBrush(Color.FromArgb(255, 150, 250, 130));

graphics.SetClip(new Rectangle(50, 65, 150, 120));
graphics.FillRectangle(aquaBrush, 50, 65, 150, 120);

graphicsContainer = graphics.BeginContainer();
// Create a path that consists of a single ellipse.
GraphicsPath path = new GraphicsPath();
path.AddEllipse(75, 50, 100, 150);

// Construct a region based on the path.
Region region = new Region(path);
graphics.FillRegion(greenBrush, region);

graphics.SetClip(region, CombineMode.Replace);
graphics.DrawLine(redPen, 50, 0, 350, 300);
graphics.EndContainer(graphicsContainer);

graphics.DrawLine(bluePen, 70, 0, 370, 300);
```

Как показывается в двух приведенных выше примерах, преобразования и области обрезки во вложенных контейнерах являются кумулятивными. Если объемные преобразования устанавливаются как для контейнера, так и для объекта Graphics, к отображаемым внутри контейнера объектам применяются оба эти преобразования. Сначала применяется преобразование контейнера, а затем осуществляется преобразование объекта Graphics. Если области обрезки устанавливаются как для контейнера, так и для объекта Graphics, областью обрезки для объектов, отображаемых внутри контейнера, является пересечение указанных областей обрезки.

Параметры качества во вложенных контейнерах

Параметры качества (SmoothingMode, TextRenderingHint и т. д.) во вложенных контейнерах не являются кумулятивными; вместо этого параметры качества контейнера временно замещают параметры качества объекта Graphics. При создании нового контейнера в качестве параметров качества для него устанавливаются значения по умолчанию. Например, рассмотрим объект Graphics с режимом сглаживания AntiAlias. Когда создается контейнер, режим сглаживания в нем является режимом сглаживания по умолчанию. Можно свободно устанавливать режим сглаживания контейнера, и любые объекты, отображаемые внутри этого контейнера, будут рисоваться в установленном режиме. Объекты, рисуемые после вызова метода EndContainer, будут рисоваться с режимом сглаживания (AntiAlias), действовавшим до вызова метода BeginContainer.

Различные уровни вложенных контейнеров

Нет ограничения на использование в объекте Graphics только одного контейнера. Можно создать последовательность контейнеров, вложенных друг в друга, и указать объемное преобразование, область обрезки и параметры качества для каждого из этих вложенных контейнеров. Если вызывается какой-либо метод рисования из контейнера наибольшего уровня вложенности, преобразования применяются по порядку, начиная с этого контейнера и оканчивая контейнером наименьшего уровня вложенности. Объекты, отображаемые из контейнера наибольшей степени вложенности, обрезаются по области, являющейся пересечением всех областей обрезки.

В следующем примере создается объект Graphics, для которого устанавливается режим сглаживания текста AntiAlias. В коде создаются два контейнера, один из них вложен в другой. Для режим сглаживания внешнего контейнера устанавливается значение SingleBitPerPixel, а для режима сглаживания внутреннего контейнера — AntiAlias. Код выводит на экран три строки: одну из внутреннего контейнера, одну из внешнего контейнера и одну из самого объекта Graphics.

```
Graphics graphics = e.Graphics;
GraphicsContainer innerContainer;
GraphicsContainer outerContainer;
SolidBrush brush = new SolidBrush(Color.Blue);
FontFamily fontFamily = new FontFamily("Times New
Roman");
Font font = new Font(fontFamily, 36, FontStyle.Regular,
GraphicsUnit.Pixel);
```

```
graphics.TextRenderingHint =
System.Drawing.Text.TextRenderingHint.AntiAlias;

outerContainer = graphics.BeginContainer();

graphics.TextRenderingHint =
System.Drawing.Text.TextRenderingHint.SingleBitPerPixel
;

innerContainer = graphics.BeginContainer();
graphics.TextRenderingHint =
System.Drawing.Text.TextRenderingHint.AntiAlias;
graphics.DrawString(
   "Inner Container",
   font,
   brush,
   new PointF(20, 10));
graphics.EndContainer(innerContainer);

graphics.DrawString(
   "Outer Container",
   font,
   brush,
   new PointF(20, 50));

graphics.EndContainer(outerContainer);

graphics.DrawString(
   "Graphics Object",
   font,
   brush,
   new PointF(20, 90));
```

Содержание работы

1. Ознакомиться с теоретической частью методических указаний.
2. Выполнить примеры
3. Получить вариант задания от преподавателя.
4. Решить поставленную задачу
5. Представить работающую программу преподавателю.

Лабораторная работа № 28 (2 часа)
Построение графиков функций - гистограмм

Построение круговых диаграмм с элементами 3D графики требует несколько больших затрат по сравнению с рассмотренным выше материалом. Прежде всего, необходимо определить дополнительные переменные для величин: оси эллипса (vfDiamX, vfDiamY), центр круговой диаграммы (vfXcirc, vfYcirc). Кроме того, если мы хотим, что бы в легенде (пояснению к графику) цвета надписей соответствовали цветам секторов диаграммы, то потребуется задать массив цветов однозначно соответствующий массиву цветов кистей. Зададим в классе:

```
private float vfDiamX = 100;
private float vfDiamY = 100;
private float vfXcirc = 100;
private float vfYcirc = 100;
private Color[] color ={
Color.LightGreen,Color.Chartreuse,Color.LimeGreen,Color.Green,Co
lor.DarkGreen,
Color.DarkOliveGreen,Color.LightPink
,Color.LightSeaGreen,Color.LightCoral ,Color.DarkCyan ,
Color.Crimson , Color.CornflowerBlue
,Color.Chocolate,Color.CadetBlue,Color.BlueViolet,
Color.Maroon,Color.Blue,Color.Brown,Color.DarkBlue, Color.Red,
Color.Coral,Color.DarkRed, Color.DarkMagenta,
Color.DarkOrange,Color.DarkOrchid};
```

Основная функция для рисования диаграммы имеет ряд особенностей, связанных с формированием объемности и расположением надписей.

Рисовать диаграмму будем в несколько этапов:

- Первый этап: Против часовой стрелки рисуем последний сектор и сектора, выходящие за границу 180 градусов, но не заходящие за границу 270 градусов. Рисовать будем кистью с прозрачностью, например 25%, и каждый из них со сдвигом на 1 пиксель вниз. Иначе, если толщина диаграммы задана 20 пикселей, то сектор потребуется нарисовать 20 раз, каждый раз сдвигая на 1 пиксель вниз.
- Второй этап: Накладываем на данную диаграмму сектора от 0 градусов до сектора, заходящего за 270 градусов, используя SolidBrush и не выполняя сдвиг - рисуем каждый сектор один раз из точки рисования всей диаграммы.
- Третий этап: по часовой стрелки рисуем сектора, начиная со второго, используя HatchBrush. Рисование выполняем до сектора заходящего за границу -90 градусов, со сдвигом на толщину диаграммы.
- Четвертый этап: По часовой стрелке накладываем без сдвига, начиная со второго сектора до сектора, заходящего за границу -90 градусов, используя SolidBrush.
- Отдельно рисуем первый сектор, сначала используя HatchBrush со сдвигом на толщину диаграммы, затем накладываем сектор SolidBrush без сдвига.

Координаты сектора определяем с учетом параметров сдвига секторов (Рис.12.5.).

- Алгоритм рисования можно упростить, например, один раз и последним этапом наложить сектора эллипса, нарисованный кистью SolidBrush, но, в этом случае пострадает наглядность.

Эти этапы рисования выполняет следующая функция:

```
#region  vDravCircle3D
//Параметры - Отступ от краев по X слева deltaaxisL, от краев по
Y справа deltaaxisR,
//deltaaxisH - отступа сверху и снизу, толщина диаграммы viH,
сдвиг сектора viDx, viDy
public void vDravCircle3D(int deltaaxisL, int deltaaxisR,
                int deltaaxisH, int viH, int viDx, int viDy)
{
 //Запоминаем отступы
 viDeltaaxisL = deltaaxisL;
 viDeltaaxisR = deltaaxisR;
 viDeltaaxisH = deltaaxisH;
 float a = viX - (deltaaxisL + deltaaxisR);
 //Нужен ли выброс сектора
 int viMov = 1;
 if (viDx == 0 && viDy == 0)
 {
     viMov = 0;
 }
 //Запоминаем диаметр
 vfDiamX = a;
 vfDiamY = viY - 2 * viDeltaaxisH;
 //Запоминаем центр элипса
 vfXcirc = deltaaxisL + a / 2;
 vfYcirc = viY / 2;
 graph.SmoothingMode = SmoothingMode.AntiAlias;
 //Определяем сумму всех значений в массиве
 float fSum = 0;
 string s = string.Empty;
 for (int i = 0; i < viMaxRg; i++)
 {
     s = rgsValues[i, 0];
     fSum += float.Parse(s);
 }
 float f = 0;
 float fBSum = 0;
 float fDeltaGrad = (fSum / (float)360);
 SolidBrush objBrush = new SolidBrush(Color.Aqua);
 Random rand = new Random(DateTime.Now.Millisecond);
 float[] frgZn = new float[viMaxRg];
 float[] frgSumGr = new float[viMaxRg];
 for (int i = 0; i < viMaxRg; i++)
 {
     s = rgsValues[i, 0];
     frgZn[i] = float.Parse(s);
     if (i == 0) frgSumGr[i] = 0;
     else frgSumGr[i] = frgZn[i] + frgSumGr[i - 1];
```

```
 }
 for (int i = viMaxRg - 1; i >= 0; i--)
 {
  if (i != viMaxRg - 1 && fBSum < 90) break;
  //f в градусах  fBSum в градусах
  f = frgZn[i] / fDeltaGrad;
  //fBSum = frgSumGr[i] / fDeltaGrad;
  if (i == viMaxRg - 1)
  {
      fBSum = 360 - f;
  }
  else
  {
      fBSum -= f;
  }
  //Для цвета
  int j = i % br.Length;
  float k = f;
  if (f < 1) k = 1;
  //objBrush.Color = Color.FromArgb(rand.Next(255),
rand.Next(255), rand.Next(255));
  if (i != 0)
  {
   if ((fBSum > 90 && fBSum < 180) || i == viMaxRg - 1)
   {
    for (int d = 0; d < viH; d++)
    {
       //Этап 1
       graph.FillPie(new HatchBrush(HatchStyle.Percent25,
color[j]/*objBrush.Color*/),
             vfXcirc - a / 2, vfYcirc - vfDiamY / 2 + d,
               vfDiamX, vfDiamY, fBSum, k);
    }
   }
   objBrush.Color = color[j];
   //Этап 2
   graph.FillPie(objBrush, vfXcirc - a / 2, vfYcirc - vfDiamY /
2,
        vfDiamX, vfDiamY, fBSum, k);
  }
 }
 fBSum = 0;
 for (int i = viMov; i < viMaxRg; i++)
 {
  //f в градусах  fBSum в градусах
  f = frgZn[i] / fDeltaGrad;
  if (i == 1)
  {
      fBSum = frgZn[0] / fDeltaGrad;
  }
  //Для цвета
  int j = i % br.Length;
  float k = f;
```

