МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

НЕФТЕЮГАНСКИЙ ИНДУСТРИАЛЬНЫЙ КОЛЛЕДЖ
(филиал) федерального государственного бюджетного образовательного
учреждения высшего образования «Югорский государственный университет»
(НИК (филиал) ФГБОУ ВО «ЮГУ»)

УТВЕРЖДАЮ
Заместитель директора по УВР
_____ Гарбар О.В.
«09» сентября 2021 г.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по выполнению практических работ

ОП.04. Основы алгоритмизации и программирования

09.02.07 Информационные системы и программирование

г. Нефтеюганск
2021

РАССМОТРЕНО:
Предметной (цикловой)
комиссией МиЕНД
Протокол № 1 от 09.09.2021г.
Председатель ПЦК
_____ Ю.Г. Шумскис

СОГЛАСОВАНО:
заседанием Методсовета
протокол № 1 от 16.09.2021г.
Председатель методсовета
_____ Н.И. Савватеева

Методические указания по выполнению практических работ разработаны в соответствии с рабочей программой ОП.04. Основы алгоритмизации и программирования по специальности 09.02.07 Информационные системы и программирование.

Разработчик:

Игнатенко Е.С. – преподаватель ИндИ (филиала) ФГБОУ ВО «ЮГУ».

СОДЕРЖАНИЕ

Пояснительная записка

Методические указания по выполнению практических работ по учебной дисциплине ОП.04 Основы алгоритмизации и программирования разработаны в соответствии с рабочей программой учебной дисциплины и предназначены для приобретения необходимых практических навыков и закрепления теоретических знаний, полученных обучающимися при изучении учебной дисциплины, обобщения и систематизации знаний перед экзаменом.

Освоение содержания учебной дисциплины «Основы алгоритмизации и программирования» во время выполнения практических работ обеспечивает достижение обучающимися следующих **результатов:**

Код ПК, ОК	Умения	Знания
ОК 1 ОК 2 ОК 4 ОК 5 ОК 9 ОК 10 ПК 1.1- ПК 1.5 ПК 2.4, 2.5	Разрабатывать алгоритмы для конкретных задач. Использовать программы для графического отображения алгоритмов. Определять сложность работы алгоритмов. Работать в среде программирования. Реализовывать построенные алгоритмы в виде программ на конкретном языке программирования. Оформлять код программы в соответствии со стандартом кодирования. Выполнять проверку, отладку кода программы.	Понятие алгоритмизации, свойства алгоритмов, общие принципы построения алгоритмов, основные алгоритмические конструкции. Эволюцию языков программирования, их классификацию, понятие системы программирования. Основные элементы языка, структуру программы, операторы и операции, управляющие структуры, структуры данных, файлы, классы памяти. Подпрограммы, составление библиотек подпрограмм Объектно-ориентированную модель программирования, основные принципы объектно-ориентированного программирования на примере алгоритмического языка: понятие классов и объектов, их свойств и методов, инкапсуляция и полиморфизма, наследования и переопределения

В соответствии с рабочей программой учебной дисциплины «Основы алгоритмизации и программирования» практические работы сгруппированы в конце четвертого семестра. Целесообразность данной группировки обусловлена необходимостью обобщения и систематизации знаний перед экзаменом.

Рабочая программа учебной дисциплины предусматривает проведение практических работ в объеме 84 часов.

Порядок выполнения практической работы

- записать название работы, ее цель в тетрадь;
- выполнить основные задания в соответствии с ходом работы;
- выполнить индивидуальные задания.

Рекомендации по оформлению практической работы

Задания выполняются обучающимися по шагам. Необходимо строго придерживаться порядка действий, описанного в практической работе

Результаты выполнения практических заданий необходимо сохранять в своей папке на компьютере или USB – накопителе.

В случае пропуска занятий обучающийся осваивает материал самостоятельно в свободное от занятий время и сдает практическую работу с пояснениями о выполнении.

Критерии оценки практической работы

- наличие цели выполняемой работы, выполнение более половины основных заданий (удовлетворительно);

- наличие цели выполняемой работы, выполнение всех основных и более половины дополнительных заданий (хорошо);
- наличие цели выполняемой работы, выполнение всех основных и индивидуальных заданий (отлично).

Перечень практических работ

№ п/п	Наименование практических работ	Кол-во часов
	Практическая работа № 1 «Знакомство со средой программирования»	2
	Практическая работа № 2 «Составление программ линейной структуры»	2
	Практическая работа № 3 «Составление программ разветвляющейся структуры»	2
	Практическая работа № 4 «Составление программ циклической структуры»	2
	Практическая работа № 5 «Обработка одномерных массивов. Обработка двумерных массивов»	4
	Практическая работа №6 «Работа со строками»	2
	Практическая работа № 7 «Работа с данными типа множество»	2
	Практическая работа №8 «Файлы последовательного доступа. Типизированные файлы»	2
	Практическая работа №9 «Нетипизированные файлы»	2
	Практическая работа № 10 «Организация процедур. Организация функций»	2
	Практическая работа №11 «Применение рекурсивных функций»	2
	Практическая работа №12 «Программирование модуля»	2
	Практическая работа №13 «Создание библиотеки подпрограмм»	2
	Практическая работа №14 «Классы ООП: виды, назначение, свойства, методы, события. Объявления класса»	2
	Практическая работа №15 «Создание наследованного класса»	2
	Практическая работа №16 «Перегрузка методов»	2
	Практическая работа №17 «Использование указателей для организации связанных списков»	2
	Практическая работа №18 «Изучение интегрированной среды разработчика»	2
	Практическая работа №19 «События компонентов (элементов управления), их сущность и назначение»	2
	Практическая работа №20 «Создание проекта с использованием компонентов ввода и отображения чисел, дат и времени»	2
	Практическая работа №21 «Создание проекта с использованием компонентов для работы с текстом»	4
	Практическая работа №22 «Создание процедур на основе событий»	2
	Практическая работа №23 «Создание проекта с использованием кнопочных компонентов»	4
	Практическая работа №24 «Создание проекта с использованием компонентов стандартных диалогов и системы меню»	4
	Практическая работа №25 «Разработка функциональной схемы работы приложения»	4
	Практическая работа №26 «Разработка оконного приложения с несколькими формами»	4
	Практическая работа №27 «Разработка игрового приложения»	4
	Практическая работа №28 «Создание процедур обработки событий. Компиляция и запуск приложения»	4
	Практическая работа №29 «Разработка интерфейса приложения»	4
	Практическая работа №30 «Тестирование, отладка приложения»	4
	Практическая работа №31 «Программирование приложений»	4
Итого		**84**

Практическая работа № 1 «Знакомство со средой программирования»

Цель работы: изучить среду быстрой разработки приложений Visual Studio. Научиться размещать и настраивать внешний вид элементов управления на форме.

Интегрированная среда разработчика Visual Studio

Среда Visual Studio визуально реализуется в виде одного окна с несколькими панелями инструментов. Количество, расположение, размер и вид панелей может меняться программистом или самой средой разработки в зависимости от текущего режима работы среды или пожеланий программиста, что значительно повышает производительность работы.

При запуске Visual Studio появляется начальная страница со спи-ском последних проектов, а также командами Создать проект и От-крыть проект. Нажмите ссылку Создать проект или выберите в меню Файл команду Создать проект, на экране появится диалог для создания нового проекта (рис. 1.1).

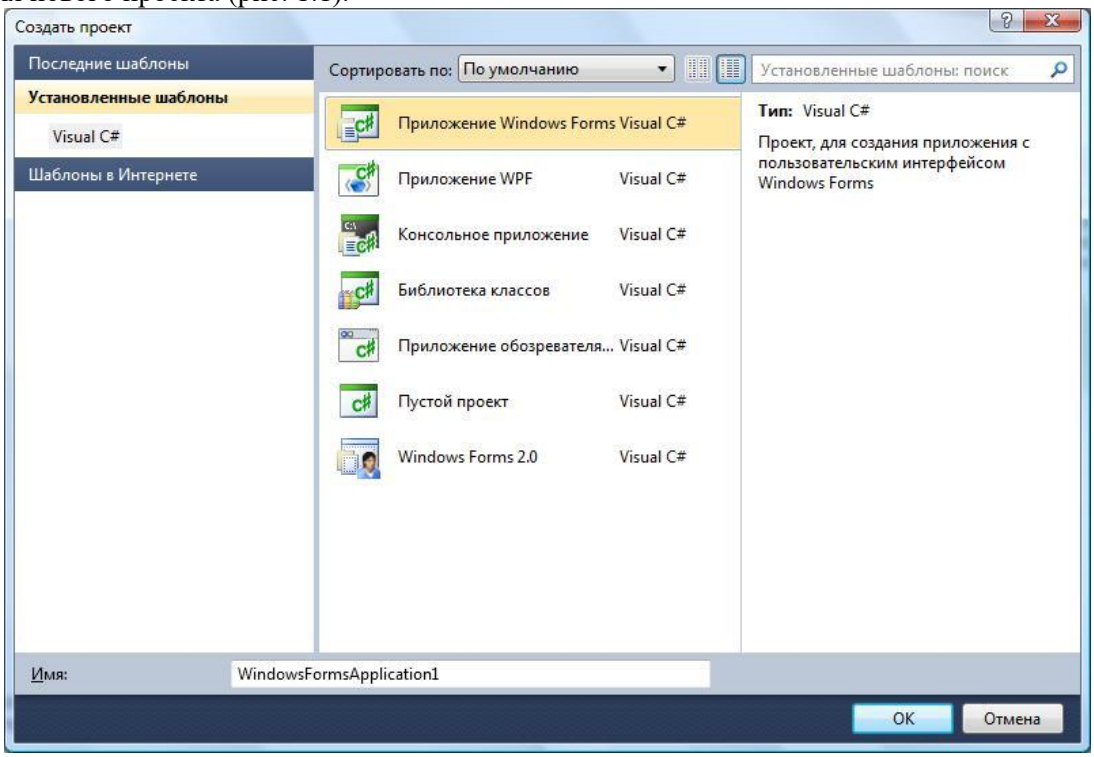


Рис. 1.1. Диалог создания нового проекта

Слева в списке шаблонов приведены языки программирования, которые поддерживает данная версия Visual Studio: убедитесь, что там выделен раздел Visual C#. В средней части приведены типы проектов, которые можно создать. В наших лабораторных работах будут использоваться два типа проектов:

1. Приложение Windows Forms – данный тип проекта позволяет создать полноценное приложение с окнами и элементами управления (кнопками, полями ввода и пр.) Такой вид приложения наиболее привычен большинству пользователей.
2. Консольное приложение – в этом типе проекта окно представляет собой текстовую консоль, в которую приложение может выводить тексты или ожидать ввода информации пользователя. Консольные при-ложения часто используются для вычислительных задач, для которых не требуется сложный или красивый пользовательский интерфейс.

Выберите в списке тип проекта «Приложение Windows Forms», в поле «имя» внизу окна введите желаемое имя проекта (например, MyFirstApp) и нажмите кнопку ОК. Через несколько секунд Visual Studio создаст проект и Вы сможете увидеть на экране картинку, подобную представленной на рис. 1.2.