```
  if (f < 1) k = 1;

  if (fBSum < 90)
  {
      for (int d = 0; d < viH; d++)
      {
          //Этап 3
          graph.FillPie(new HatchBrush(HatchStyle.Percent25,
color[j]),
           vfXcirc - a / 2, vfYcirc - vfDiamY / 2 + d,
              vfDiamX, vfDiamY, fBSum, k);
      }
      objBrush.Color = color[j];
      //Этап 4
      graph.FillPie(objBrush, vfXcirc - a / 2, vfYcirc - vfDiamY
/ 2,
       vfDiamX, vfDiamY, fBSum, k);
  }
  else
  {
      break;
  }
  fBSum += f;
 }
 //Рисуем сдвинутым первый сектор
 //Этап 5
 if (viMov == 1)
 {
  f = frqZn[0] / fDeltaGrad;
  fBSum = 0;
  float k1 = f;
  if (f < 1) k1 = 1;
  for (int d = 0; d < viH; d++)
  {
      graph.FillPie(new HatchBrush(HatchStyle.Percent25,
color[0]),
       vfXcirc - a / 2 + viDx, vfYcirc - vfDiamY / 2 + d - viDy,
          vfDiamX, vfDiamY, fBSum, k1);
  }
  objBrush.Color = color[0];
  graph.FillPie(objBrush, vfXcirc - a / 2 + viDx, vfYcirc -
vfDiamY / 2 - viDy,
  vfDiamX, vfDiamY, fBSum, k1);
 }
}
#endregion
```

Добавляем функции надписи и легенду и, в принципе, построение диаграммы закончено. Единственное, что потребуется от нас при рисовании надписей на диаграмме - это немного вспомнить начальную школу при расчете координат нанесения значений:

```
#region vDravTextCircle
public void vDravTextCircle1(bool vfGde)
{
```

```
  float fSum = 0;
  string s = string.Empty;
  for (int i = 0; i < viMaxRg; i++)
  {
      s = rgsValues[i, 0];
      fSum += float.Parse(s);
  }
  float f = 0;
  float fBSum = 0;
  float flRadian = (float)Math.PI / 180;
  float fDeltaGrad = fSum / 360;
  for (int i = 0; i < viMaxRg; i++)
  {
   s = rgsValues[i, 0];
   f = float.Parse(s);
   //f в градусах
   f = f / fDeltaGrad;
   int j = i % br.Length;
   //Угол в радианах
   float fRad = (f + fBSum) * flRadian;
   float fty = 0;
   float ftx = 0;
   float fSin = (float)Math.Sin((360 - (f / 2 + fBSum)) *
flRadian);
   float fCos = (float)Math.Cos((360 - (f / 2 + fBSum)) *
flRadian);
   float c = (float)Math.Sqrt((vfDiamX / 2 * vfDiamX / 2 *
vfDiamY / 2 * vfDiamY / 2) /
       (vfDiamY / 2 * vfDiamY / 2 * fCos * fCos + vfDiamX / 2 *
vfDiamX / 2 * fSin * fSin));
   c -= 3 * objFont.Size;
   if (c < 0) c = 0;
   ftx = c * fCos;
   fty = c * fSin;
   ftx = vfXcirc + ftx;
   fty = vfYcirc - fty;
   if (vfGde)
   {
    graph.DrawString(Convert.ToString(i + 1), objFont, objBrush,
ftx, fty);
   }
   else
   {
    graph.DrawString(rgsValues[i, 0], objFont, objBrush, ftx,
fty);
   }
   fBSum += f;
  }
}
#endregion

#region Текст легенды
public void vDravTextKeyCircle(bool vfGde)
```

```
{
 float fSum = 0;
 float f = 0;
 string s = string.Empty;
 for (int i = 0; i < viMaxRg; i++)
 {
  s = rgsValues[i, 0];
  fSum += float.Parse(s);
 }
 //Сдвиг от круговой диаграммы
 float vfSdvig = vfXcirc + vfDiamX / 2;
 vfSdvig += (viX - vfSdvig) / 5;
 //Высота места для легенды
 //На одну строку по высоте отводится - +1 на заголовок
 float vfHg = viY / (viMaxRg + 2);
 vSetFont("Arial", 12, true);
 if (viMaxRg > 100)
 {
  graph.DrawString("Легенда не может быть размещена",
   objFont, Brushes.DarkBlue, vfSdvig + (viX - vfSdvig) / 10,
objFont.Size);
 }
 else
 {
  //Шрифт в 2 раза меньше места на строку надписи
  if (viMaxRg > 15)
  {
      vSetFont("Arial", (vfHg / 2), true);
  }
  else
  {
   if (viMaxRg > 10)
   {
       vSetFont("Arial", (vfHg / 3), true);
   }
   else
   {
       vSetFont("Arial", (vfHg / 6), true);
   }
  }
  if (vfGde)
  {
      graph.DrawString("Пояснения к графику",
       objFont, Brushes.DarkBlue, vfSdvig /*+ (viX - vfSdvig) /
10*/, objFont.Size);
  }
  else
  {
      graph.DrawString("Пояснения к графику",
        objFont, objBrush, vfSdvig/* + (viX - vfSdvig) / 10*/,
objFont.Size);
  }
  if (viMaxRg > 15)
```

```
  {
      vSetFont("Arial", (vfHg / 2) + 1, true);
  }
  else
  {
   if (viMaxRg > 10)
   {
       vSetFont("Arial", (vfHg / 4) + 1, true);
   }
   else
   {
       vSetFont("Arial", (vfHg / 7) + 1, true);
   }
  }
  for (int i = 0; i < rgsValues.Length / 2; i++)
  {
   Brush brTxt = null;
   int j = i % br.Length;
   if (vfGde) brTxt = br[j];
   else brTxt = objBrush;
   graph.DrawString(Convert.ToString(i + 1), objFont, brTxt,
vfSdvig, vfHg * (i + 2));
   f = float.Parse(rgsValues[i, 0]);
   f = (f * 100) / fSum;
   graph.DrawString(rgsValues[i, 0], objFont,
       brTxt, vfSdvig + 1 * (viX - vfSdvig) / 5, vfHg * (i +
2));
   graph.DrawString(f.ToString("0.0") + "%", objFont,
       brTxt, vfSdvig + 2 * (viX - vfSdvig) / 5, vfHg * (i +
2));
   graph.DrawString(rgsValues[i, 1], objFont,
       brTxt, vfSdvig + 3 * (viX - vfSdvig) / 5, vfHg * (i +
2));
  }
 }
}
#endregion


#region Смена шрифта по секторам
private void vSetFont(string name, float size, bool bold)
{
 if (objFont != null) objFont = null;
 if (bold)
 {
     objFont = new Font(name, size, FontStyle.Bold);
 }
 else
 {
     objFont = new Font(name, size);
 }
}
#endregion
```

Оформим вызовы функций:

```
private void button3_Click(object sender, EventArgs e)
{
 viNumButton = 3;
 vCreateCircleDiagramm();
}
private void vCreateCircleDiagramm()
{
 //Создаем массив значений для вывода на графике
 vCreateRg();
 //Создаем класс и передаем ему размер холсты
 PaintCl clPaint = new PaintCl(pictureBox1.Width,
pictureBox1.Height);
 //Фон холста
 clPaint.vSetBackground(Color.White);
 //Передаем значения массива в класс
 clPaint.RgValue = rgsValues;
 //Рисуем график. Параметры: отступ осей x слева, x справа ,
 //y от краев холста, толщина диаграммы,вынос сектора
 clPaint.vDravCircle3D(20, 250, 50, 20, 20, 40);
 //Круговые надписи true цифры 1-20, false - значения
 clPaint.vDravTextCircle1(true);
 //false - Разноцветные надписи в легенде true - Цветом шрифта
 clPaint.vDravTextKeyCircle(true);
 //Принимаем нарисованное в pictureBox
 pictureBox1.Image = clPaint.Bmp;
}
```

Отметим, что при задании толщины диаграммы, равной нулю, получим обычную эллиптическую диаграмму, а при равенстве осей X и Y - круговую. В заключении, еще раз повторим, что все параметры целесообразно иметь настраиваемыми, что позволяет быстро подобрать приемлемый вид графического отображения для демонстрации. Целесообразно также выполнить автономную настройку диаграмм по тестовым значениям. Тогда мы сможем быстро подбирать параметры, например так:

```
//Смена фона диаграммы
private void ColorBackGround_Click(object sender, EventArgs e)
{
 if (colorDialog1.ShowDialog() == DialogResult.OK)
 {
  //Переменная objColorBackGroung задана глобально, сохраняется
в реестре
  //при закрытии приложения и передается в класс при рисовании
диаграммы
  //clPaint.vSetBackground(objColorBackGroung);
  objColorBackGroung = colorDialog1.Color;
  vGhangeDiagramm();
 }
}
//Перерисовка конкретной диаграммы при настройке
private void vGhangeDiagramm()
```