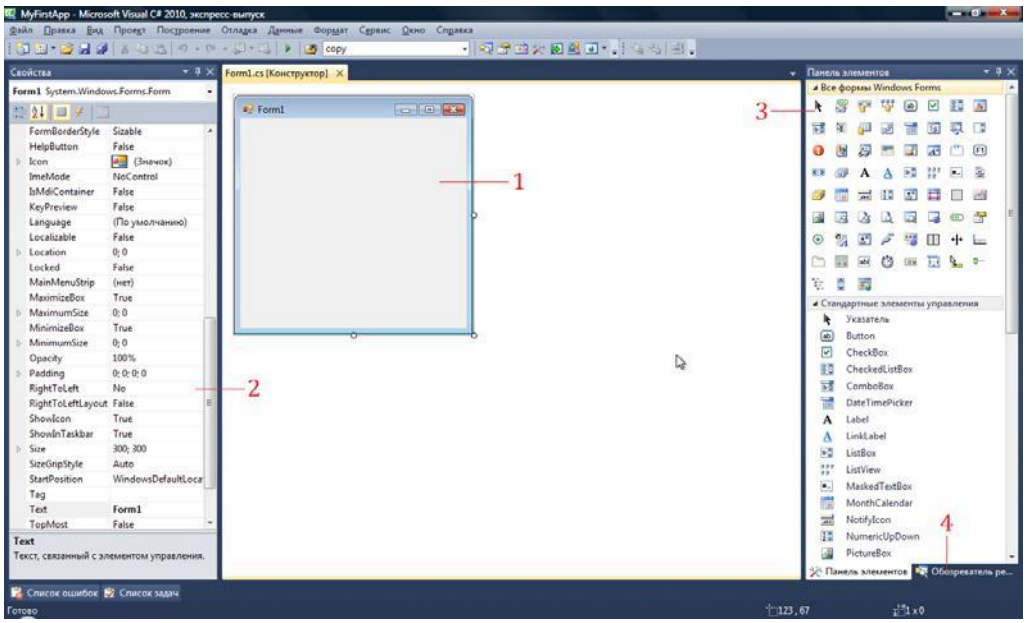


Рис. 1.2. Главное окно Visual Studio

В главном окне Visual Studio присутствует несколько основных элементов, которые будут помогать нам в работе. Прежде всего, это **форма**

(1) – будущее окно нашего приложения, на котором будут размещаться элементы управления. При выполнении программы помещенные элемен-ты управления будут иметь тот же вид, что и на этапе проектирования.

Второй по важности объект – это **окно свойств** (2), в котором приведены все основные свойства выделенного элемента управления или окна. С помощью кнопки можно просматривать свойства элемента управления, а кнопка переключает окно в режим просмотра событий. Чтобы было удобнее искать нужные свойства, можно отсортировать их по алфавиту с помощью кнопки . Если этого окна на экране нет, его можно активировать в меню Вид → Окно свойств (иногда этот пункт вложен в подпункт Другие окна).

Сами элементы управления можно брать на **панели элементов** (3). Все элементы управления разбиты на логические группы, что облегчает поиск нужных элементов. Если панели нет на экране, ее нужно активировать командой Вид → Панель элементов.

Наконец, **обозреватель решений** (4) содержит список всех файлов, входящих в проект, включая добавленные изображения и служебные файлы. Активируется командой Вид → Обозреватель решений.

Указанные панели могут уже присутствовать на экране, но быть скрытыми за другими панелями или свернуты к боковой стороне окна.

В этом случае достаточно щелкнуть на соответствующем ярлычке, что-бы вывести панель на передний план.

Окно текста программы предназначено для просмотра, написания и редактирования текста программы. Переключаться между формой и текстом программы можно с помощью команд Вид → Код и Вид → Конструктор. При первоначальной загрузке в окне текста программы находится текст, содержащий минимальный набор операторов для нормального функционирования пустой формы в качестве Windows-окна. При помещении элемента управления в окно формы, текст программы автоматически дополняется описанием необходимых для его работы библиотек стандартных программ (раздел using) и переменных для дос-тупа к элементу управления (в скрытой части класса формы).

Программа на языке C# составляется как описание алгоритмов, которые необходимо выполнить, если возникает определенное событие, связанное с формой (например, щелчок «мыши» на кнопке – событие Click, загрузка формы – Load). Для каждого обрабатываемого в форме события, с помощью окна свойств, в тексте программы организуется метод, в котором программист записывает на языке C# требуемый алгоритм.

Настройка формы

Настройка формы начинается с настройки размера формы. С помощью мыши, «захватывая» одну из кромок формы или выделенную строку заголовка, отрегулируйте нужные размеры формы.

Для настройки будущего окна приложения задаются свойства фор-мы. Для задания любых свойств формы и элементов управления на форме используется окно свойств.

Новая форма имеет одинаковые имя (Name) и заголовок (Text) – Form1.

Для изменения заголовка щелкните кнопкой мыши на форме, в ок-не свойств найдите и щелкните мышкой на строчке с названием Text.

В выделенном окне наберите «Лаб. раб. N1. Ст. гр. 7А62 Иванов А. А». Для задания цвета окна используйте свойство BackColor.

Размещение элементов управления на форме

Для размещения различных элементов управления на форме используется панель элементов. Панель элементов содержит элементы управления, сгруппированные по типу . Каждую группу элементов управления можно свернуть, если она в настоящий момент не нужна. Для выполнения лабораторных работ потребуются элементы управления из группы Стандартные элементы управления.

Щелкните на элементе управления, который хотите добавить, а за-тем щелкните в нужном месте формы – элемент появится на форме. Эле-мент можно перемещать по форме, схватившись за него левой кнопкой мышки (иногда это можно сделать лишь за появляющийся при нажатии на элемент квадрат со стрелками). Если элемент управления позволяет изменять размеры, то на соответствующих его сторонах появятся белые кружки, ухватившись за которые и можно изменить размер. После размещения элемента управления на форме, его можно выделить щелчком мыши и при этом получить доступ к его свойствам в окне свойств.

Размещение строки ввода

Если необходимо ввести из формы в программу или вывести на форму информацию, которая вмещается в одну строку, используют окно однострочного редактора текста, представляемого элементом управления TextBox.

В данной программе с помощью однострочного редактора будет вводиться имя пользователя.

Выберите на панели элементов пиктограмму с названием TextBox, щелкните мышью в том месте формы, где вы хотите ее поставить. Захватив его мышкой, отрегулируйте размеры элемента управления и его положение. Обратите внимание на то, что теперь в тексте программы можно использовать переменную textBox1, которая соответствуют добавленному элементу управления. В этой переменной в свойстве Text будет содержаться строка символов (тип string) и отображаться в соответствующем окне TextBox.

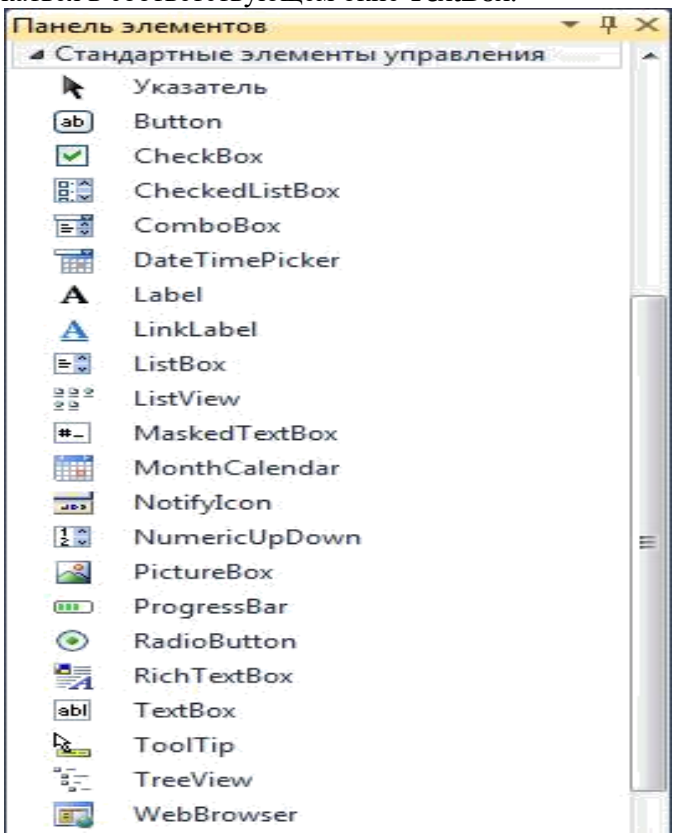


Рис. 1.3. Панель элементов

С помощью окна свойств установите шрифт и размер символов, отражаемых в строке TextBox (свойство Font).

Размещение надписей

На форме могут размещаться пояснительные надписи. Для нанесения таких надписей на форму используется элемент управления Label. Выберите на панели элементов пиктограмму с названием Label, щелкните на ней мышью. После этого в нужном месте формы щелкните мышью, появится надпись label1 . Щелкнув на ней мышью, отрегулируйте размер и, изменив свойство Text в окне свойств, вве-дите строку, например «Введите свое имя:», а также выберите размер символов (свойство Font).

Обратите внимание, что в тексте программы теперь можно обращаться к новой переменной типа Label. В ней хранится пояснительная строка, которую можно изменять в процессе работы программы.

Написание программы обработки события

С каждым элементом управления на форме и с самой формой могут происходить события во время работы программы. Например, с кноп-кой может произойти событие – нажатие кнопки, а с окном, которое проектируется с помощью формы, может произойти ряд событий: соз-дание окна, изменение размера окна, щелчок мыши на окне и т. п. Эти события могут обрабатываться в программе. Для обработки таких событий необходимо создать обработчики события – специальный метод. Для создания обработчика события существует два способа.

Первый способ – создать обработчик для события по умолчанию (обычно это самое часто используемое событие данного элемента управления). Например, для кнопки таким образом создается обработ-чик события нажатия.

Написание программы обработки события нажатия кнопки

Поместите на форму кнопку, которая описывается элементом управления Button. С помощью окна свойств измените заголовок (Text) на слово « Привет » или другое по вашему желанию. Отрегулируйте положение и размер кнопки.

После этого два раза щелкните мышью на кнопке, появится текст программы:

```
private void button1_Click(object sender, EventArgs e)
{
}
```

Это и есть обработчики события нажатия кнопки. Вы можете до-бавлять свой код между скобками { }. Например, наберите:

```
MessageBox.Show("Привет, " + textBox1.Text + "!");
```

Написание программы обработки события загрузки формы

Второй способ создания обработчика события заключается в выбо-ре соответствующего события для выделенного элемента на форме. При этом используется окно свойств и его закладка  . Рассмотрим этот спо-соб. Выделите форму щелчком по ней, чтобы вокруг нее появилась рам-ка из точек. В окне свойств найдите событие Load. Щелкнете по данной строчке дважды мышкой. Появится метод:

```
private void Form1_Load(object sender, EventArgs e)
{
}
```

Между скобками { } вставим текст программы:

```
BackColor = Color.AntiqueWhite;
```

Каждый элемент управления имеет свой набор обработчиков событий, однако некоторые из них присущи большинству элементов управления. Наиболее часто применяемые события описаны ниже:

- Activated: форма получает это событие при активации.
- Load: возникает при загрузке формы. В обработчике данного события следует задавать действия, которые должны происходить в мо-мент создания формы, например установка начальных значений.
- KeyPress: возникает при нажатии кнопки на клавиатуре. Параметр e.KeyChar имеет тип char и содержит код нажатой клавиши (клавиша Enter клавиатуры имеет код #13, клавиша Esc – #27 и т. д.). Обычно это событие используется в том случае, когда необходима реакция на нажатие одной из клавиш.
- KeyDown: возникает при нажатии клавиши на клавиатуре. Обработчик этого события получает информацию о нажатой клавише и со-стоянии клавиш Shift, Alt и Ctrl, а

также о нажатой кнопке мыши. Информация о клавише передается параметром e.KeyCode, который представляет собой перечисление Keys с кодами всех клавиш, а информацию о клавишах-модификаторах Shift и др. можно узнать из параметра e.Modifiers.

- KeyUp: является парным событием для KeyDown и возникает при отпускании ранее нажатой клавиши.
- Click: возникает при нажатии кнопки мыши в области элемента управления.
- DoubleClick: возникает при двойном нажатии кнопки мыши в области элемента управления.

Важное примечание! Если какой -то обработчик был добавлен по ошибке или больше не нужен, то для его удаления нельзя просто удалить программный код обработчика! Сначала нужно удалить строку с именем обработчика в окне свойств на закладке ⚡. В противном случае программа может перестать компилироваться и да-же отображать форму в дизайнере Visual Studio.

Запуск и работа с программой

Запустить программу можно, выбрав в меню Отладка команду Начать отладку. При этом происходит трансляция и, если нет ошибок, компоновка программы и создание единого загружаемого файла с расширением .exe. На экране появляется активное окно программы.

Если в программе есть ошибки, то в окне Список ошибок появятся найденные ошибки, а программа обычно не запускается. Иногда, одна-ко, может быть запущена предыдущая версия программы, в которой еще нет последних изменений: чтобы этого не происходило, нужно настройках Visual Studio установить опцию показа окна ошибок и за-претить запуск при наличии ошибок (рис. 1.4 и 1.5).

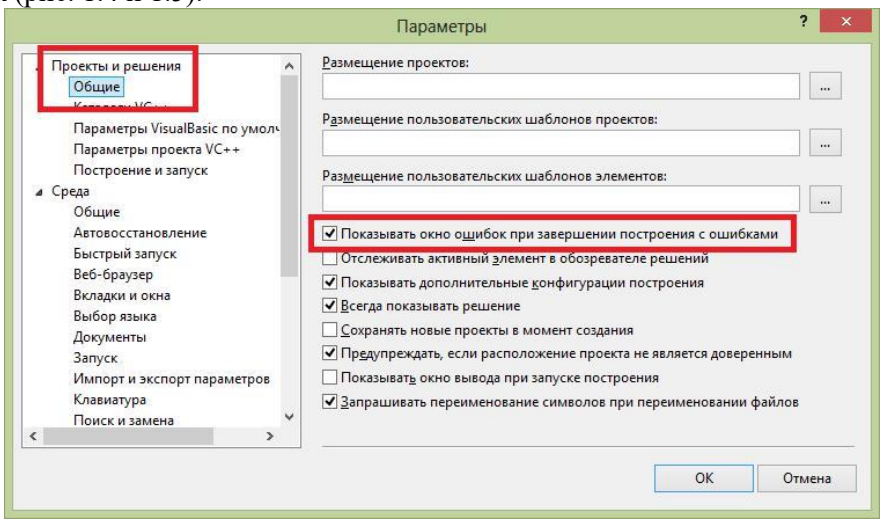


Рис. 1.4. Опция показа сообщений об ошибках

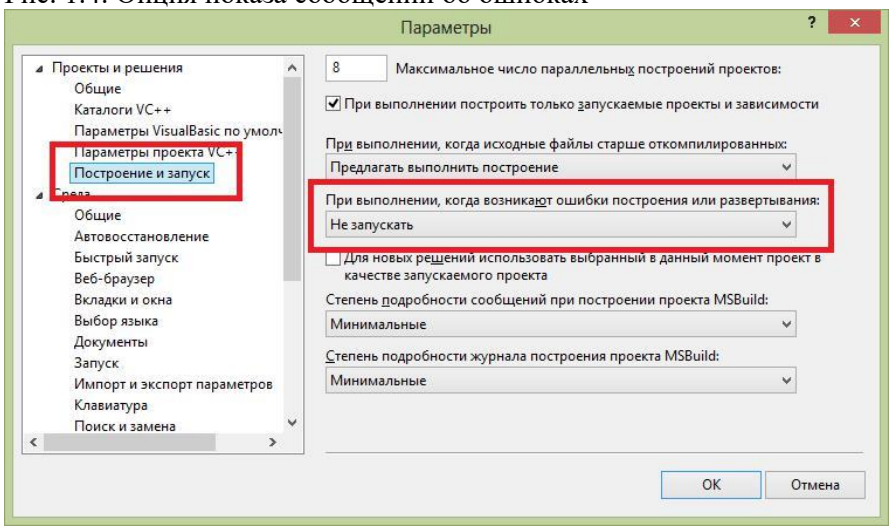


Рис. 1.5. Опция отключения запуска предыдущей версии при ошибках построения

Для завершения работы программы и возвращения в режим проектирования формы не забудьте закрыть окно программы!

Динамическое изменение свойств

Свойства элементов на окне могут быть изменены динамически во время выполнения программы. Например, можно изменить текст надпи-си или цвет формы. Изменение свойств происходит внутри обработчика события (например, обработчика события нажатия на кнопку). Для это-го используют оператор присвоения вида:

<имя элемента>.<свойство> = <значение>;

Например:

label1.Text = "Привет";

<Имя элемента> определяется на этапе проектирования формы, при размещении элемента управления на форме. Например, при размещении на форме ряда элементов TextBox, эти элементы получают имена textBox1, textBox2, textBox3 и т. д., которые могут быть заменены в ок-не свойств в свойстве (Name) для текущего элемента. Допускается использование латинских или русских символов, знака подчеркивания цифр (цифра не должна стоять в начале идентификатора). Список свойств для конкретного элемента можно посмотреть в окне свойств, а также в приложении к данным методическим указаниям.

Если требуется изменить свойства формы, то никакое имя элемента перед точкой вставлять не нужно, как и саму точку. Например, чтобы задать цвет формы, нужно просто написать:

BackColor = Color.Green;

Задание на практическую работу

По указанию преподавателя выберите свое индивидуальное задание. Уточните условие задания, количество, наименование, типы исходных данных. Прочтите в Приложении 1 описание свойств и описание элементов управления Form, Label, TextBox, Button. С помощью окна свойств установите первоначальный цвет формы, шрифт выводимых символов.

Индивидуальные задания

1. Разместите на форме четыре кнопки (Button). Сделайте на кнопках следующие надписи: «красный», «зеленый», «синий», «желтый». Создайте четыре обработчика события нажатия на данные кнопки, которые будут менять цвет формы в соответствии с текстом на кнопках.
2. Разместите на форме две кнопки (Button) и одну метку (Label). Сделайте на кнопках следующие надписи: «привет», «до свидания». Создайте обработчики события нажатия на данные кнопки, которые будут менять текст метки на слова, написанные на кнопках. Создайте обработчик события создания формы (Load), который будет устанавливать цвет формы и менять текст метки на строку «Начало работы».
3. Разместите на форме ряд кнопок (Button) напротив каждой поле ввода (TextBox) и одну метку (Label). Создайте обработчики события нажатия на данные кнопки, которые будут менять текст в метке. Текст метке берется из поля ввода напротив нажимаемой кнопки.
4. Разместите на форме ряд кнопок (Button), и одно поле ввода (TextBox). Создайте обработчики события нажатия на данные кнопки, которые будут менять текст на нажатой кнопке. Текст на кнопке берется из поля ввода.
5. Разместите на форме ряд кнопок (Button) и ряд меток (Label). Создайте обработчики события нажатия на данные кнопки, которые будут менять цвет двух меток. Создайте обработчик события нажатия кнопки мыши на форме (Click), который будет устанавливать цвет всех меток в белый.
6. Разместите на форме ряд кнопок (Button) и ряд меток (Label). Создайте обработчик события создания формы (Load), который будет делать все метки невидимыми. Создайте обработчики события нажатия на кнопки, которые будут менять свойство метки Visible, тем самым делать их видимыми.
7. Разместите на форме ряд кнопок (Button), напротив каждой по-ле ввода (TextBox). Создайте обработчики события нажатия на данные кнопки, которые будут менять заголовок окна. Текст в заголовке берется из поля ввода напротив нажимаемой кнопки.
8. Разместите на форме две кнопки (Button) и одну метку (Label). Сделайте на кнопках следующие надписи: «скрыть», «показать». Создайте обработчики события нажатия на данные кнопки, которые будут скрывать или показывать метку. Создайте обработчик события создания формы (Load), который будет устанавливать цвет формы и менять текст метки на строку «Начало работы».

9. Разместите на форме три кнопки (Button) и одно поле ввода (TextBox). Сделайте на кнопках следующие надписи: «скрыть», «показать», «очистить». Создайте обработчики события нажатия на данные кнопки, которые будут скрывать или показывать поле ввода. При нажатии на кнопку «очистить» текст из поля ввода должен быть удален.

10. Разместите на форме две кнопки (Button) и одно поле ввода (TextBox). Сделайте на кнопках следующие надписи: «заполнить», «очистить». Создайте обработчики события нажатия на данные кнопки, которые будут очищать или заполнять поле ввода знаками «******». Создайте обработчик события создания формы (Load), который будет устанавливать цвет формы и менять текст в поле ввода на строку «+++++».

11. Разработайте игру, которая заключается в следующем. На форме размещены пять кнопок (Button). При нажатии на кнопку некоторые кнопки становятся видимыми, а другие – невидимыми. Цель игры – скрыть все кнопки.

12. Разработайте игру, которая заключается в следующем. На форме размещены четыре кнопки (Button) и четыре метки (Label). При нажатии на кнопку часть надписей становится невидимой, а часть, наоборот, становится видимой. Цель игры – скрыть все надписи.

13. 13. Разместите на форме ряд кнопок (Button). Создайте обработчики события нажатия на данные кнопки, которые будут делать неактивными текущую кнопку. Создайте обработчик события изменения размера формы (Resize), который будет устанавливать все кнопки активный режим.

14. Разместите на форме ряд кнопок (Button). Создайте обработчи-ки события нажатия на данные кнопки, которые будут делать неактив-ными следующую кнопку. Создайте обработчик события нажатия кнопки мыши на форме (Click), который будет устанавливать все кнопки в активный режим.

15. Разместите на форме три кнопки (Button) и одно поле ввода (TextBox). Сделайте на кнопках следующие надписи: «*****», «+++++», «00000». Создайте обработчики события нажатия на данные кнопки, которые будут выводить текст, написанный на кнопках, в поле ввода. Создайте обработчик события создания формы (Load), который будет устанавливать цвет формы и менять текст в поле ввода на строку «Готов к работе».

16. Разместите на форме ряд полей ввода (TextBox). Создайте обработчики события нажатия кнопкой мыши на данные поля ввода, которые будут выводить в текущее поле ввода его номер. Создайте обработчик события изменения размера формы (Resize), который будет очищать все поля ввода.

17. Разместите на форме поле ввода (TextBox), метку (Label) кнопку (Button). Создайте обработчик события нажатия на кнопку, который будет копировать текст из поля ввода в метку. Создайте обработчик события нажатия кнопки мышки на форме (Click), который будет устанавливать цвет формы и менять текст метки на строку «Начало работы» и очищать поле ввода.

18. Разместите на форме поле ввода (TextBox) и две кнопки (Button) с надписями: «блокировать», «разблокировать». Создайте обработчики события нажатия на кнопки, которые будут делать активным или неактивным поле ввода. Создайте обработчик события нажатия кнопки мышки на форме (Click), который будет устанавливать цвет формы и делать невидимыми все элементы.

19. Реализуйте игру минер на поле 3×3 из кнопок (Button). Первоначально все кнопки не содержат надписей. При попытке нажатия на кнопку на ней либо показывается количество мин, либо надпись «мина!» и меняется цвет окна.

20. Разместите на форме четыре кнопки (Button). Напишите для каждой обработчик события, который будет менять размеры и местоположение на окне других кнопок.

Практическая работа № 2 «Составление программ линейной структуры»

Цель работы: научиться составлять каркас простейшей программы в среде Visual Studio. Написать и отладить программу линейного алгоритма.

Структура приложения

Перед началом программирования необходимо создать проект. Проект содержит все исходные материалы для приложения, такие как файлы исходного кода, ресурсов, значки, ссылки на внешние файлы, на которые опирается программа, и данные конфигурации, такие как параметры компилятора.

Кроме понятия проект часто используется более глобальное понятие – решение (solution). Решение содержит один или несколько проек-тов, один из которых может быть указан как стартовый проект. Выполнение решения начинается со стартового проекта.

Таким образом, при создании простейшей C# программы в Visual Studio создается папка решения, в которой для каждого проекта создается подпапка проекта, а уже в ней – другие подпапки с результатами компиляции приложения.

Проект – это основная единица, с которой работает программист. При создании проекта можно выбрать его тип, а Visual Studio создаст каркас проекта в соответствии с выбранным типом.

В предыдущей практической работе мы попробовали создавать оконные приложения, или иначе Приложения Windows Forms. Примером другого типа проекта является привести проект консольного приложения.

По своим «внешним» проявлениям консольные напоминают приложения DOS, запущенные в Windows. Тем не менее, это настоящие Win32-приложения, которые под DOS работать не будут. Для консольных приложений доступен Win32 API, а кроме того, они могут исполь-зовать консоль – окно, предоставляемое системой, которое работает в текстовом режиме и в которое можно вводить данные с клавиатуры.

Особенность консольных приложений в том, что они работают не в графическом, а в текстовом режиме.

Проект в Visual Studio состоит из файла проекта (файл с расширением .csproj), одного или нескольких файлов исходного текста (с рас-ширением .cs), файлов с описанием окон формы (с расширением .designer.cs), файлов ресурсов (с расширением .resx), а также ряда служебных файлах.