```
{
 switch (viNumButton)
 {
  case 1:
   vCreateLinGr();
  break;
  case 2:
   vCreateRectangleDiagramm();
  break;
  case 3:
   vCreateCircleDiagramm();
   break;
 }
}
```

Содержание работы

1. Ознакомиться с теоретической частью методических указаний.
2. Выполнить примеры
3. Получить вариант задания от преподавателя.
4. Решить поставленную задачу
5. Представить работающую программу преподавателю.

Лабораторная работа № 29 (2 часа)

Работа с растровыми изображениями

Основой для графических операций *Windows* служит интерфейс прикладных программ (*API - Application Program Interface*) графических устройств (*GDI - Graphic Device Interface*).

Перечень атрибутов *GDI*, их значения, устанавливаемые при запуске приложения по умолчанию, а также функции, связанные с этими атрибутами, приведены в табл.1.

Таблица 1

Атрибуты и функции GDI

Атрибут	Значение по умолчанию	Функции GDI
Цвет фона	White	SetBkColor
Режим фона	OPAQUE	SetBkMode
Растровое изображение	Нет	CreateBitmap CreateBitmapIndirect CreateCompatibleBitmap
Кисть	WHITE_BRUSH	SelectObject CreateBrushIndirect CreateDIBPatternBrush CreateHatchBrush CreatePatternBrush CreateSolidBrush SelectObject
Начало системы координат (СК) для кисти	(0,0)	SetBrushOrg UnrealizeObject
Отсечение	Поверхность экрана	ExcludeClipRect IntersectClipRect OffsetClipRgn SelectClipRgn
Цветовая палитра	DEFAULT_PALETTE	CreatePalette RealizePalette SelectPalette
Положение пера	(0,0)	MoveTo
Режим рисования	R2_COPYPEN	SetROP2
Шрифт	SYSTEM_FONT	CreateFont CreateFontIndirect SelectObject
Межсимвольное расстояние	0	SetTextCharacterExtra
Режим отображения	MM_TEXT	SetMapMode
Перо	BLACK_PEN	CreatePen CreatePenIndirect

		SelectObject
Режим заполнения	ALTERNATE	SetPolyFillMode
Режим растяжения	BLACKONWHITE	SetStretchBltMode
Цвет текста	Черный	SetTextColor
Масштаб экрана	(1,1)	SetViewportExt
Начало физической СК	(0,0)	SetViewportOrg
Масштаб окна	(1,1)	SetWindowExt
Начало СК окна	(0,0)	SetWindowOrg

Непосредственное использование подобных функций низкого уровня приводит к значительным временным затратам. Для ускорения процесса программирования в современных ИСР имеются набор элементов более высокого уровня и поддержка их программирования в виде иерархий классов. Подобный прием позволяет программисту сосредоточиться на решении основной задачи (создании интерфейса), а не на проблемах взаимодействия приложения с логическими и физическими устройствами компьютера.

Классы *TCanvas* и *TGraphicsObject*

Базовым классом для работы с графикой является класс *TCanvas*, инкапсулирующий внутри себя все функции *GDI* (см. табл.1). Построение изображений из плоских графических примитивов осуществляется при помощи методов класса *TGraphicsObject*, включающего в себя функции для работы со шрифтами (*Font*), перьями (*Pen*) и закраски областей (*Brush*). При помощи данных инструментов можно оперировать с такими графическими элементами, как отрезки прямых, дуги и хорды, прямоугольники и многоугольники, круги, эллипсы и их сегменты, а также отображать на экране текстовую информацию с использованием различных шрифтов *Windows*.

У ряда объектов из библиотеки визуальных компонентов есть свойство *Canvas* (канва, холст), которое предоставляет возможность рисования на них. К таким объектам относятся: *TBitmap, TComboBox, TDBComboBox, TDBGrid, TDBListBox, TDirectoryListBox, TDrawGrid, TFileListBox, TForm, TImage, TListBox, TOutline, TPaintBox, TPrinter, TStringGrid. Canvas* является в свою очередь объектом, объединяющим в себе поле для рисования (*PaintBox*), перо (*Pen*), кисть (*Brush*) и шрифт (*Font*). *Canvas* обладает также рядом графических методов: *Draw, TextOut, Arc, Rectangle* и др. Используя *Canvas*, вы можете воспроизводить на форме любые графические объекты - картинки, многоугольники, текст и т.п. без использования компонент *TImage,TShape* и *TLabel* (т.е. без использования дополнительных ресурсов), однако при этом вы должны обрабатывать событие *OnPaint* того объекта, на канве которого вы рисуете. Рассмотрим подробнее свойства и методы объекта *Canvas*.

Свойства объекта *Canvas*

Свойство *Brush* (кисть) представляет собой объект, с помощью которого производится заполнение сплошных областей. Кисть обладает следующим набором свойств:
Bitmap - картинка кисти размером 8´8, используемая для заливки (заполнения) области на экране;
Color - цвет заливки;
Style - предопределенный стиль заливки (это свойство является взаимоисключающим со свойством *Bitmap*);
Handle – указатель на объект, который дает возможность использовать кисть в прямых вызовах процедур *Windows API*.
Свойство *ClipRect* (прямоугольник) определяет прямоугольную область, на которой происходит рисование.
Свойство *CopyMode* (режим копирования) определяет, каким образом будет происходить копирование при помощи метода *CopyRect* графических элементов на форму (наложение один к одному, с инверсией изображения и т.п.).
Свойство *Font* (шрифт) определяет шрифт, которым выводится текст (при помощи метода *TextOut*).
Свойство *Handle* (указатель) используется для прямых вызовов функций и структур данных *Windows API*.
Свойство *Pen* (перо) определяет вид линий. Как и кисть, оно является объектом, имеющим ряд свойств:
Color - цвет линии;
Handle - указатель для прямых вызовов *Windows API*;
Mode - режим вывода: простая линия, с инвертированием, с выполнением "исключающего ИЛИ" и др.;
Style - стиль вывода: линия, пунктир и др.;
Width - ширина линии в точках.

Свойство *PenPos* (позиция пера) определяет текущую позицию пера. Однако перо рекомендуется перемещать при помощи метода *MoveTo*, а не прямой установкой данного свойства.
Свойство *Pixels* - двухмерный массив элементов изображения (*пикселей*), с помощью которого можно получить доступ к отдельной точке изображения.

Методы объекта *Canvas*
Методы для рисования простейших графических элементов - *Arc*, *Chord*, *LineTo*, *Pie*, *Polygon*, *PolyLine*, *Rectangle*, *RoundRect*. При прорисовке линий в этих методах используются перо (*Pen*), а для заполнения внутренних областей - кисть (*Brush*).
Методы для *вывода картинок* на канву - *Draw* и *StretchDraw*. В качестве параметров указываются прямоугольник и графический объект для вывода (это может быть *TBitmap*, *TIcon* или *TMetafile*). Метод *StretchDraw* отличается тем, что растягивает или сжимает картинку так, чтобы она заполнила весь заданный прямоугольник.

Методы для *вывода текста* - *TextOut* и *TextRect*. При выводе текста используется шрифт (*Font*) канвы. При использовании *TextRect* текст выводится только внутри указанного прямоугольника. Длину и высоту текста можно узнать с помощью функций *TextWidth* и *TextHeight*.

Далее описывается, как внедрить метафайла и растровых изображений, как .png и .jpg файлы как ресурсы в Visual C#.NET и как извлечь их для использования с **System.Drawing** классы.

Создание проекта

1. Запустите Microsoft Visual Studio.
2. На **Файл** меню, нажмите кнопку **Новый**, а затем нажмите кнопку **Проект- Приложение Windows**.
3. Введите соответствующее имя (**WindowsApplication1**), а затем нажмите кнопку **ОК**.

Добавить ресурс

1. На **Представление** меню, нажмите кнопку **Обозреватель решений**, а затем выберите новый проект, созданный ранее.
2. На **Проект** меню, нажмите кнопку **Добавление существующего элемента**.
3. Найдите и откройте один растровый файл (.jpg, .png или .tif) и один метафайлов (.wmf и .emf) и затем выполните следующие действия для каждого файла:
   a. В **Обозреватель решений**, выберите файл, а затем на **Представление** меню, нажмите кнопку **Окно "Свойства"**.
   b. Установка **Действие при построении** Кому **Внедренный ресурс**.
   c. На **Проект** меню, нажмите кнопку **Добавление существующего элемента**.

Создать кнопку для отображения растровых изображений

1. На **Представление** меню, нажмите кнопку **Панели инструментов** и затем добавьте кнопку в форму. По умолчанию это **Button1**.
2. Повторите шаг 1 для создания **Button2**.
3. Дважды щелкните значок **Button1** Чтобы открыть окно кода и добавьте следующий код для **Button1_Click()** функции:

```
4.      // Replace "filename" below with the actual filename for the JPG
5.      //  file you added as a resource; the name is case-sensitive.
6.      // Also make sure that "WindowsApplication1" is replaced with the
7.      //  name of your project, if different.
8.      Stream s =
this.GetType().Assembly.GetManifestResourceStream("WindowsApplication1.
filename.jpg");
9.      Bitmap bmp = new Bitmap( s );
10.     s.Close();
11.     Graphics g = CreateGraphics();
12.     g.DrawImage( bmp, 0, 0 );
```

```
13.     bmp.Dispose();
14.     g.Dispose();
```

15. Заменить «windowsapplication1» в **GetManifestResourceStream()** вызов с именем проекта.
16. Заменить «filename.jpg» в **GetManifestResourceStream()** вызов с учетом регистра имя добавленного файла.

Создать кнопку для отображения изображения метафайла

1. На **Представление** меню, нажмите кнопку **Конструктор**.
2. Дважды щелкните значок **Button2** Чтобы открыть окно кода и добавьте следующий код для **Button2_Click()** функции:

```
3.      // Replace "filename" below with the actual filename for the metafile
4.      //  file you added as a resource; the name is case-sensitive.
5.      // Also make sure you replace "WindowsApplication1" with the
6.      //  name of your project, if different.
7.      Stream s =
this.GetType().Assembly.GetManifestResourceStream("WindowsApplication1.
filename.emf");
8.      Metafile emf = new Metafile( s );
9.      s.Close();
10.     Graphics g = CreateGraphics();
11.     g.DrawImage( emf, 0, 0, 300, 300 );
12.     emf.Dispose();
13.     g.Dispose();
```

14. Заменить «windowsapplication1» в **GetManifestResourceStream()** вызов с именем проекта.
15. Заменить «filename.emf» в **GetManifestResourceStream()** Вызовите к регистру имя метафайла, который был добавлен.

Готово, построение и запуск приложения

1. В начале исходного файла добавьте следующие строки кода:

```
2.      using System.IO;
3.      using System.Drawing.Imaging;
```

4. На **Построение** меню, нажмите кнопку **Построение решения**.
5. На **Отладка** меню, нажмите кнопку **Начало**.
6. Чтобы отобразить растровое изображение, нажмите кнопку **Button1**. Для отображения ваших метафайла, нажмите кнопку **Button2**.

Устранение неполадок

Исключения могут возникать, если какой-либо из следующих условий:

- Строка «windowsapplication1» не отражает фактическое имя вашего проекта.

- Строка «filename.jpg», описанных в разделе «Создать кнопку для отображения растровое изображение» не отражает фактическое имя файла, который был добавлен в качестве ресурса.
- Существует вариантов несоответствия между текстом аргумент для **GetManifestResourceStream()** и фактических имен объектов, составляющих строку.

Содержание работы

1. Ознакомиться с теоретической частью методических указаний.
2. Выполнить примеры
3. Получить вариант задания от преподавателя.
4. Решить поставленную задачу
5. Представить работающую программу преподавателю.

Лабораторная работа № 30 (2 часа)
Разработка программы с элементами анимации

На примере многочисленных хранителей экрана (screen-saver) вы видели, как гладко работает анимация в OpenGL. OpenGL использует два буфера памяти (front and back). Первый (front-буфер) отображается на экране, второй в это время может обрабатываться процессором. Когда обработка закончится, то есть очередная сцена будет готова, вы можете произвести быстрое переключение буферов (swap), обеспечивая тем самым гладкую анимацию изображения. При обмене копирование массивов не происходит, изменяется лишь значение указателя (адреса) отображаемого блока памяти. Отметьте, что процесс рисования в back-буфер происходит быстрее, чем в front, так как большинство видеокарт запрещают редактировать изображение в момент вертикальной развертки, а это происходит 60-90 раз в секунду.

Рассмотрим основную схему алгоритма анимации, используемого в OpenGL-приложениях. В кино эффект движения достигается тем, что каждый кадр проецируется на экран в течение короткого промежутка времени, затем шторка проектора моментально закрывается, пленка продвигается на один кадр, вновь открывается шторка и цикл повторяется. Период цикла равен 1/24 с или даже 1/48 с в современных кинопроекторах. Современные компьютеры допускают смену кадра (refresh rate) до 120 раз в секунду. Рассмотренный алгоритм можно записать так в цикле по всем кадрам:

1. сотри старое изображение;
2. создай новое изображение;
3. задержи изображение на какое-то время.

Если реализовать анимацию по такой схеме, то эффект будет тем более удручающим, чем ближе к 1/24 с подходит время создания изображения, так как полное изображение существует на экране лишь долю периода. Большую часть периода мы видим процесс рисования.

OpenGL предоставляет возможность двойной буферизации (аппаратной или программной - зависит от видеокарты). Алгоритм анимации в этом случае таков: пока проецируется первый кадр, создается второй. Переключение кадров происходит только после того, как закончится формирование второго кадра. Пользователь никогда не видит незавершенный кадр. Эту схему можно представить в виде проектора с двумя кадрами пленки. В момент демонстрации первого второй стирается и вновь рисуется. Новый алгоритм можно записать в цикле по кадрам:

1. Сотри старое изображение в back-буфере.
2. Создай в нем новое изображение.
3. Переключи буферы (front-back).

Последний шаг алгоритма не начнет выполняться, пока не закончится предыдущий шаг - создание нового кадра в back-буфере. Ожидание этого события (конец рисования в невидимый буфер) дополняется ожиданием завершения цикла прямого хода развертки экрана. Поэтому самая большая

частота смены изображений равна текущему значению частоты кадров дисплея. Допустим, что эта частота равна 60 Гц, тогда частота смены изображений будет 60 fps (frames per second - кадров в секунду). Если время рисования занимает немногим более 1/60 с (пусть 1/45 с), то один и тот же кадр будет проецироваться два такта цикла развертки и частота смены изображений реально будет 30 fps. Промежуток времени между 1/30 с и 1/45 с процессор простаивает (is idle). Если время подготовки невидимого кадра нестабильно (плавает), то частота смены изображений может измениться скачком, что воспринимается как неприятная помеха. Для сглаживания этого эффекта иногда искусственно добавляют небольшую задержку, с тем чтобы немного снизить частоту, но сделать ее стабильной. Отметьте, что OpenGL не имеет команды переключения буферов, так как такая команда всегда зависит от платформы. Мы будем пользоваться функцией SwapBuf f ers(HDC hdc), входящей в состав Windows API.

Самый простой способ создания это поочередная прорисовка картинок на форме (Form) или PictureBox методом **PaintPicture**.

Для реализации этого способа нам потребуется форма (Form), таймер (Timer), и массив имеджей (Image), количество последних зависит от количества кадров.

Установите свойства элементов следующим образом:

Form

AutoRedraw	True

Timer

Interval	100

Image

Name	imgPic
Index	от 0 до количества картинок (у меня от 0 до 3)
Picture	в каждый по картинке
Visible	False

Теперь, когда все на месте, перейдем к коду. Для начала в General Declaration необходимо объявить три локальные переменные:

```
Dim i As Integer 'счетчик кадров
Dim fX As Single, fY As Single 'позиция ввода
```

Переменная i будет использоваться как счетчик кадров, а fX и fY необходимы для определения позиции прорисовки картинки на форме. Позиция будет определяться в событие Form_Resize:

```
Private Sub Form_Resize()
 'забиваем данные в fX и fY, чтобы картинка _
 была по центру формы
 fX = Round((Me.ScaleWidth - imgPic(0).Width) / 2)
 fY = Round((Me.ScaleHeight - imgPic(0).Height) / 2)

End Sub
```

Далее добавляем код прорисовки в таймер:

```
Private Sub Timer1_Timer()

 'Очищаем форму
 Me.Cls
 'Рисуем картинку на форме
 Me.PaintPicture imgPic(i).Picture, fX, fY
 i = i + 1
 If i > (imgPic.Count - 1) Then i = 0

End Sub
```

Единственный положительный момент этого способа, так это то, что он очень простой.

Способ #2. Анимация с использованием технологии BitBlt

Для создания анимации с использованием технологии **BitBlt** нам также потребуется форма (Form), таймер (Timer), но в отличие от предыдущего способа, уже два массива имеджей (Image) и два пикчербокса (PictureBox). Один массив имеджей будет использоваться для хранения картинок, другой для масок этих картинок. Маски это те же самые картинки, но черного цвета, сделать их можно средствами обычного MS Paint(a)

Установите свойства элементов следующим образом:

Form

AutoRedraw	True
ScaleMode	3

Timer

Interval	100

Image

Name	imgPic
Index	от 0 до количества картинок (у меня от 0 до 3)
Picture	в каждый по картинке
Visible	False

Image

Name	imgMask
Index	от 0 до количества масок (у меня от 0 до 3)
Picture	в каждый по картинке
Visible	False

PictureBox

Name	picPic
AutoRedraw	True
AutoSize	True
ScaleMode	3

Visible False

PictureBox
Name picMask
AutoRedraw True
AutoSize True
ScaleMode 3
Visible False

Далее, по сложившейся традиции, займемся кодом и в General Declaration объявим API функцию **BitBlt**:

```
Private Declare Function BitBlt Lib "gdi32" (ByVal hDestDC As Long, ByVal X As Long,
ByVal Y As Long, ByVal nWidth As Long, ByVal nHeight As Long, ByVal hSrcDC As Long,
ByVal xSrc As Long, ByVal ySrc As Long, ByVal dwRop As Long) As Long
```

, а также объявим необходимые переменные:

```
Dim i As Integer 'счетчик кадров
Dim fX As Single, fY As Single 'позиция ввода
Dim fW As Single, fH As Single 'ширина/высота картинки
```

Помимо уже знакомых i, fX и fY, появились еще две переменные fW и fH они необходимы для указания размера рисуемой картинки. Позицию прорисовки определяем также как и в предыдущем способе:

```
Private Sub Form_Resize()
  'забиваем данные в fX и fY, чтобы картинка _
 была по центру формы
 fX = Round((Me.ScaleWidth - imgPic(0).Width) / 2)
 fY = Round((Me.ScaleHeight - imgPic(0).Height) / 2)

End Sub
```

Размер картинок установим в событие Form_Load:

```
 Private Sub Form_Load()
   'устанавливаем размер рисуемой картинки _
 в зависимости от размера первой картинки
 fW = imgPic(0).Width
 fH = imgPic(0).Height
 End Sub
```

Затем, добавляем код прорисовки в таймер:

```
Private Sub Timer1_Timer()
 Me.Cls
```

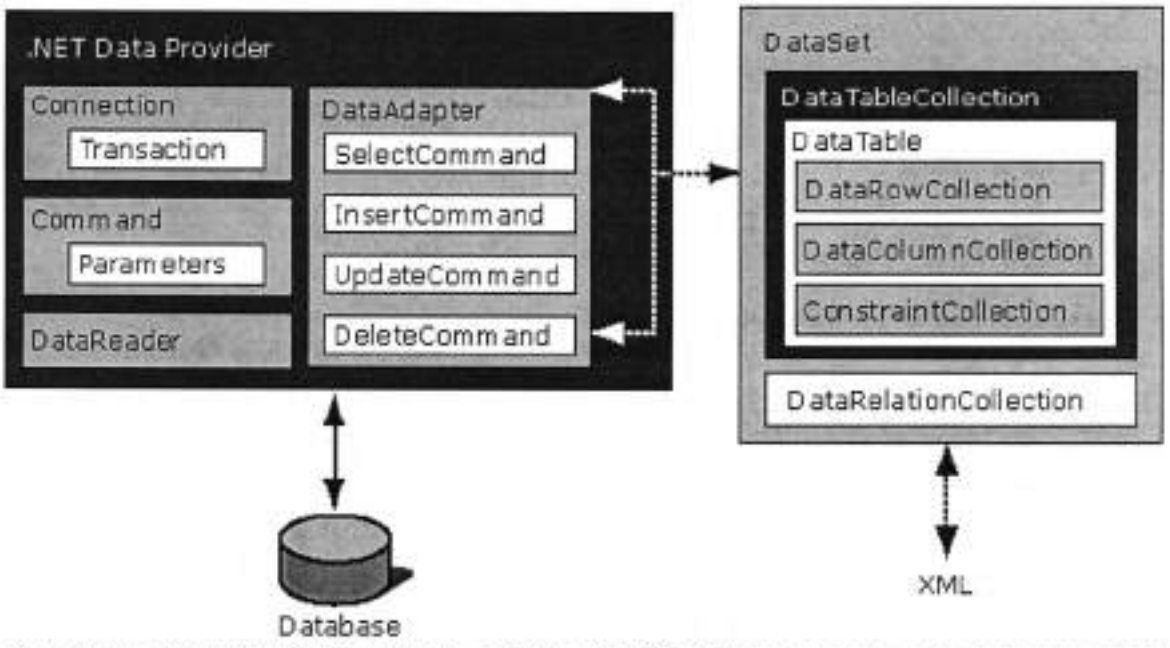
```
 picMask.Picture = imgMask(i).Picture
picPic.Picture = imgPic(i).Picture
'Сначала рисуем маску
BitBlt Me.hDC, fX, fY, fW, fH, _
picMask.hDC, 0, 0, vbMergePaint
'Затем картинку
BitBlt Me.hDC, fX, fY, fW, fH, _
picPic.hDC, 0, 0, vbSrcAnd
'Обновляем форму, а то ничего не будет видно
Me.Refresh
'Прибавляем счетчик
i = i + 1
'Если значение счетчика больше количества кадров _
то аннулируем его
If i > (imgPic.Count - 1) Then i = 0

End Sub
```

Содержание работы

1. Ознакомиться с теоретической частью методических указаний.
2. Выполнить примеры
3. Получить вариант задания от преподавателя.
4. Решить поставленную задачу
5. Представить работающую программу преподавателю.

Лабораторная работа № 31 (2 часа)
Изучение объектной модели ADO.NET

Объектная модель ADO.NET
MSDN изображает объектную модель ADO.NET следующим образом:



Как видно из данной диаграммы, основу ADO.NET составляют два основных компонента (разумеется, в данном случае речь не идет о компонентах COM): DataSet и Провайдер Данных.

Провайдер данных
Провайдер Данных, как это ясно следует из его названия, отвечает за связь приложения с источником данных и манипуляции данными. Основные объекты, входящие в его состав, - это Connection, Command, DataAdapter иDataReader.

Connection используется для установления соединения с источником данных, а также для управления транзакциями.

Command позволяет манипулировать данными источника, а также выполнять хранимые процедуры. При этом могут использоваться параметры для передачи данных в обоих направлениях.
DataAdapter служит связующим звеном между DataSet и источником данных. Он использует Command для выполнения команд SQL как для заполнения DataSet данными, так и для обратной передачи измененных клиентом данных к источнику. Для выполнения этих функций объект имеет 4 метода: SelectCommand, InsertCommand, UpdateCommand и Deletecommand.

DataReader представляет однонаправленный поток данных от источника только на чтение. Если приложение клиента не модифицирует данные и не требуется произвольная выборка данных, а достаточно их однократного просмотра, то использование DataReader вместо DataSet позволит сохранить ресурсы RAM и CPU, а также поднять быстродействие приложения.

DataSet
DataSet можно представить себе как небольшую реляционную базу данных, резидентную в оперативной памяти клиента. В DataSet загружаются данные, после чего соединение с их источником (или источниками, которых в принципе может быть несколько, поскольку в одном

экземпляре DataSet могут объединяться данные из нескольких источников) может быть разорвано. Далее клиент в автономном режиме производит обработку данных, при необходимости модифицирует их, после чего снова устанавливается соединение с источником и модифицированные данные передаются обратно. Такая схема работы может оказаться предпочтительной для корпоративных приложений, работающих в глобальных сетях, где поддержание постоянного соединения может повлечь дополнительные накладные расходы.

DataSet может сохранять свое текущее содержимое в файлах XML, а схему в виде XML Schema definition language (XSD).

Внутреннее устройство DataSet (заимствовано из MSDN):

Рассмотрим вкратце наиболее важные компоненты DataSet, не углубляясь во второстепенные детали.
DataSet включает две основные коллекции: DataTableCollection и DataRelationCollection.

DataTableCollection это коллекция объектов DataTable, каждый из которых представляет одну таблицу отношения и в свою очередь состоит из коллекций:

- DataColumnCollection представляет все столбцы таблицы. Помимо данных, столбец может включать формулу для динамического расчета своего содержимого.
- DataRowCollection представляет набор строк таблицы (возможно, пустой). Следует отметить, что хранятся не только текущие значения данных, но и первоначальные, что позволяет выделить измененные данные.
- ConstraintCollection набор ограничений целостности данных (например, можно задать, допустимо ли для данного столбца значение NULL или должны ли данные быть уникальными).

Другая коллекция из состава DataSet это DataRelationCollection, которая служит для навигации по связанному набору таблиц и содержит ограничения ссылочной целостности DataSet. В ней, например, можно задать ограничения, которое не позволит удалить запись главной таблицы, если

на нее есть ссылки в подчиненной таблице посредством внешнего ключа, тем самым не допуская появления осиротевших (orphaned) записей.

Как видно из диаграммы объектной модели ADO.NET, помимо работы с Провайдерами Данных, DataSet может работать и с данными формата XML, причем данные эти могут как храниться в файлах, так и транспортироваться по сетям передачи данных посредством транспорта HTTP. В последнем случае существенно облегчается построение распределенных приложений корпоративного уровня, поскольку HTTP без труда может проходить через брандмауэры, не создавая особой угрозы для безопасности внутренних сетей.

Провайдеры Данных

Как уже говорилось ранее, Провайдер Данных служит для установления соединения с источником данных, манипуляции данными и доставки результатов этих манипуляций.

Совместно с платформой .NET поставляется два штатных Провайдера: SQL Server .NET Data Provider и OLE DB .NET Data Provider. С Web-узла Microsoft можно загрузить ODBC .NET Data Provider (естественно, для работы с источниками данных ODBC). Ходят слухи о скором выпуске специального Провайдера для Oracle.

SQL Server .NET Data Provider оптимизирован для работы с Microsoft SQL Server V7.0 и выше. Он потребляет меньше ресурсов, чем Провайдер OLE DB, и обладает лучшей производительностью. За это приходится платить потерей переносимости на другие сервера баз данных. Находится этот Провайдер в пространстве имен System.Data.SqlClient.

OLE DB .NET Data Provider это более общий Провайдер, способный работать с почти любым источником данных, поддерживающим OLE DB (исключения представлены далее). Разумеется, для своей работы он использует COM, поэтому между Провайдером и источником данных в этом случае присутствуют две прослойки: обертка, позволяющая managed-приложению использовать COM, и собственно провайдер OLE DB для COM. Естественно, в этом случае производительность уступает предыдущему решению, зато можно использовать практически любые источники данных. Находится этот провайдер в пространстве имен System.Data.OleDb.

Microsoft тестировал ADO.NET со следующими провайдерами OLE DB:

Driver	Provider
SQLOLEDB	Microsoft OLE DB Provider for SQL Server
MSDAORA	Microsoft OLE DB Provider for Oracle
Microsoft.Jet.OLEDB.4.0	OLE DB Provider for Microsoft Jet

Не поддерживается OLE DB Provider for ODBC (MSDASQL), а также OLE DB version 2.5 (в частности, Microsoft OLE DB Provider for Exchange и the Microsoft OLE DB Provider for Internet Publishing). **Не рекомендуется** использование баз данных Microsoft Access для многоуровневых приложений.

Для корректной работы Провайдера OLE DB следует установить MDAC 2.6 и выше.

Следует заметить, что хотя все Провайдеры предназначены для одной цели, тем не менее между ними есть разница в деталях строения. Поэтому перейти, допустим, с MS SQL на Oracle простым переименованием объекта не получится, придется внести некоторые изменения в текст программы. Эти изменения не столь обширны, но все же нужно иметь их в виду.

Простой пример приложения ADO.NET

Разумеется, представленной информации явно недостаточно для разработки полноценных приложений для работы с базами данных. Но тем не менее считаю, что простой работающий пример поможет закрепить полученные знания на практике, тем более что в нем, пусть в весьма примитивном виде, представлены основные этапы работы с данными: установка соединения с источником данных, выборка необходимых данных, обработка их в приложении и закрытие соединения по завершении работы.

Конечно, приведенное приложение весьма далеко от совершенства; в нем даже отсутствует код для обработки исключительных ситуаций. Впрочем, на данном этапе это, пожалуй, только сделало бы программу более громоздкой и мало что добавило к пониманию основному предмету нашего обсуждения архитектуры ADO.NET.

Итак, вот текст программы на языке C#. Для простоты она оформлена в виде консольного приложения. Создайте в среде Visual Studio .NET пустой проект консольного приложения, скопируйте в него текст программы и все должно заработать. Конечно же, те, кто пока обходится без Visual Studio и пользуется .NET Framework, тоже могут откомпилировать ее из командной строки. Я бы рекомендовал выполнить ее по шагам в отладчике и понаблюдать за изменениями состояния объектов программы.

Код:

```
using System;
using System.Data;
using System.Data.OleDb;
namespace ConsoleApplication1
{
        class CAdoNetSample
        {
                [STAThread]
                static void Main(string[] args)
                {
                        // строка соединения; настройте ее на расположение базы
                        //данных Борей.mdb в вашей системе
                        string strConn=
        "Provider=Microsoft.Jet.OLEDB.4.0;Password="";User ID=Admin;"+
        "Data Source=C:\Program Files\Microsoft Office\Office10\Samples\Борей.mdb;";

                        // создаем Connection и открываем его
                        OleDbConnection cnn = new OleDbConnection(strConn);
                        cnn.Open();

                        // создаем команду SQL, которая выберет все страны, в которых
                        // живут клиенты, и упорядочит их список по алфавиту обратите
                        // внимание - создаваемая команда связана с соединением cnn
                        OleDbCommand cmd = cnn.CreateCommand();
                        cmd.CommandText=
                                "SELECT DISTINCT Страна FROM Клиенты ORDER BY Страна";

                        // поскольку мы намереваемся читать список последовательно,
                        // то нас вполне устроит функциональность объекта DataReader
```

```
            // выполняем команду; результаты ее выполнения будут доступны
            // через rdr
            OleDbDataReader rdr = cmd.ExecuteReader();

            // выдаем на печать все полученные результаты
            while (rdr.Read())
            {
                Console.WriteLine(rdr.GetString(0));
            }