В файле проекта находится информация о модулях, составляющих данный проект, входящих в него ресурсах, а также параметров построе-ния программы. Файл проекта автоматически создается и изменяется средой Visual Studio и не предназначен для ручного редактирования.

Файл исходного текста – программный модуль, предназначен для размещения текстов программ. В этом файле программист размещает текст программы, написанный на языке C#. Модуль имеет следующую структуру:

```
//    Раздел подключенных пространств имен using System;
//    Пространство имен нашего проекта namespace
MyFirstApp
{
//          Класс окна
public partial class Form1 : Form
{
//                Методы окна public
Form1()
{
InitializeComponent();
}
}
}
```

В разделе подключения пространств имен (каждая строка которого располагается в начале файла и начинается ключевым словом using) описываются используемые пространства имен. Каждое пространство имен включает в себя классы, выполняющие определенную работу, например, классы для работы с сетью располагаются в пространстве System.Net, а для работы с файлами – в System.IO. Большая часть пространств, которые используются в обычных проектах,

уже подключена при создании нового проекта, но при необходимости можно дописать дополнительные пространства имен.

Для того чтобы не происходило конфликтов имен классов и переменных, классы нашего проекта также помещаются в отдельное про-странство имен. Определяется оно ключевым словом namespace, после которого следует имя пространства (обычно оно совпадает с именем проекта).

Внутри пространства имен помещаются наши классы – в новом проекте это класс окна, который содержит все методы для управления поведением окна. Обратите внимание, что в определении класса присутствует ключевое слово partial, это говорит о том, что в исходном тексте представлена только часть класса, с которой мы работаем непосредственно, а служебные методы для обслуживания окна скрыты в другом модуле (при желании их тоже можно посмотреть, но редактировать вручную не рекомендуется).

Наконец, внутри класса располагаются переменные, методы и другие элементы программы. Фактически, основная часть программы размещается внутри класса при создании обработчиков событий.

При компиляции программы Visual Studio создает исполняемые .exe-файлы в каталоге bin.

Работа с проектом

Проект в Visual Studio состоит из многих файлов, и создание сложной программы требует хранения каждого проекта в отдельной папке. При создании нового проекта Visual Studio по умолчанию сохраняет его в отдельной папке. Рекомендуется создать для себя свою папку со своей фамилией внутри папки своей группы, чтобы все проекты хранились в одном месте. После этого можно запускать Visual Studio и создавать новый проект (как это сделать, показано в предыдущей лабораторной работе).

Сразу после создания проекта рекомендуется сохранить его в подготовленной папке: Файл → Сохранить все. При внесении значительных изменений в проект следует еще раз сохранить проект той же ко-мандой, а перед запуском программы на выполнение среда обычно сама сохраняет проект на случай какого-либо сбоя. Для открытия существующего проекта используется команда Файл → Открыть проект, ли-бо можно найти в папке файл проекта с расширением .sln и сделать на нем двойной щелчок.

Описание данных

Типы данных имеют особенное значение в C#, поскольку это строго типизированный язык. Это означает, что все операции подвергаются строгому контролю со стороны компилятора на соответствие типов, причем недопустимые операции не компилируются. Такая строгая проверка типов позволяет предотвратить ошибки и повысить надежность программ. Для обеспечения контроля типов все переменные, выражения и значения должны принадлежать к определенному типу. Такого понятия, как бестиповая переменная, допустимая в ряде скриптовых языков, в C# вообще не существует. Более того, тип значения определяет те операции, которые разрешается выполнять над ним. Операция, разрешенная для одного типа данных, может оказаться недопустимой для другого.

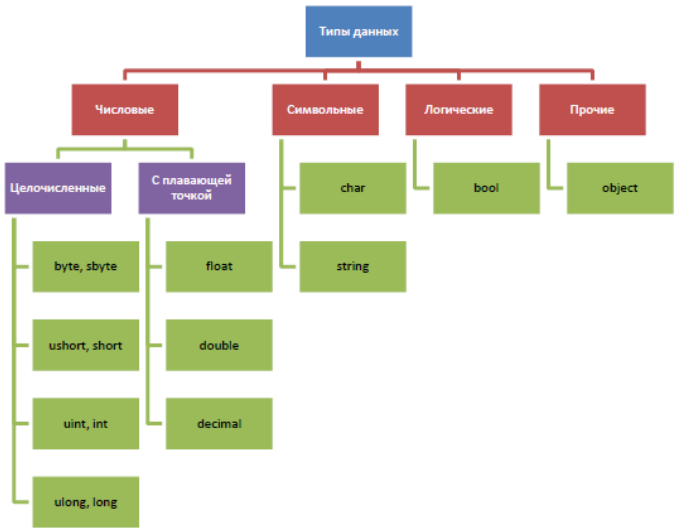


Рис. 2.1. Структура типов данных

Целочисленные типы

В C# определены девять целочисленных типов: char, byte, sbyte, short, ushort, int, uint, long и ulong. Тип char может хранить числа, но чаще используется для представления символов. Остальные восемь целочисленных типов предназначены для числовых расчетов.

Некоторые целочисленные типы могут хранить как положительные, так и отрицательные значения (sbyte, short, int и long), другие же – только положительные (char, byte, ushort, uint и ulong).

Типы с плавающей точкой

Такие типы позволяют представлять числа с дробной частью. В C# имеются три разновидности типов данных с плавающей точкой: float, double и decimal. Первые два типа представляют числовые значения с одинарной и двойной точностью, вычисления над ними выполняются аппаратно и поэтому быстро. Тип decimal служит для представления чисел с плавающей точкой высокой точности без округления, характерного для типов float и double. Вычисления с использованием этого типа выполняются программно и поэтому более медленны.

Числа, входящие в выражения, C# по умолчанию считает целочисленными. Поэтому следующее выражение будет иметь значение 0, ведь если 1 нацело разделить на 2, то получится как раз 0: double x = 1 / 2;

Чтобы этого не происходило, в подобных случаях нужно явно указывать тип чисел с помощью символов -модификаторов: f для float и d для double. Приведенный выше пример правильно будет выглядеть так: double x = 1d / 2d;

Иногда в программе возникает необходимость записать числа в экспоненциальной форме. Для этого после мантиссы числа записывается символ «e» и сразу после него – порядок. Например, число $2{,}5 \cdot 10^{-2}$ будет записано в программе следующим образом: 2.5e-2

Символьные типы

В C# символы представлены не 8-разрядным кодом, как во многих других языках программирования, а 16-разрядным кодом, который на-зывается юникодом (Unicode). В юникоде набор символов представлен настолько широко, что он охватывает символы практически из всех естественных языков на свете.

Основным типом при работе со строками является тип string, задающий строки переменной длины. Тип string представляет последовательность из нуля или более символов в кодировке Юникод. По сути, текст хранится в виде последовательной доступной только для чтения коллекции объектов char.

Логический тип данных

Тип bool представляет два логических значения: «истина» и «ложь». Эти логические значения обозначаются в C# зарезервированными словами true и false соответственно. Следовательно, переменная или выражение типа bool будет принимать одно из этих логических значений.

Рассмотрим самые популярные данные – переменные и константы. Переменная – это ячейка памяти, которой присвоено некоторое имя, и это имя используется для доступа к данным, расположенным в данной ячейке.

Для каждой переменной задается тип данных – диапазон всех возможных значений для данной переменной. Объявляются переменные непосредст-венно в тексте программы. Лучше всего сразу присвоить им начальное значение с помощью знака присвоения «=» (переменная = значение):

```
int a;              // Только объявление
int b = 7; // Объявление и инициализация
```

Для того чтобы присвоить значение символьной переменной, достаточно заключить это значение (т. е. символ) в одинарные кавычки:

```
char ch;                        // Только объявление
char symbol = 'Z'; // Объявление и инициализация
```

Частным случаем переменных являются константы. Константы – это переменные, значения которых не меняются в процессе выполнения программы. Константы описываются как обычная переменная, только с ключевым словом const впереди:

```
const int c = 5;
```

Ввод/вывод данных в программу

Рассмотрим один из способов ввода данных через элементы, размещенные на форме. Для ввода данных чаще всего используют элемент управления TextBox, через обращение к его свойству Text. Свойство Text хранит в себе строку введенных символов. Поэтому данные можно считать таким образом:

```
private void button1_Click(object sender,EventArgs e)
{
string s = textBox1.Text;
}
```

Однако со строкой символов трудно производить арифметические операции, поэтому лучше всего при вводе числовых данных пе-ревести строку в целое или вещественное число. Для этого у типов int и double существуют методы Parse для преобразования строк числа. С этими числами можно производить различные арифметические действия. Таким образом, предыдущий пример можно переделать следующим образом:

```
private void button1_Click(object sender, EventArgs e)
{
string s = textBox1.Text;
int a = int.Parse(s);
int b = a * a;
}
```

Арифметические действия и стандартные функции

При вычислении выражения, стоящего в правой части оператора присвоения, могут использоваться арифметические операции:

- умножение (×);
- сложение (+);
- вычитание (–);
- деление (/);
- остаток от деления (%).

Для задания приоритетов операций могут использоваться круглые скобки ( ). Также могут использоваться стандартные математические функции, представленные методами класса Math:

- Math.Sin(a) – синус;
- Math.Sinh(a) – гиперболический синус;
- Math.Cos(a) – косинус (аргумент задается в радианах);
- Math.Atan(a) – арктангенс (аргумент задается в радианах);
- Math.Log(a) – натуральный логарифм;
- Math.Exp(a) – экспонента;
- Math.Pow(x, y) – возводит переменную x в степень y;
- Math.Sqrt(a) – квадратный корень;
- Math.Abs(a) – модуль числа;
- Math.Truncate(a) – целая часть числа;
- Math.Round(a) – округление числа.

В тригонометрических функциях все аргументы задаются в радианах.

Пример написания программы

Задание: составить программу вычисления для заданных зна-чений x, y, z арифметического выражения:

$$u = \mathrm{tg}^2(x+y) - \mathrm{e}^{y-z}\sqrt{\cos x^2 + \sin z^2}$$

Панель диалога программы организовать в виде, представленном на рис. 2.2.

Для вывода результатов работы программы в программе используется текстовое окно, которое представлено обычным элементом управления. После установки свойства Multiline в True появляется возможность растягивать элемент управления не только по горизонтали, но и по вертикали. А после установки свойства ScrollBars в значение Both в окне появится вертикальная, а при необходимости и горизонтальная полосы прокрутки.

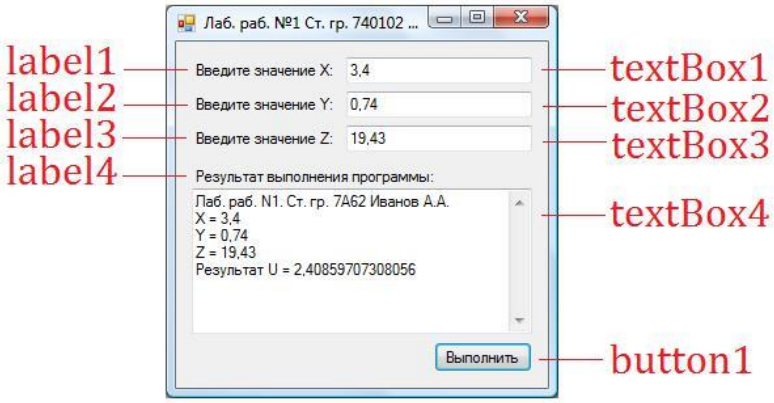


Рис. 2.2. Внешний вид программы

Информация, которая отображается построчно в окне, находится в массиве строк Lines, каждая строка которого имеет тип string. Однако нельзя напрямую обратиться к этому свойству для добавления новых строк, поскольку размер массивов в C# определяется в момент их инициализации. Для добавления нового элемента используется свойство Text, к текущему содержимому которого можно добавить новую строку:

textBox4.Text += Environment.NewLine + "Привет";

В этом примере к текущему содержимому окна добавляется символ перевода курсора на новую строку (который может отличаться в разных операционных системах и потому представлен свойством класса Environment) и сама новая строка. Если добавляется числовое значение, то его предварительно нужно привести в символьный вид методом

ToString().

Работа с программой происходит следующим образом. Нажмите (щелкните мышью) кнопку «Выполнить». В окне textBox4 появляется результат. Измените исходные значения x, y, z в окнах textBox1– textBox3 и снова нажмите кнопку «Выполнить» – появятся новые ре-зультаты.