            // освобождаем занятые ресурсы
            rdr.Close();
            cnn.Close();
        }
    }
```

Содержание работы

1. Ознакомиться с теоретической частью методических указаний.
2. Выполнить примеры
3. Получить вариант задания от преподавателя.
4. Решить поставленную задачу
5. Представить работающую программу преподавателю.

Лабораторная работа № 32 (2 часа)
Подключение компонентов ADO к проекту

Задание. Информационная система хранит информацию о врачах, о пациентах и их лечении. База данных включает в себя следующие сущности:

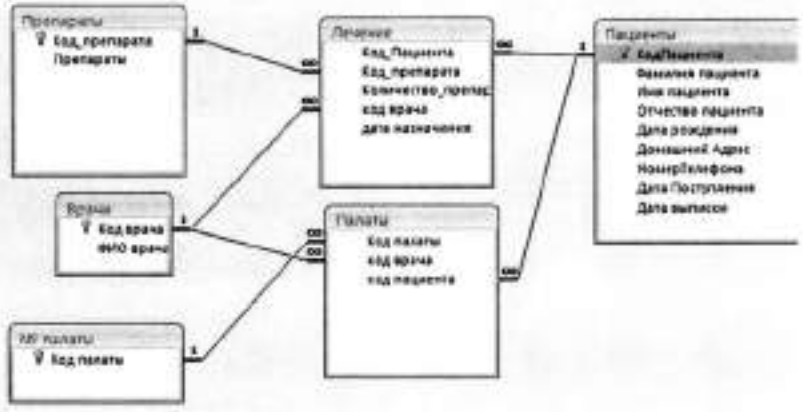


Рис. 1

Создать проект и обеспечить доступ к БД , используя ADO-компоненты.

Создаем проект, сохраняем созданное нами приложение в папку «Больница», где расположен файл базы данных.
Помещаем на форму компонент ADOConnection с вкладки ADO палитры компонентов.
Устанавливаем свойство ConnectionString для этого компонента (на вкладке «Поставщик данных» выбираем «Microsoft Jet 4.0 OLE DB Provider» на вкладке «Подключение» выбираем подключаемый файл).
Поскольку файл базы данных и приложение, разрабатываемое в , находятся в одной папке «Больница», удобнее при подключении не указывать весь путь к базе данных, а ввести только его полное имя. Тогда при переносе в дальнейшем папки «Больница» в другое место, никаких проблем с приложением возникать не будет!

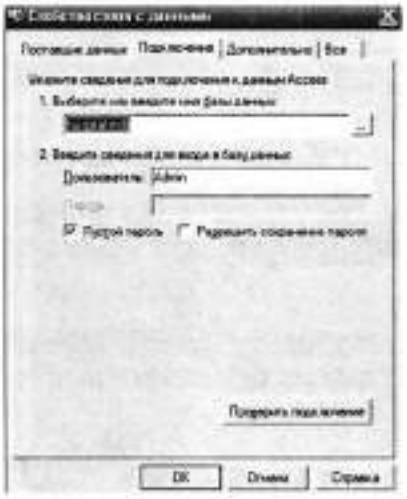


Рис. 2 Проверяем подключение.

Помещаем на форму компонент ADOTable с вкладки ADO палитры компонентов.
Устанавливаем свойство Connection (указываем наш ADOConnection1) – в свойстве TableName компонента ADOTable должно появиться окно 'Database Login' а затем в выпадающем списке появится список таблиц нашей базы данных.

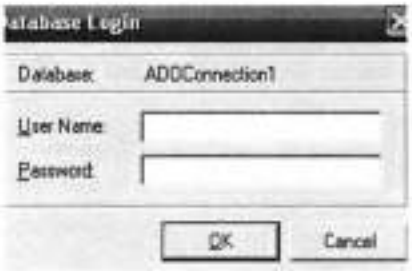


Рис. 3

Выбираем таблицу или запрос, которые нам необходимо вывести (обратите внимание, что для вывода каждой таблицы или запроса необходим отдельный компонент ADOTable или ADOQuery соответственно, но ADOConnection для всех один!).
Устанавливаем на форму компонент DataSource со вкладки Data Access. Свойство DataSet этого компонента меняем на ADOTable1.
Помещаем DBGrid с вкладки Data Controls и свойству DataSource присваиваем значение DataSource1.
Помещаем DBNavigator с вкладки Data Controls и свойству DataSource присваиваем значение DataSource1.
Активируем подключенную таблицу, задав свойству Active компонента ADOTable1 значение True, в результате чего в DBGrid отобразится таблица из нашей базы данных. Можно активировать таблицу с помощью кнопки:
procedure TForm1.Button1Click(Sender: TObject).

```
begin
adotable1.Open;
end;
```

Чтобы каждый раз при запуске программы на экран не выходило окно 'Database Login', установите свойство LoginPrompt компонента ADOConnection равным False.

III. Создание интерфейса приложения

По шаблону, предложенному выше, необходимо для каждой из таблиц определить свой ADO-компонент. Для этого размещаем на форме компоненты ADOTable1, …, ADOTable6, подключаем их к файлу базы данных (свойство Connection), задаем свойству TableName для этих компонентов следующие значения:

ADOTable1 – таблица «Пациенты»
ADOTable2 – таблица «№ палаты»
ADOTable3 – таблица «Врачи»
ADOTable4 – таблица «Палаты»

ADOTable5 – таблица «Препараты»
ADOTable6 – таблица «Лечение»

Размещаем на форме компоненты DataSource1, …, DataSource6, которые могут нам понадобиться для отображения или передачи данных. Для каждого из них определяем в свойстве DataSet соответствующую таблицу.

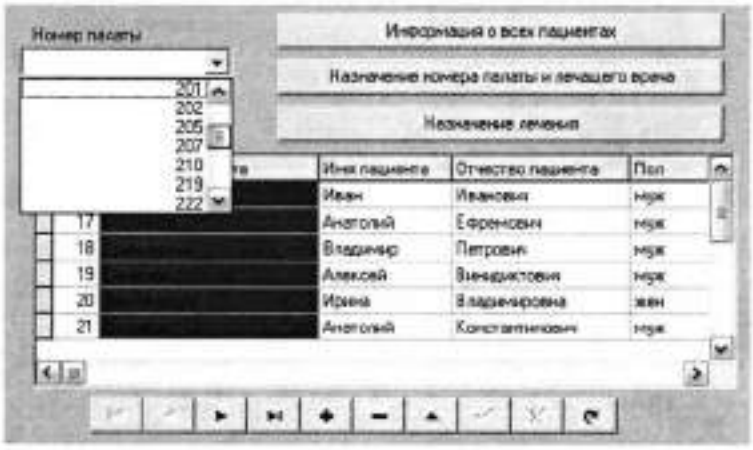


Рис. 4

Устанавливаем на форму компонент DBLookupComboBox (см. рис 5) с вкладки Data Controls (список будет использоваться для отображения номеров палат) и в свойствах ListSource, ListField и KeyField этого компонента задаём имя отображаемого столбца . Свойство DataSource компонента необходимо оставить пустым.

Отобразим с помощью компонента DBGrid содержимое таблицы «Пациенты» (присваиваем свойству DataSource значение DataSource1)

Обеспечим возможность модификации созданной таблицы с помощью компонента DBNavigator и проверим его работоспособность (если редактирование таблицы с помощью этого компонента оказалось затруднительным, то необходимо изменить свойство Options компонента DBGrid, задав dgEditing —> true)

Определим событие OnClick для компонента DBLookupComboBox, при выполнении которого выводится информация о пациентах, размещённых в палате с выбранным номером. Реализовать это событие можно двумя способами:

1 способ (с использованием фильтра):

Размещаем на форме компоненты DataSource7, который будет использован для отображения и передачи данных запроса ADOQuery1 (определяем в свойстве DataSet компонент ADOQuery1). Создаём SQL- запрос на вывод информации о пациентах путём использования свойства SQL —> Text компонета ADOQuery1 с помощью команды
form1.ADOQuery1.SQL.Text:='SELECT … FROM …..'.
Открываем SQL- запрос и осуществляем фильтрацию информации по заданному критерию с помощью команд:

```
form1.ADOTable3.Active:=false;

{ в качестве поставщика информации определяем компонент DataSource3,
который обращается к таблице "Врачи"}
form1.DBLookupListBox1.ListSource:= form1.DataSource3;
{выбираем поле "ФИО врача" для отображения информации в
компоненте DBLookupListBox1}
form1.DBLookupListBox1.ListField :='ФИО врача';
form1.DBLookupListBox1.KeyField :='ФИО врача';
{делаем активной таблицу "Врачи"}
form1.ADOTable3.Active:=true;
```

Рис. 5

2 способ (критерий отбора задаётся в разделе Where SQL-запроса):

Создаём SQL-запрос на вывод информации о пациентах находящихся в палате, номер которой задан в компоненте DBLookupComboBox, с помощью команды:

```
var dialog:string; id:integer;
begin
  form1.adoquery1.Close;  //закрываем SQL-запрос, используемый ранее
 {очищаем SQL-оператор в запросе adoquery1 и записываем новый SQL-запрос}
 form1.adoquery1.SQL.Clear;
 {Для добавления новой записи необходимо определить идентификатор ID,
 необходимый для заполнения уникального ключевого поля "Код врача". Для
 этого определим максимальное значение в поле "Код врача" таблицы
 "Врачи" и увеличим значение на единицу }
 form1.adoquery1.SQL.Add('Select max(Врачи.[Код врача]) FROM Врачи;');
 form1.adoquery1.Open;
 ID := form1.adoquery1.Fields.Fields[0].AsInteger + 1;
  //запускаем новый SQL-запрос
            //вводим новую запись с помощью диалогового окна (см.рис.14)
            dialog:=inputbox('ввод данных','введите фамилию врача','');
            {добавляем информацию в таблицу "Врачи": "ФИО врача" берем из
            диалогового окна,"код врача" - идентификатор ID}
            form1.ADOTable3.Insert;
            form1.ADOTable3.FieldByName('ФИО врача').asstring:=dialog;
            form1.ADOTable3.FieldByName('код врача').asinteger:=id;
            form1.ADOTable3.Post;
            showmessage('запись добавлена');
end;
```

Рис. 6

Определим событие OnClick для кнопки «Назначение номера палаты и лечащего врача».

При нажатии на эту кнопку открывается форма «Добавление информации», на которой отображаются «№ палаты» и «ФИО врача» пациента с заданным номером. Поиск этих данных осуществляется по схеме: Палаты.[Код пациента] —> Палаты. [Код палаты]

Палаты.[Код пациента] —> Палаты.[Код врача] —> Врачи.[Код врача] —> Врачи.[ФИО врача]

Содержание работы

1. Ознакомиться с теоретической частью методических указаний.
2. Выполнить примеры
3. Получить вариант задания от преподавателя.
4. Решить поставленную задачу
5. Представить работающую программу преподавателю.

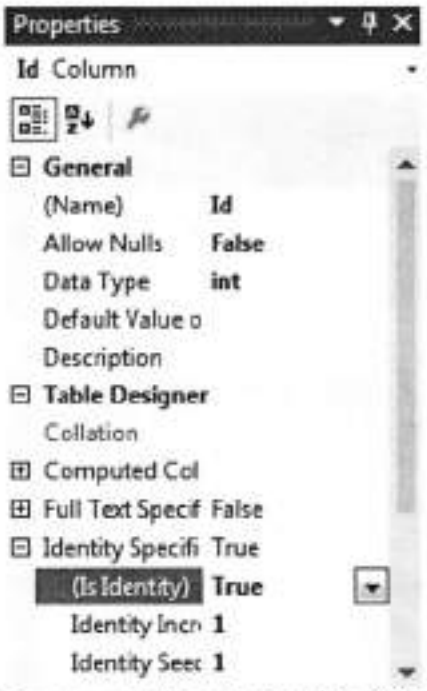
Лабораторная работа № 33 (2 часа)

Подключение к базе данных

Чтобы хранить данные, нам естественным образом нужна база данных. Как правило, в качестве базы данных используется MS SQL Server, на примере которого мы и посмотрим весь процесс создания БД и подключения к ней.
Мы можем создать базу данных прямо в проекте, либо же создать ее на сервере MS SQL. Для хранения баз данных проекте у нас предназначена папка **App_Data**. Для этого нажмем правой кнопкой мыши на папку App_Data и в появившемся контекстном меню выберем **Add-> New Item....** В появившемся окне добавления нового элемента выберем **SQL Server Database** и назовем новую базу данных Bookstore.mdf:

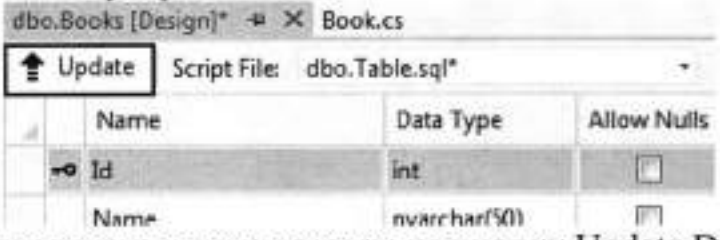
Мы можем создать базу данных равнозначным образом и на сервере. После этого база данных добавляется в проект, и мы можем увидеть ее в папке App_Data. Теперь в обозревателе баз данных (окно **Database Explorer**) мы можем подключиться к ней и создать таблицы, которые будут хранить данные.
Раскроем узел Bookstore.mdf и найдем узел Tables. Нажмем на этот узел правой кнопкой мыши и в появившемся меню выберем пункт *Add New Table*. И перед нами появится окно, в котором нам надо определить названия и типы столбцов новой таблицы. По соглашениям о наименованиях таблицы при работе с Entity Framework должны соответствовать имени модели. То есть, так как наша модель называется Book, то таблица будет называться Books. А Entity Framework автоматически распознает, что таблица Books соответствует классу Book.
Итак, создадим структуру таблицы:

Не забудьте установить ниже в окне Properties (в Visual Stidio 2010 - окно Column Properties) для столбца Id соответствующие параметры для первичного ключа:



После этого, если мы работаем с Visual Studio 2010, нам будет предложено просто ввести имя таблицы - введем имя Books, и затем таблица добавляется в БД.

Надо сгенерировать таблицу на основе заданного выше определения. Для этого нажмем на кнопку Update:



В появившемся диалоговом окне нажмем на кнопку Update Database. После этого в нашу базу данных добавляется только что сгенерированная таблица.

Подобным образом определим таблицу Purchases для модели Purchase:

Добавим в таблицу Books несколько записей:

dbo.Books [Data] | Book.cs

Max Rows: 1000

Id	Name	Author	Price
1	Война и мир	Л. Толстой	220
2	Чайка	А. Чехов	120
3	Отцы и дети	И. Тургенев	160
NULL	NULL	NULL	NULL

Теперь, во-первых, чтобы взаимодействовать с БД, нам нужен класс контекста данных, пусть это будет следующий класс BookContext:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Data.Entity;

namespace BookStore.Models
{
   public class BookContext : DbContext
        {
                public DbSet<Book> Books { get; set; }
                public DbSet<Purchase> Purchases { get; set; }
        }
}
```

Во-вторых, определим строку подключения к БД. Для этого откроем файл Web.config и добавим в конец секции **configuration** определение строки подключения..

Строка подключения будет выглядеть так:

```
<configuration>
.....................
   <connectionStrings>
```

```
    <add name="BookContext" connectionString="Data
Source=.\SQLEXPRESS;AttachDbFilename='|DataDirectory|\Bookstore.mdf';Inte
grated Security=True;User Instance=True"
 providerName="System.Data.SqlClient" />
  </connectionStrings>
</configuration>
```

Использование подстановки *|DataDirectory|* позволяет опустить полный физический путь к базе данных, которая хранится в папке App_Data.
Теперь мы можем получить содержимое таблицы Books в контроллере Home:

```
public class HomeController : Controller
{
  BookContext db = new BookContext();

  public ActionResult Index()
  {
    return View(db.Books);
  }
}
```

И вывести данные в представлении Index.cshtml:

```
@model IEnumerable<BookStore.Models.Book>
@{
  Layout = "~/Views/Shared/_Layout.cshtml";
}
  <div>
    <h3>Распродажа книг</h3>
    <table>
      <tr class="header"><td><p>Название книги</p></td>
        <td><p>Автор</p></td>
        <td><p>Цена</p></td><td></td>
      </tr>
      @foreach (BookStore.Models.Book b in Model)
      {
      <tr>
        <td><p>@b.Name</p></td>
        <td><p>@b.Author</p></td>
        <td><p>@b.Price</p></td>
        <td><p><a href="/Home/Buy/@b.Id">Купить</a></p></td>
      </tr>
      }
    </table>
  </div>
```

Закрытие подключения

Чтобы наверняка быть уверенным, что подключение к базе данных закрыто, следует вызывать метод Dispose у контекста данных:

```
protected override void Dispose(bool disposing)
{
   db.Dispose();
   base.Dispose(disposing);
}
```

Это переопределенная версия метода Dispose контроллера, которая вызывается при уничтожении объекта контроллера. В нее помещается вызов db.Dispose(), который уничтожает все связанные с контекстом данных ресурсы и подключения.

Содержание работы

1. Ознакомиться с теоретической частью методических указаний.
2. Выполнить примеры
3. Получить вариант задания от преподавателя.
4. Решить поставленную задачу
5. Представить работающую программу преподавателю.

Лабораторная работа № 34 (2 часа)

Построение отчетов с использованием подключения к базе данных

Понятие отчета

Под отчетом понимают средство для подготовки информации из БД к печати на принтере. С помощью отчетов удается формировать необходимую документацию, основанную на данных, с которыми работает информационная система. При этом для каждого вида документа создается отдельный отчет, являющийся как бы бланком документа. В процессе работы отчет многократно вызывается, автоматически заполняется данными и печатается на принтере.

Quick Reports – простейший набор компонентов для создания отчетов. Он хорошо подходит в тех случаях, когда требуется быстро построить простой отчет.

Rave Reports хорошо подходит для построения сложных отчетов. Этот набор компонентов теперь считается основным.

Набор компонентов Quick Reports

Чтобы набор **Quick Reports** стал доступен, необходимо установить соответствующий пакет (package) компонентов. Пакет представляет собой специальную динамическую библиотеку с расширением ***.bpl**, Для установки пакета следует выполнить пункт меню Component–Install Packages, нажать кнопку Add и далее выбрать требуемый пакет.

Установленный набор компонентов размещен на вкладке **QReport**.

Перечислим наиболее часто используемые компоненты, входящие в состав **Quick Reports**:

- **TQuickRep** – визуальный компонент, составляющий основу отчета. На нем размещаются все остальные элементы отчета;
- **TQRLabel** – текстовая метка. Компонент аналогичен стандартному компоненту TLabel, но приспособлен для использования в отчетах;
- **TQRDBText** – текстовое поле. Компонент предназначен для отображения в отчете значений заданного поля БД;
- **TQRExpr** – выражение. Компонент предназначен для отображения в отчете вычисляемых полей.

Пусть требуется создать отчет для печати информации о перечне всех товаров с указанием названия товара и его цены в рублях и долларах.

Порядок создания отчета следующий:

1. Создать новую форму, дав ей имя FormQuickRep, и сохранить ее в модуле

UQuickRep.pas.

2. Подключить к модулю UQuickRep.pas модуль главной формы приложения (будем считать, что он называется UMain.pas):

uses
Windows, ..., UMain;

Аналогично, подключим к модулю главной формы модуль UQuickRep.pas:

uses
Windows, ..., UQuickRep;

3. Разместить на форме FormQuickRep компонент типа **TQuickRep**, дав ему имя **QuickRep**.

4. Разместить на форме **FormQuickRep** компоненты **TADOConnection** и **TADOQuery**, задать им имена **ADOConnection** и QueryGoods. Связать компоненты между собой, подключить компонент **ADOConnection** к требуемой БД и занести в свойство SQL компонента QueryGoods необходимый запрос на выборку к БД, например:

SELECT * FROM Товары

5. Задать для компонента **QuickRep** следующие свойства:

- Набор данных, по которому составляется отчет:

DataSet := QueryGoods

Если это свойство не задать, то в отчете будет выводиться информация только об одной строке набора данных.

- Разрешить отображение полосы заголовка:

Bands.HasTitle := true

- Разрешить отображение полосы названия полей:

Bands.HasColumnHeader := true

- Разрешить отображение полосы отдельных записей:

Bands.HasDetail := true

6. Создать заголовок отчета, для чего разместить в центре полосы заголовка компонент типа **TQRLabel** и задать название отчета в его свойстве Caption, например, «Перечень товаров».

7. Создать шапку отчета, для чего разместить на полосе названий полей несколько компонентов типа **TQRLabel** и задать названия полей в их свойствах Caption, например, «Наименование»,«Цена в руб»,«Цена в $».

8. Разместить на полосе отдельных записей два компонента типа **TQRDBText** для вывода наименования товара и цены товара в рублях. Задать

для обоих компонентов имя набора данных QueryGoods в их свойствах DataSet и названия соответствующих полей набора данных в свойствах **DataField**.

9. Разместить на полосе отдельных записей компонента типа **TQRExpr**. Задать свойство Expression, записав в него следующее выражение для вычисления цены в долларах:

QueryGoods.Цена /30

При необходимости можно использовать мастер для построения выражений, позволяющий выбирать поля таблиц БД (кнопка **Database Field**), стандартные функции (кнопка Function) и переменные (кнопка Variable).

10. Разместить на главной форме кнопку, дав ей название **"QuickRep"** и написав для нее обработчик, вызывающий предварительный просмотр отчета:

```
FormQuickRep.QuickRep.Preview;
```

При необходимости печати отчета обработчик выглядел бы следующим образом:

```
FormQuickRep.QuickRep.Print;
```

11. Откомпилировать и запустить приложение, убедиться в его работоспособности.

12. Улучшить внешний вид отчета, изменяя шрифты, местоположение полей и другие настройки компонентов отчета.