Полный текст программы имеет следующий вид:

```
using System;
using System.Windows.Forms;
namespace MyFirstApp
{
public partial class Form1 : Form
{
public Form1()
{
InitializeComponent();
}
private void Form1_Load(object sender, EventArgs e)
{
//                    Начальное    значение    X
textBox1.Text = "3,4";
//                    Начальное    значение    Y
textBox2.Text = "0,74";
//                    Начальное     значение     Z
textBox3.Text = "19,43";
}
private void button1_Click(object sender, EventArgs e)
{
//                    Считывание значения X
double x = double.Parse(textBox1.Text);
//                    Вывод  значения  X  в  окно  textBox4.Text  +=
Environment.NewLine +
"X = " + x.ToString();
//                    Считывание значения Y
double y = double.Parse(textBox2.Text);
```

```
        //                  Вывод значения Y в окно textBox4.Text +=
Environment.NewLine +
        "Y = " + y.ToString();
        //                  Считывание значения Z
        double z = double.Parse(textBox3.Text);
        //                  Вывод значения Z в окно textBox4.Text +=
Environment.NewLine +
        "Z = " + z.ToString();
        //                  Вычисляем арифметическое выражение double a =
Math.Tan(x + y) *
        Math.Tan(x + y); double b = Math.Exp(y - z);
        double c = Math.Sqrt(Math.Cos(x*x) + Math.Sin(z*z));
        double u = a -b * c;
        //                  Выводим результат в окно textBox4.Text +=
Environment.NewLine +
        "Результат U = " + u.ToString(); } } }
```

Если просто скопировать этот текст и заменить им то, что было в редакторе кода Visual Studio, то программа не заработает. Правильнее будет создать обработчики событий Load у формы и Click у кнопки и уже в них вставить соответствующий код. Это замечание относится и ко всем последующим лабораторным работам.

Задание на практическую работу

Ниже приведено 20 вариантов задач. По указанию преподавателя выберите свое индивидуальное задание. Уточните условие задания, количество, наименование, типы исходных данных. В соответствии с этим установите необходимое количество окон TextBox, тексты заголовков на форме, размеры шрифтов, а также типы переменных и функции преобразования при вводе и выводе результатов. Для проверки правильности программы после задания приведен контрольный пример: тестовые значения переменных, используемых в выражении, и результат, который при этом получается.

Индивидуальные задания

1. $t=\dfrac{2\cos\left(x-\dfrac{\pi}{6}\right)}{0.5+\sin^2 y}\left(1+\dfrac{z^2}{3-z^2/5}\right)$.

   При $x=14.26$, $y=-1.22$, $z=3.5\times10^{-2}$ $t=0.564849$.

2. $u=\dfrac{\sqrt[3]{8+|x-y|^2+1}}{x^2+y^2+2}-e^{|x-y|}\left(\operatorname{tg}^2 z+1\right)^x$.

   При $x=-4.5$, $y=0.75\times10^{-4}$, $z=0.845\times10^2$ $u=-55.6848$.

3. $v=\dfrac{1+\sin^2(x+y)}{\left|x-\dfrac{2y}{1+x^2y^2}\right|}x^{|y|}+\cos^2\left(\operatorname{arctg}\dfrac{1}{z}\right)$.

   При $x=3.74\times10^{-2}$, $y=-0.825$, $z=0.16\times10^2$, $v=1.0553$.

4. $w=\left|\cos x-\cos y\right|^{\left(1+2\sin^2 y\right)}\left(1+z+\dfrac{z^2}{2}+\dfrac{z^3}{3}+\dfrac{z^4}{4}\right)$.

   При $x=0.4\times10^4$, $y=-0.875$, $z=-0.475\times10^{-3}$ $w=1.9873$.

5. $\alpha=\ln\left(y^{-\sqrt{|x|}}\right)\left(x-\dfrac{y}{2}\right)+\sin^2\operatorname{arctg}(z)$.

   При $x=-15.246$, $y=4.642\times10^{-2}$, $z=20.001\times10^2$ $\alpha=-182.036$.

6. $\beta=\sqrt{10\left(\sqrt[3]{x}+x^{y+2}\right)}\left(\arcsin^2 z-|x-y|\right)$.

   При $x=16.55\times10^{-3}$, $y=-2.75$, $z=0.15$ $\beta=-38.902$.

7. $\gamma=5\operatorname{arctg}(x)-\dfrac{1}{4}\arccos(x)\dfrac{x+3|x-y|+x^2}{|x-y|z+x^2}$.

   При $x=0.1722$, $y=6.33$, $z=3.25\times10^{-4}$ $\gamma=-172.025$.

8. $\varphi=\dfrac{e^{|x-y|}|x-y|^{x+y}}{\operatorname{arctg}(x)+\operatorname{arctg}(z)}+\sqrt[3]{x^6+\ln^2 y}$.

   При $x=-2.235\times10^{-2}$, $y=2.23$, $z=15.221$ $\varphi=39.374$.

9. $\psi = \left| x^{\frac{y}{x}} - \sqrt[3]{\frac{y}{x}} \right| + (y-x)\frac{\cos y - \frac{z}{(y-x)}}{1+(y-x)^2}.$

При $x = 1.825\times10^{2}$, $y = 18.225$, $z = -3.298\times10^{-2}$ $\psi = 1.2131$.

10. $a = 2^{-x}\sqrt{x + \sqrt[4]{|y|}\sqrt[3]{e^{x-1/\sin z}}}.$

При $x = 3.981\times10^{-2}$, $y = -1.625\times10^{3}$, $z = 0.512$ $a = 1.26185$.

11. $b = y^{\sqrt[3]{|x|}} + \cos^3(y)\frac{|x-y|\left(1+\frac{\sin^2 z}{\sqrt{x+y}}\right)}{e^{|x-y|}+\frac{x}{2}}.$

При $x = 6.251$, $y = 0.827$, $z = 25.001$ $b = 0.7121$.

12. $c = 2^{(y^x)} + (3^x)^y - \frac{y\left(\operatorname{arctg} z - \frac{\pi}{6}\right)}{|x| + \frac{1}{y^2+1}}.$

При $x = 3.251$, $y = 0.325$, $z = 0.466\times10^{-4}$ $c = 4.025$.

13. $f = \frac{\sqrt[4]{y + \sqrt[3]{x-1}}}{|x-y|\left(\sin^2 z + \operatorname{tg} z\right)}.$

При $x = 17.421$, $y = 10.365\times10^{-3}$, $z = 0.828\times10^{5}$ $f = 0.33056$.

14. $g = \frac{y^{x+1}}{\sqrt[3]{|y-2|+3}} + \frac{x+\frac{y}{2}}{2|x+y|}(x+1)^{-1/\sin z}.$

При $x = 12.3\times10^{-1}$, $y = 15.4$, $z = 0.252\times10^{3}$ $g = 82.8257$.

15. $h = \frac{x^{y+1} + e^{y-1}}{1+x|y-\operatorname{tg} z|}\left(1+|y-x|\right) + \frac{|y-x|^2}{2} - \frac{|y-x|^3}{3}.$

При $x = 2.444$, $y = 0.869\times10^{-2}$, $z = -0.13\times10^{3}$ $h = -0.49871$.

16. $y = \sqrt{cx} - 2.7\frac{|c|+|x|}{c^2x^2}\cdot e^{cx} + \cos\frac{(a+b)^2}{cx-b}$

$a = 3.7$; $b = 0.07$; $c = 1.5$; $x = 5.75$

17. $y = 4.5\frac{(a+b)^2}{(a-b)^2} - \sqrt{(a+b)(a-b)} + 10^{-1}\frac{\ln(a-b)}{\ln(a+b)}\cdot e^{x^2}$

$a = 7.5$; $b = 1.2$; $x = 0.5$

18. $y = 2.4\left|\frac{x^2+b}{a}\right| + (a-b)\sin^2(a-b) + 10^{-2}(x-b)$

$a = 5.1$; $b = 0.7$; $x = -0.05$

19. $y = \frac{ax-\sqrt{b}}{5.7(x^2+b^2)} - \frac{|x+b|-a^2}{x^2}\operatorname{tg}^2 b$

$a = 0.1$; $b = 2.4$; $x = -0.3$

20. $y = \sqrt{\frac{c-dx^2}{x}} + \frac{\ln(x^2+c)}{0.7x+ad} - \frac{10^{-2}}{c-dx^3}$

$a = 4.5$; $c = 7.4$; $d = -2.1$; $x = 0.15$

Практическая работа № 3 «Составление программ разветвляющейся структуры»

Цель работы: научиться пользоваться элементами управления для организации переключений (RadioButton). Написать и отладить программу разветвляющегося алгоритма.

Логические переменные и операции над ними

Переменные логического типа описываются посредством служебного слова bool. Они могут принимать только два значения – False (ложь) и True (истина). Результат False (ложь) и True (истина) возника-ет при использовании операций сравнения > (больше), < (меньше), != (не равно), >= (больше или равно), <= (меньше или равно), == (равно). Описываются логические переменные следующим образом:

bool b;

В языке C# имеются логические операции, применяемые к переменным логического типа. Это операции логического отрицания (!), логическое И (&&) и логическое ИЛИ (||). Операция логического отрицания является унарной операцией. Результат операции ! есть False, если операнд истинен, и True, если операнд имеет значение «ложь». Так, !True → False (неправда есть ложь), !False → True (не ложь есть правда).

Результат операции логическое И (&&) есть истина, только если оба операнда истинны, и ложь во всех других случаях. Результат опера-ции логическое ИЛИ (||) есть истина, если какой-либо из ее операндов истинен, и ложен только тогда, когда оба операнда ложны.

Условные операторы

Операторы ветвления позволяют изменить порядок выполнения операторов в программе. К операторам ветвления относятся условный оператор if и оператор выбора switch.

Условный оператор if используется для разветвления процесса об-работки данных на два направления. Он может иметь одну из форм: со-кращенную или полную.

Форма сокращенного оператора if:

if (B) S;

где B – логическое или арифметическое выражение, истинность которого проверяется; S – оператор.

При выполнении сокращенной формы оператора if сначала вычисляется выражение B, затем проводится анализ его результата: если B истинно, то выполняется оператор S; если B ложно, то оператор S пропускается. Таким образом, с помощью сокращенной формы оператора if можно либо выполнить оператор S, либо пропустить его.

Форма полного оператора if:

if (B) S1; else S2;

где B – логическое или арифметическое выражение, истинность которого проверяется; S1, S2 – операторы.

При выполнении полной формы оператора if сначала вычисляется выражение B, затем анализируется его результат: если B истинно, то выполняется оператор S1, а оператор S2 пропускается; если B ложно, то выполняется оператор S2, а S1 – пропускается. Таким образом, с помощью полной формы оператора if можно выбрать одно из двух альтернативных действий процесса обработки данных.

Для примера вычислим значение функции:

$$y(x) = \begin{cases} \sin(x), \text{ если } x \le a \\ \cos(x), \text{ если } a < x < b \\ \operatorname{tg}(x), \text{ если } x \ge b \end{cases}$$

Указанное выражение может быть запрограммировано в виде

```
if (x <= a)
y = Math.Sin(x);
if ((x > a) && (x < b))
y = Math.Cos(x);
if (x >= b)
y = Math.Sin(x) / Math.Cos(x);
```

или с помощью оператора else:

```
if (x <= a)
y = Math.Sin(x);
else
```

```
if (x < b)
y = Math.Cos(x);
else
y = Math.Sin(x) / Math.Cos(x);
```

Важное примечание! В С-подобных языках программирования, которым относится и C#, операция сравнения представляется двумя знаками равенства, например:

```
if (a == b)
```

Оператор выбора switch предназначен для разветвления процесса вычислений по нескольким направлениям. Формат оператора:

```
switch (<выражение>)
{
case <константное_выражение_1>:
[<оператор 1>];
<оператор перехода>;
case <константное_выражение_2>:
[<оператор 2>];
<оператор перехода>;
...
case <константное_выражение_n>:
[<оператор n>];
<оператор перехода>;
[default:
<оператор>;]
}
```

Замечание. Выражение, записанное в квадратных скобках, является необязательным элементом в операторе switch. Если оно отсутствует, то может отсутствовать и оператор перехода.

Выражение, стоящее за ключевым словом switch, должно иметь арифметический, символьный или строковый тип. Все константные выражения должны иметь разные значения, но их тип должен совпадать с типом выражения, стоящим после switch или приводиться к нему. Ключевое слово case и расположенное после него константное выражение называют также меткой case.

Выполнение оператора начинается с вычисления выражения, расположенного за ключевым словом switch. Полученный результат сравнивается с меткой case. Если результат выражения соответствует метке case, то выполняется оператор, стоящий после этой метки, за которым обязательно должен следовать оператор перехода: break, goto, return т. д. При использовании оператора break происходит выход из switch управление передается оператору, следующему за switch. Если же используется оператор goto, то управление передается оператору, помеченному меткой, стоящей после goto. Наконец, оператор return завершает выполнение текущего метода.

Если ни одно выражение case не совпадает со значением оператора switch, управление передается операторам, следующим за необязательной подписью default. Если подписи default нет, то управление передается за пределы оператора switch.

Пример использования оператора switch:

```
int dayOfWeek = 5;
switch (dayOfWeek)
{
case 1:
MessageBox.Show("Понедельник");
break;
case 2:
MessageBox.Show("Вторник");
break;
case 3:
MessageBox.Show("Среда");
break;
case 4:
```

```
MessageBox.Show("Четверг");
break;
case 5:
MessageBox.Show("Пятница");
break;
case 6:
MessageBox.Show("Суббота");
break;
case 7:
MessageBox.Show("Воскресенье");
break;
default:
MessageBox.Show("Неверное значение!"); break;
}
```

Поскольку на момент выполнения оператора switch в этом примере переменная dayOfWeek равнялась 5, то будут выполнены операторы из блока case 5.

В ряде случаев оператор switch можно заменить несколькими опе-раторами if, однако не всегда такая замена будет легче для чтения. Например, приведенный выше пример можно переписать так:

```
int dayOfWeek = 5;
if (dayOfWeek == 1)
MessageBox.Show("Понедельник");
else
if (dayOfWeek == 2)
MessageBox.Show("Вторник");
else
if (dayOfWeek == 3)
MessageBox.Show("Среда");
else
if (dayOfWeek == 4)
MessageBox.Show("Четверг");
else
if (dayOfWeek == 5)
MessageBox.Show("Пятница");
else
if (dayOfWeek == 6)
MessageBox.Show("Суббота");
else
if (dayOfWeek == 7)
MessageBox.Show("Воскресенье");
else
MessageBox.Show("Неверное значение!");
```

Кнопки-переключатели

При создании программ в Visual Studio для организации разветвлений часто используются элементы управления в виде кнопок-переключателей (RadioButton). Состояние такой кнопки (включено–выключено) визуально отражается на форме, а в программе можно узнать его с помощью свойства Checked: если кнопка включена, это свойство будет содержать True, в противном случае False. Если пользователь выбирает один из вариантов переключателя в группе, все остальные автоматически отключаются.

Группируются радиокнопки с помощью какого-либо контейнера – часто это бывает элемент GroupBox. Радиокнопки, размещенные в разных контейнерах, образуют независимые группы.

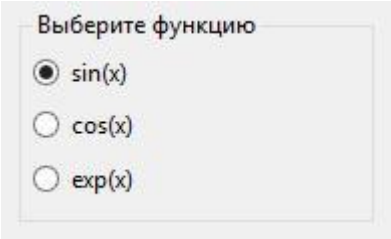


Рис. 3.1. Группа радиокнопок

```
if (radioButton1.Checked)
MessageBox.Show("Выбрана функция: синус");
else if (radioButton2.Checked)
MessageBox.Show("Выбрана функция: косинус");
else if (radioButton1.Checked)
MessageBox.Show("Выбрана функция: экспонента");
```

Пример написания программы

Задание: ввести

$$U = \begin{cases} y \cdot \sin(x) + z, & \text{при } z - x = 0 \\ y \cdot e^{\sin(x)} - z, & \text{при } z - x < 0 \\ y \cdot \sin(\sin(x)) + z, & \text{при } z - x > 0 \end{cases}$$

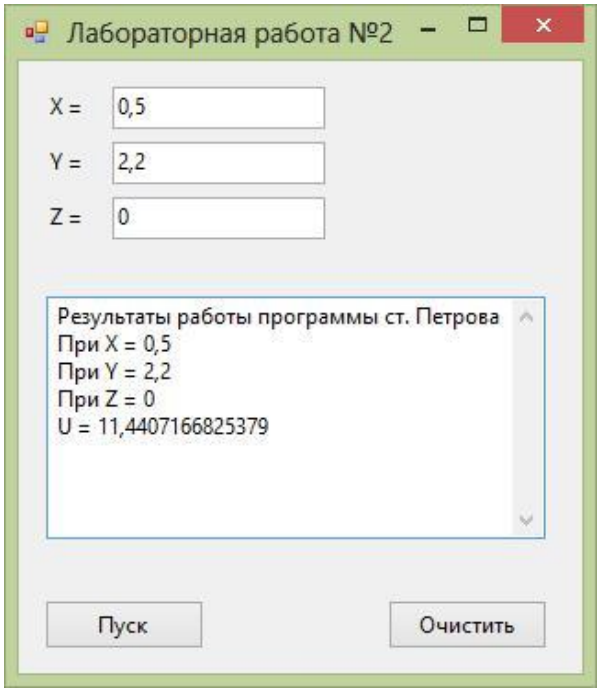


Рис. 3.2. Окно лабораторной работы

Создание формы

Создайте форму, в соответствии с рис. 3.2.

Разместите на форме элементы Label, TextBox и Button. Поле для вывода результатов также является элементом TextBox с установленным в True свойством Multiline и свойством ScrollBars установленным в Both.

Создание обработчиков событий

Обработчики событий создаются аналогично тому, как и в преды-дущих лабораторных работах. Текст обработчика события нажатия на кнопку «Пуск» приведен ниже.

```
private void button1_Click(object sender, EventArgs e)
{
// Получение исходных данных из TextBox double x = Convert.ToDouble(textBox2.Text); double y = Convert.ToDouble(textBox1.Text);
double z = Convert.ToDouble(textBox3.Text);
// Ввод исходных данных в окно результатов textBox4.Text = "Результаты работы программы " +
```

```
"ст. Петрова И.И. " + Environment.NewLine;
textBox4.Text += "При X = " + textBox2.Text + Environment.NewLine;
textBox4.Text += "При Y = " + textBox1.Text + Environment.NewLine;
textBox4.Text += "При Z = " + textBox3.Text + Environment.NewLine;
//              Вычисление выражения
double u;
if ((z - x) == 0)
u = y * Math.Sin(x) * Math.Sin(x) + z; else
if ((z - x) < 0)
u = y * Math.Exp(Math.Sin(x)) - z;
else
u = y * Math.Sin(Math.Sin(x)) + z;
//              Вывод результата
textBox4.Text += "U = " + u.ToString() + Environment.NewLine;
}
```

Запустите программу и убедитесь в том, что все ветви алгоритма выполняются правильно.

Задание на практическую работу

По указанию преподавателя выберите индивидуальное задание из нижеприведенного списка. В качестве f(x) использовать по выбору: sh(x), x^2, e^x. Отредактируйте вид формы и текст программы, в соответствии с полученным заданием.

Усложненный вариант задания для продвинутых студентов: с помощью радиокнопок (RadioButton) дать пользователю возможность во время работы программы выбрать одну из трех приведенных выше функций.

Индивидуальные задания

1. $a=\begin{cases}(f(x)+y)^2-\sqrt{f(x)y}, & xy\rangle 0\\ (f(x)+y)^2+\sqrt{|f(x)\,y|}, & xy\langle 0\\ (f(x)+y)^2+1, & xy=0.\end{cases}$

2. $b=\begin{cases}\ln(f(x))+(f(x)^2+y)^3, & x/y\rangle 0\\ \ln|f(x)/y|+(f(x)+y)^3, & x/y\langle 0\\ (f(x)^2+y)^3, & x=0\\ 0, & y=0.\end{cases}$

3. $c=\begin{cases}f(x)^2+y^2+\sin(y), & x-y=0\\ (f(x)-y)^2+\cos(y), & x-y\rangle 0\\ (y-f(x))^2+\operatorname{tg}(y), & x-y\langle 0.\end{cases}$

4. $d=\begin{cases}(f(x)-y)^3+\operatorname{arctg}(f(x)), & x\rangle y\\ (y-f(x))^3+\operatorname{arctg}(f(x)), & y\rangle x\\ (y+f(x))^3+0.5, & y=x.\end{cases}$

5. $e=\begin{cases}i\sqrt{f(x)}, & i-\text{нечетное},\ x\rangle 0\\ i/2\sqrt{|f(x)|}, & i-\text{четное},\ x\langle 0\\ \sqrt{|if(x)|}, & \text{иначе}.\end{cases}$

6. $g=\begin{cases}e^{f(x)-|b|}, & 0.5\langle xb\langle 10\\ \sqrt{|f(x)+b|}, & 0.1\langle xb\langle 0.5\\ 2f(x)^2, & \text{иначе}.\end{cases}$

7. $s=\begin{cases}e^{f(x)}, & 1\langle xb\langle 10\\ \sqrt{|f(x)+4*b|}, & 12\langle xb\langle 40\\ bf(x)^2, & \text{иначе}.\end{cases}$

8. $j=\begin{cases}\sin(5f(x)+3m|f(x)|), & -1\langle m\langle x\\ \cos(3f(x)+5m|f(x)|), & x\rangle m\\ (f(x)+m)^2, & x=m.\end{cases}$

9. $l=\begin{cases}2f(x)^3+3p^2, & x\rangle|p|\\ |f(x)-p|, & 3\langle x\langle|p|\\ (f(x)-p)^2, & x=|p|.\end{cases}$

10. $k=\begin{cases}\ln(|f(x)|+|q|), & |xq|\rangle 10\\ e^{f(x)+q}, & |xq|\langle 10\\ f(x)+q, & |xq|=10\end{cases}$

11. $m=\dfrac{\max(f(x),y,z)}{\min(f(x),y)}+5.$

12. $n=\dfrac{\min(f(x)+y,y-z)}{\max(f(x),y)}.$

13. $p=\dfrac{|\min(f(x),y)-\max(y,z)|}{2}.$

14. $q=\dfrac{\max(f(x)+y+z,xyz)}{\min(f(x)+y+z,xyz)}.$

15. $c=\begin{cases}f(\sin(x))^2+\sin(f(y)), & x-y=0\\ (f(\cos(x)))+\cos(f(y)), & x-y\rangle 0\\ (y-f(\operatorname{tg}(x)))^2+\operatorname{tg}(y), & x-y\langle 0.\end{cases}$

16. $a=\begin{cases}\dfrac{(ax^2+2)}{(x^2+1)}f(x), & 1<|x|<3,\\ a^2+f(x), & |x|\ge 3\\ ax\dfrac{f(x)}{(x+2)}, & |x|\le 1.\end{cases}$

17. $c=\begin{cases}f(x)^3-y^3\cdot\cos(x), & x+y=0\\ (f(x)\cdot y)^2-\cos(y), & x+y\rangle 0\\ (y\cdot f(x))^2+\pi, & x+y\langle 0.\end{cases}$

18. $k=\begin{cases}\ln(|f(x^2)|+|k|), & |x\cdot k|\rangle 10\\ \pi^{f(x)+q}, & |x\cdot k|\langle 10\\ f(x)-k, & |x\cdot k|=10\end{cases}$

19. $c=\begin{cases}\sin(f(x))+\cos(f(y)), & x-y=0\\ \operatorname{tg}(f(x+y), & x-y\rangle 0\\ \sin^2(f(x))+\cos^2(f(y)), & x-y\langle 0.\end{cases}$

20. $r=\max(\min(f(x),y),z).$

Практическая работа № 4 «Составление программ циклической структуры»

Цель работы: изучить простейшие средства отладки программ в среде Visual Studio. Составить и отладить программу циклического алгоритма.

Операторы организации циклов

Под циклом понимается многократное выполнение одних и тех же операторов при различных значениях промежуточных данных. Число повторений может быть задано в явной или неявной форме.