Набор компонентов Rave Reports

Набор компонентов размещен на вкладке Rave. В его состав входят следующие основные компоненты:

TRvProject – компонент отчета. Обеспечивает загрузку ранее созданного в визуальной среде Rave Reports отчета из файла с расширением rav.

TRvSystem – компонент управления отчетом. Обеспечивает работу приложения с отчетом. Взаимодействуя с компонентом отчета, с одной стороны, и сервером отчета Rave Reports, с другой, этот компонент обеспечивает просмотр и печать отчетов.

TRvCustomConnection, TRvDataSetConnection, TRvTableConnection, TRvQueryConnection – группа компонентов соединения с источниками данных, предназначенные для подключения отчетов к различным источникам данных.

TRvNDRWriter, TRvRenderHTML, TRvRenderPreview, TRvRenderRTF, TRvRenderPrinter, TRvRenderText, TRvRenderPDF – группа компонентов преобразования данных, позволяющих конвертировать отчеты из формата

данных Rave Reports в другие форматы (текстовый, PDF, HTML, RTF), а также распечатывать или просматривать отчеты.
Основой отчета является файл отчета с расширением rav, который создается в визуальной среде разработки Rave Reports и называется проектом отчета. Созданный проект отчета необходимо связать с приложением. Для этого используется компонент TRvProject, который обеспечивает представление отчета в приложении. Для просмотра и печати отчета используется компонент TRvSystem, котоый взаимодействует непосредственно с ядром Rave Reports. При компоновке приложения ядро Rave Reports автоматически включается в его состав. Схема взаимодействия компонентов Rave Reports приведена на рисунке 2.

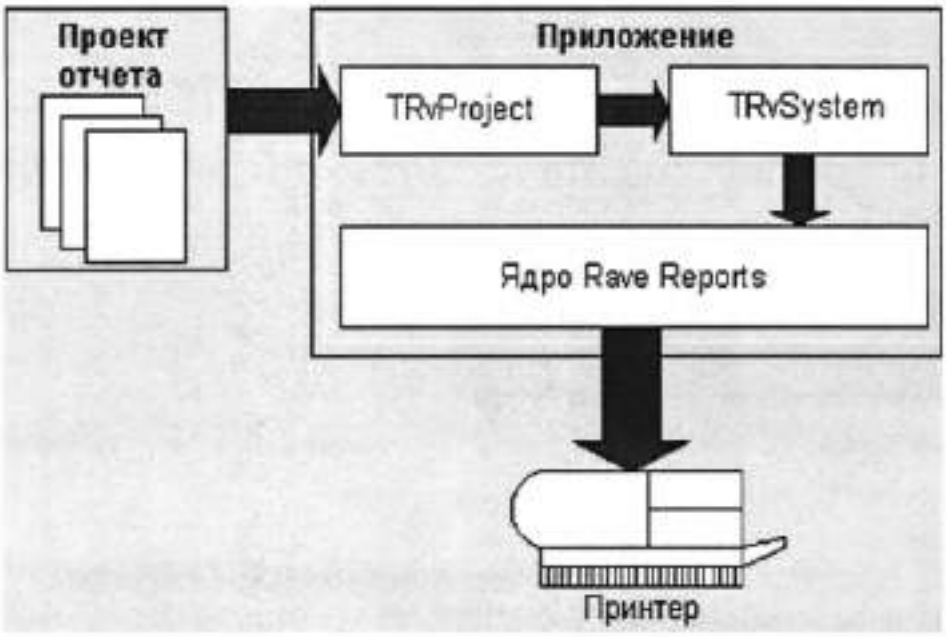


Рисунок 2 – Взаимодействие компонентов Rave Reports

Для того чтобы приложение могло выполнять функции печати отчетов, разработчик должен выполнить следующий набор операций.
Создадим отчет аналогичный рассмотренному ранее, используя набор компонентов Rave Reports. Порядок создания отчета следующий:
1. Разместить на главной форме приложения следующие компоненты: TRvProject, TRvSystem, TRvDataSetConnection, TADOConnection и TADOQuery, задав им имена, соответственно: RvProject, RvSystem, RvDataSetConnection, ADOConnection и QueryRaveRep. С помощью свойства Engine связать компонент RvProject с компонентом RvSystem. Настроить компонент RvDataSetConnection, подключив его к компоненту QueryRaveRep с помощью свойства DataSet. Связать компоненты QueryRaveRep и ADOConnection, подключить компонент ADOConnection к требуемой БД. Для компонента QueryRaveRep задать следующий текст запроса:

SELECT Наименование as Name,
Цена as PriceRub,
Цена /30 as PriceUSD
FROM Товары

2. При помощи визуальной среды разработки Rave Reports создать проект отчета. Для этого выполнить двойной щелчок по компоненту RvProject или выбрать пункт меню Tools–Rave Designer. В открывшемся редакторе отчетов выполним пункт меню File–New Data Objects, чтобы настроить соединение с БД. В открывшемся окне выберем пункт меню **Direct Data View**

В следующем окне выберем ранее созданный компонент соединения **RvDataSetConnection** и нажмем кнопку **Finish**.

Далее для упрощения создания отчета воспользуемся мастером построения отчетов. Для этого выполним пункт меню **Tools–Report Wizards–Simple Table**. В ходе выполнения мастера потребуется выбрать поля, включаемые в отчет, и порядок следования полей, а также задать текст заголовка отчета.

Теперь можно отредактировать сгенерированный отчет, задав необходимые настройки шрифтов. После окончания редактирования необходимо сохранить отчет, выбрав пункт меню **File–Save As** и задав имя отчета, например, **goods.rav**. Для предварительного просмотра отчета следует нажать **F9**.

Далее следует закрыть окно **Rave Reports**.

3. В компоненте **RvProject** задать свойство ProjectFile, указав в нем имя ранее созданного файла проекта отчета (**goods.rav**).

4. Разместить на главной форме кнопку, дав ей название "QuickRave" и написав для нее обработчик, вызывающий предварительный просмотр отчета или выполняющий его печать:

RvProject.Execute;

Чтобы задать, какое действие будет выполнялся по умолчанию, следует задать в компоненте **RvSystem** свойство **DefaultDest**:

- предварительный просмотр:

DefaultDest := rdPreview;

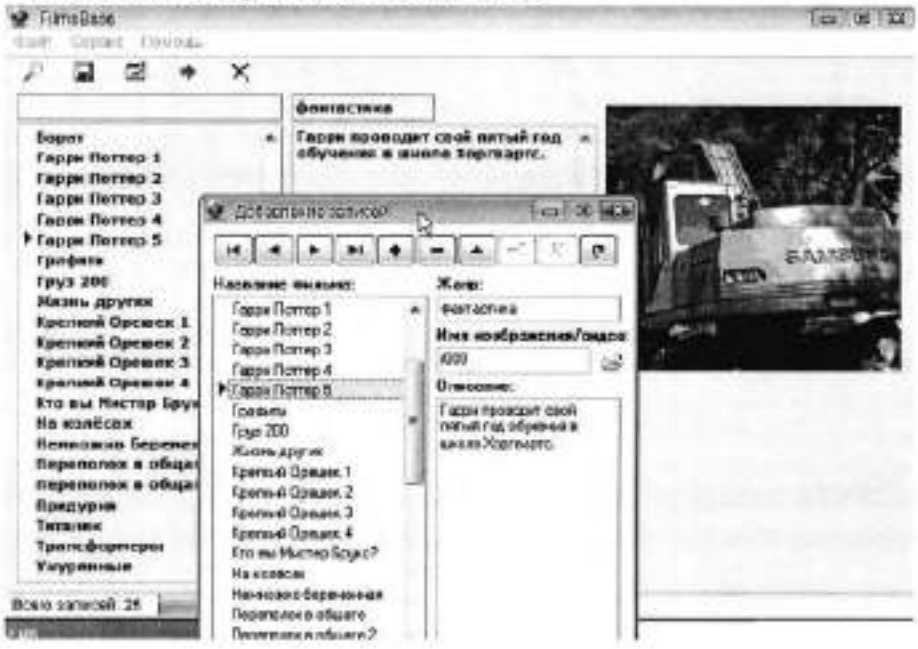
- печать:

DefaultDest := rdPrinter.

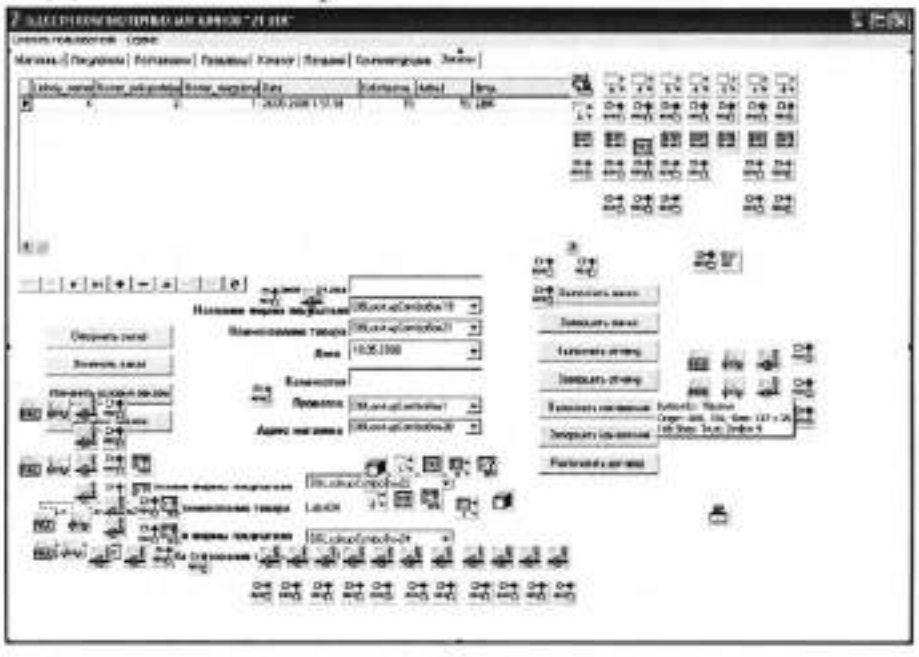
5. Откомпилировать и запустить приложение, убедиться в его работоспособности.

Исходники:

• База данных фильмов FilmsBase



• БД сеть компьютерных магазинов



Содержание работы

1. Ознакомиться с теоретической частью методических указаний.
2. Выполнить примеры
3. Получить вариант задания от преподавателя.
4. Решить поставленную задачу
5. Представить работающую программу преподавателю.