К операторам цикла относятся: цикл с предусловием while, цикл постусловием do while, цикл с параметром for и цикл перебора foreach. Рассмотрим некоторые из них.

Цикл с предусловием

Оператор цикла while организует выполнение одного оператора (простого или составного) неизвестное заранее число раз. Формат цикла while: while (B) S; где B – выражение, истинность которого проверяется (условие завершения цикла); S – тело цикла – оператор (простой или составной). Перед каждым выполнением тела цикла анализируется значение выражения B: если оно истинно, то выполняется тело цикла, и управление передается на повторную проверку условия B; если значение B лож-но – цикл завершается и управление передается на оператор, следующий за оператором S. Если результат выражения B окажется ложным при первой проверке, то тело цикла не выполнится ни разу. Отметим, что если условие B во время работы цикла не будет изменяться , то возможна ситуация зацикливания, то есть невозможность выхода из цикла. Поэтому внутри тела должны находиться операторы, приводящие к изменению значения выражения B так, чтобы цикл мог корректно завершиться.

В качестве иллюстрации выполнения цикла while рассмотрим программу вывода целых чисел от 1 до n по нажатию кнопки на форме:

```
private void button1_Click(object sender, EventArgs e){
int n = 10; // Количество повторений цикла int i = 1; // Начальное значение
while (i <= n) // Пока i меньше или равно n
{
MessageBox.Show(i.ToString()); // Показываем i i++; // Увеличиваем i на 1 }}
```

Цикл с постусловием

Оператор цикла do while также организует выполнение одного оператора (простого или составного) неизвестное заранее число раз. Однако в отличие от цикла while условие завершения цикла проверяет-ся после выполнения тела цикла. Формат цикла do while: do S while (B); где B – выражение, истинность которого проверяется (условие заверше-ния цикла); S – тело цикла – оператор (простой или блок). Сначала выполняется оператор S, а затем анализируется значение вы-ражения B: если оно истинно, то управление передается оператору S, если ложно – цикл завершается, и управление передается на оператор, следую-щий за условием B. Так как условие B проверяется после выполнения тела цикла, то в любом случае тело цикла выполнится хотя бы один раз.

В операторе do while, так же как и в операторе while, возможна ситуация зацикливания в случае, если условие B всегда будет оставаться истинным.

Цикл с параметром

Цикл с параметром имеет следующую структуру:

for (<инициализация>; <выражение>; <модификация>) <оператор>;

Инициализация используется для объявления и/или присвоения начальных значений величинам, используемым в цикле в качестве параметров (счетчиков). В этой части можно записать несколько операторов, разделенных запятой. Областью действия переменных, объявленных части инициализации цикла, является цикл и вложенные блоки. Инициализация выполняется один раз в начале исполнения цикла.

Выражение определяет условие выполнения цикла: если его результат истинен, цикл выполняется. Истинность выражения проверяется перед каждым выполнением тела цикла, таким образом, цикл с параметром реализован как цикл с предусловием. В блоке выражение через запятую можно записать несколько логических выражений, тогда запятая равносильна операции логическое И (&&).

Модификация выполняется после каждой итерации цикла и служит обычно для изменения параметров цикла. В части модификация можно записать несколько операторов через запятую.

Оператор (простой или составной) представляет собой тело цикла. Любая из частей оператора for (инициализация, выражение, модификация, оператор) может отсутствовать, но точку с запятой, определяющую позицию пропускаемой части, надо оставить.

Пример формирования строки, состоящей из чисел от 0 до 9, разделенных пробелами:

string s = ""; // Инициализация строки

for (int i = 0; i <= 9; i++) // Перечисление всех чисел s += i.ToString() + " "; // Добавляем число и пробел

MessageBox.Show(s.ToString()); // Показываем результат

Данный пример работает следующим образом. Сначала вычисляется начальное значение переменной i. Затем, пока значение i меньше или равно 9, выполняется тело цикла, и затем повторно вычисляется значение i. Когда значение i становится больше 9, условие – ложно и управление передается за пределы цикла.

Средства отладки программ

Практически в каждой вновь написанной программе после запуска обнаруживаются ошибки. Ошибки первого уровня (ошибки компиляции) связаны с неправильной записью операторов (орфографические, синтаксические). При обнаружении ошибок компилятор формирует список, который отобра-жается по завершению компиляции (рис. 4.1). При этом возможен только запуск программы, которая была успешно скомпилирована для пре-дыдущей версии программы.

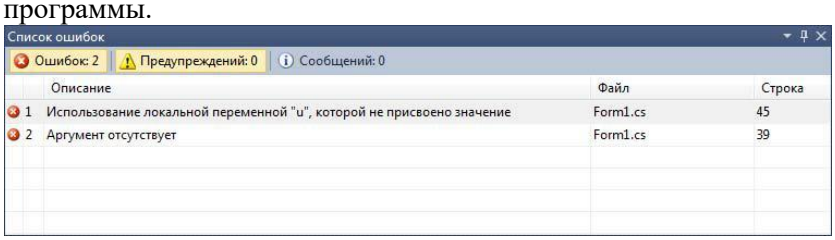


Рис. 4.1. Окно со списком ошибок компиляции

При выявлении ошибок компиляции в нижней части экрана появляется текстовое окно (см. рис. 4.1), содержащее сведения обо всех ошибках, найденных в проекте. Каждая строка этого окна содержит имя файла, в котором найдена ошибка, номер строки с ошибкой и характер ошибки. Для быстрого перехода к интересующей ошибке необходимо дважды щелкнуть на строке с ее описанием. Следует обратить внимание на то, что одна ошибка может повлечь за собой другие, которые исчезнут при ее исправлении. Поэтому необходимо исправлять их последовательно, сверху вниз и, после исправления каждой – компилировать программу снова.

Ошибки второго уровня (ошибки выполнения) связаны с ошибками выбранного алгоритма решения или с неправильной программной реализацией алгоритма. Эти ошибки проявляются в том, что результат расчета оказывается неверным либо происходит переполнение, деление на ноль и др. Поэтому перед использованием отлаженной программы ее надо протестировать, т. е. сделать просчеты при таких комбинациях исходных данных, для которых заранее известен результат. Если тестовые расчеты указывают на ошибку, то для ее поиска следует использовать встроенные средства отладки среды программирования.

1. простейшем случае для локализации места ошибки рекомендуется поступать следующим образом. В окне редактирования текста установить точку останова перед подозрительным участком, которая позволит остановить выполнение программы и далее более детально следить за ходом работы операторов и изменением значений переменных. Для этого достаточно в окне редактирования кода щелкнуть слева от нужной стро-ки. В результате чего данная строка будет выделена красным (рис. 4.2).

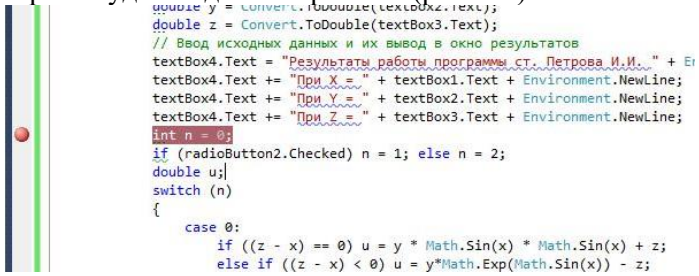


Рис. 4.2. Фрагмент кода с точкой останова

При выполнении программы и достижения установленной точки программа будет остановлена, и далее можно выполнять код по шагам с помощью команд Отладка → Шаг с обходом (без захода в методы) или Отладка → Шаг с заходом (с заходом в методы) (рис. 4.3).

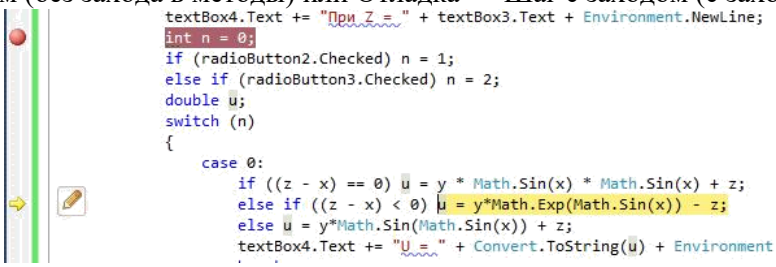


Рис. 4.3. Отладка программы

Желтым цветом выделяется оператор, который будет выполнен. Значение переменных во время выполнения можно увидеть, наведя на них курсор. Для прекращения отладки и остановки программы нужно выполнить команду меню Отладка → Остановить отладку.

Для поиска алгоритмических ошибок можно контролировать значения промежуточных переменных на каждом шаге выполнения подозрительного кода и сравнивать их с результатами, полученными вручную.

Порядок выполнения задания

Задание: Вычислить и вывести на экран таблицу значений функ-ции y = a·ln(x) при x, изменяющемся от x_0 до x_k с шагом dx, a – константа.

Панель диалога представлена на рис. 4.4. Текст обработчика нажа-тия кнопки **Вычислить** приведен ниже.

```
private void button1_Click(object sender, EventArgs e){
//          Считывание начальных данных
double x0 = Convert.ToDouble(textBox1.Text); double xk = Convert.ToDouble(textBox2.Text); double dx = Convert.ToDouble(textBox3.Text); double a = Convert.ToDouble(textBox4.Text); textBox5.Text = "Работу выполнил ст. Иванов М.А." +Environment.NewLine;
1.          Цикл для табулирования функции double x = x0;
while (x <= (xk + dx / 2)){
double y = a * Math.Log(x);
textBox5.Text += "x=" + Convert.ToString(x) +
"; y=" + Convert.ToString(y) +
Environment.NewLine;
x = x + dx;}}
```

После отладки программы следует проверить правильность работы программы с помощью контрольного примера (см. рис. 4.4). Установите точку останова на оператор перед циклом и запустите программу . После попадания на точку остановки, выполните пошагово программу и про-следите, как меняются все переменные в процессе выполнения.

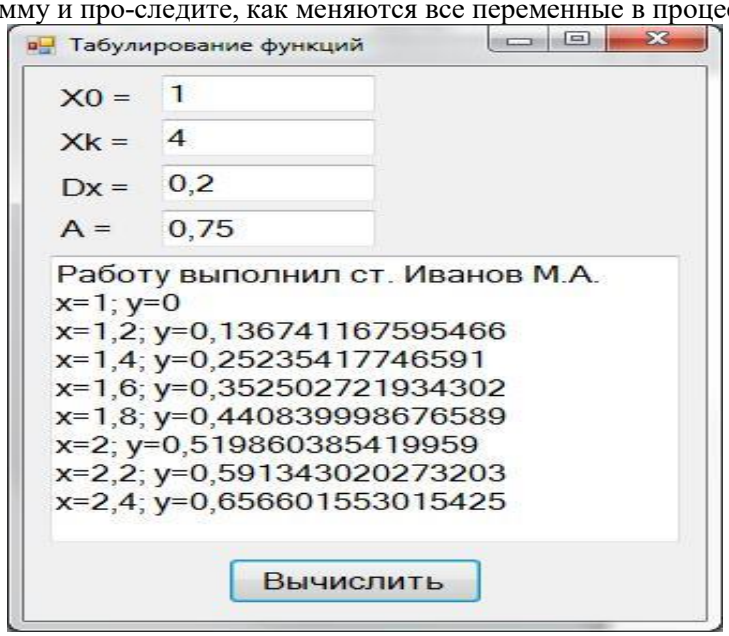


Рис. 4.4. Окно программы для табулирования функции

Задание на практическую работу

Составьте программу табулирования функции y(x), выведите на экран значения x и y(x) . Нужный вариант задания выберите из нижеприведенного списка по указанию преподавателя. Откорректируйте элементы управления в форме в соответствии со своим вариантом задания.

Индивидуальные задания

1) $y = 10^{-2}bc / x + \cos\sqrt{a^3 x}$,
$x_0 = -1.5; x_k = 3.5; dx = 0.5;$
$a = -1.25; b = -1.5; c = 0.75;$

2) $y = 1.2(a\text{-}b)^3 e^{x^2} + x$,
$x_0 = -0.75; x_k = -1.5; dx = -0.05;$
$a = 1.5; b = 1.2;$

3) $y = 10^{-1} ax^3 tg(a - bx)$,
$x_0 = -0.5; x_k = 2.5; dx = 0.05;$
$a = 10.2; b = 1.25;$

4) $y = ax^3 + \cos^2(x^3 - b)$,
$x_0 = 5.3; x_k = 10.3; dx = 0.25;$
$a = 1.35; b = -6.25;$

5) $y = x^4 + \cos(2 + x^3 - d)$,
$x_0 = 4.6; x_k = 5.8; dx = 0.2;$
$d = 1.3;$

6) $y = x^2 + tg(5x + b / x)$,
$x_0 = -1.5; x_k = -2.5; dx = -0.5;$
$b = -0.8;$

7) $y = 9(x + 15\sqrt{x^3 + b^3})$,
$x_0 = -2.4; x_k = 1; dx = 0.2;$
$b = 2.5;$

8) $y = 9x^4 + \sin(57.2 + x)$,
$x_0 = -0.75; x_k = -2.05; dx = -0.2;$

9) $y = 0.0025bx^3 + \sqrt{x + e^{0.82}}$,
$x_0 = -1; x_k = 4; dx = 0.5;$
$b = 2.3;$

10) $y = x \cdot \sin(\sqrt{x + b - 0.0084})$,
$x_0 = -2.05; x_k = -3.05; dx = -0.2;$
$b = 3.4;$

11) $y = x + \sqrt{\left|x^3 + a - be^x\right|}$,
$x_0 = -4; x_k = -6.2; dx = -0.2;$
$a = 0.1;$

12) $y = 9(x^3 + b^3)tgx$,
$x_0 = 1; x_k = 2.2; dx = 0.2;$
$b = 3.2;$

13) $y = \left|x - b\right|^{1/2} / \left|b^3 - x^3\right|^{3/2} + \ln\left|x - b\right|$,
$x_0 = -0.73; x_k = -1.73; dx = -0.1;$
$b = -2;$

14) $y = (x^{5/2} - b)\ln(x^2 + 12.7)$,
$x_0 = 0.25; x_k = 5.2; dx = 0.3;$
$b = 0.8;$

15) $y = 10^{-3}\left|x\right|^{5/2} + \ln\left|x + b\right|$,
$x_0 = 1.75; x_k = -2.5; dx = -0.25;$
$b = 35.4;$

16) $y = 15.28\left|x\right|^{-3/2} + \cos(\ln\left|x\right| + b)$,
$x_0 = 1.23; x_k = -2.4; dx = -0.3;$
$b = 12.6;$

17) $y = 0.00084(\ln\left|x\right|^{5/4} + b)/(x^2 + 3.82)$,
$x_0 = -2.35; x_k = -2; dx = 0.05;$
$b = 74.2;$

18) $y = 0.8 \cdot 10^{-5}(x^3 + b^3)^{7/6}$,
$x_0 = -0.05; x_k = 0.15; dx = 0.01;$
$b = 6.74;$

19) $y = (\ln(\sin(x^3 + 0.0025)))^{3/2} + 0.8 \cdot 10^{-3}$,
$x_0 = 0.12; x_k = 0.64; dx = 0.2;$

20) $y = a + x^{2/3}\cos(x + e^x)$,
$x_0 = 5.62; x_k = 15.62; dx = 0.5;$
$a = 0.41$

Практическая работа № 5 «Обработка одномерных массивов. Обработка двумерных массивов»

Одномерные массивы

Цель работы: Изучить способы получения случайных чисел. Написать программу для работы с одномерными массивами.

Работа с массивами

Массив – набор элементов одного и того же типа, объединенных общим именем. Массивы в C# можно использовать по аналогии с тем, как они используются в других языках программирования. Однако C#-массивы имеют существенные отличия: они относятся к ссылочным типам данных, более того – реализованы как объекты. Фактически имя массива является ссылкой на область кучи (динамической памяти), в которой последовательно размещается набор элементов определенного типа. Выделение памяти под элементы происходит на этапе инициализации массива. А за освобождением памяти следит система сборки мусора – неиспользуемые массивы автоматически утилизируются данной системой.

Рассмотрим в данной лабораторной работе одномерные массивы. Одномерный массив – это фиксированное количество элементов одного и того же типа, объединенных общим именем, где каждый элемент имеет свой номер. Нумерация элементов массива в C# начинается с нуля, то есть если массив состоит из 10 элементов, то его элементы будут иметь следующие номера: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Одномерный массив в C# реализуется как объект, поэтому его создание представляет собой двухступенчатый процесс. Сначала объявляется ссылочная переменная на массив, затем выделяется память под требуемое количество элементов базового типа, и ссылочной переменной присваивается адрес нулевого элемента в массиве. Базовый тип определяет тип данных каждого элемента массива. Количество элементов, которые будут храниться в массиве, определяет размер массива.

В общем случае процесс объявления переменной типа массив и выделение необходимого объема памяти может быть разделен. Кроме того, на этапе объявления массива можно произвести его инициализацию. Поэтому для объявления одномерного массива может использоваться одна из следующих форм записи: тип[] имя_массива;

В этом случае описывается ссылка на одномерный массив, которая в дальнейшем может быть использована для адресации на уже существующий массив. Размер массива при таком объявлении не задается. Пример, в котором объявляется массив целых чисел с именем a: int[] a;

Другая форма объявления массива включает и его инициализацию указанным количеством элементов: тип[] имя_массива = new тип[размер];

В этом случае объявляется одномерный массив указанного типа и выделяется память под указанное количество элементов. Адрес данной области памяти записывается в ссылочную переменную. Элементы массива инициализируются значениями, которые по умолчанию приняты для данного типа: массивы числовых типов инициализируются нулями, строковые переменные – пустыми строками, символы – пробелами, объекты ссылочных типов – значением null. Пример такого объявления: int[] a = new int[10];

Здесь выделяется память под 10 элементов типа int.

Наконец, третья форма записи дает возможность сразу инициализировать массив конкретными значениями: тип[] имя_массива = {список инициализации};

При такой записи выделяется память под одномерный массив, размерность которого соответствует количеству элементов в списке инициализации. Адрес этой области памяти записан в ссылочную переменную. Значе-ние элементов массива соответствует списку инициализации. Пример: int[] a = { 0, 1, 2, 3 };

В данном случае будет создан массив a, состоящий из четырех элементов, и каждый элемент будет инициализирован очередным значением из списка.

Обращение к элементам массива происходит с помощью индекса: для этого нужно указать имя массива и в квадратных скобках – его номер. Например: a[0], b[10], c[i]. Следует помнить, что нумерация элементов начинается с нуля!

Так как массив представляет собой набор элементов, объединенных общим именем, то обработка массива обычно производится в цик-ле. Например:

```
int[] myArray = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }; for (int i = 0; i < 10; i++)
MessageBox.Show(myArray[i]);
```

Случайные числа

Одним из способов инициализации массива является задание элементов через случайные числа. Для работы со случайными числами используют класс Random. Метод Random.Next создает случайное число в диапазоне значений от нуля до максимального значения типа int (его можно узнать с помощью свойства Int32.MaxValue). Для создания случайного числа в диапазоне от нуля до какого-либо другого положительного числа используется перегрузка метода Random.Next(Int32) – единственный параметр метода указывает верхнюю границу диапазона, сама граница в диапазон не включается. Для создания случайного числа в другом диапазоне используется перегрузка метода Random.Next(Int32, Int32) – первый аргумент задает нижнюю границу диапазона, а второй – верхнюю.

Порядок выполнения индивидуального задания

Создайте форму с элементами управления, как показано на рис. 5.1. Опишите одномерный массив. Создайте обработчики события для кнопок (код приведен ниже). Данная программа заменяет все отрицательные числа нулями. Протестируйте правильность выполнения программы. Модифицируйте программу в соответствии с индивидуальным заданием.

```
//      Глобальная переменная видна всем методам int[] Mas = new int[15];
//      Заполнение исходного массива
private void button1_Click(object sender, EventArgs e){
//          Очищаем элемент управления listBox1.Items.Clear();
//          Инициализируем класс случайных чисел
Random rand = new Random();
//          Генерируем и выводим 15 элементов for (int i = 0; i < 15; i++){
Mas[i] = rand.Next(-50, 50);
listBox1.Items.Add("Mas[" + i.ToString() + "] = " + Mas[i].ToString());}}
//      Замена отрицательных элементов нулями
private void button2_Click(object sender, EventArgs e){
//          Очищаем элемент управления listBox2.Items.Clear();
//          Обрабатываем все элементы for (int i = 0; i < 15; i++){
if (Mas[i] < 0)
Mas[i] = 0;
listBox2.Items.Add("Mas[" + Convert.ToString(i) +"] = " + Mas[i].ToString());}}
```

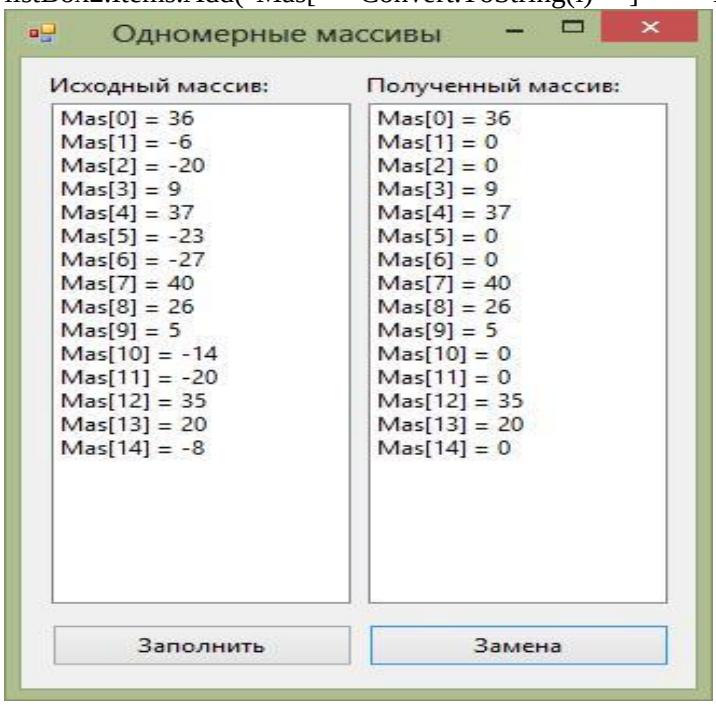


Рис. 5.1. Окно программы для работы с одномерными массивами

Задание на практическую работу

Индивидуальные задания

1. В массиве из 20 целых чисел найти наибольший элемент и поменять его местами с первым элементом.
2. В массиве из 10 целых чисел найти наименьший элемент и поменять его местами с предпоследним элементом.
3. Дан массив F, содержащий 18 элементов. Вычислить и вывести элементы нового массива по формуле $p_i = 0.13f_i^3 - 2.5f_i + 8$. Вывести отрицательные элементы массива P.
4. В массиве R, содержащем 25 элементов, заменить значения отрицательных элементов квадратами значений, значения положительных увеличить на 7, а нулевые значения оставить без изменения. Вывести массив R.
5. Дан массив A целых чисел, содержащий 30 элементов. Вычислить и вывести сумму тех элементов, которые кратны 5.
6. Дан массив A целых чисел, содержащий 30 элементов. Вычислить и вывести сумму тех элементов, которые нечетны и отрицательны.
7. Дан массив A целых чисел, содержащий 30 элементов. Вычислить и вывести количество и сумму тех элементов, которые делятся на 5 и не делятся на 7.
8. Дан массив A вещественных чисел, содержащий 25 элементов. Вычислить и вывести число отрицательных элементов и число членов, принадлежащих отрезку [1,2].
9. Дан массив Z целых чисел, содержащий 35 элементов. Вычислить и вывести R = S + P, где S – сумма четных элементов, меньших 3, P – произведение нечетных элементов, больших 1.
10. Дан массив Q натуральных чисел, содержащий 20 элементов. Найти и вывести те элементы, которые при делении на 7 дают остаток 1,2 или 5.
11. Дан массив, содержащий 10 элементов. Вычислить произведение элементов, стоящих после первого отрицательного элемента. Вывести исходный массив и результат вычислений.
12. Дан массив, содержащий 14 элементов. Вычислить сумму элементов, стоящих до первого отрицательного элемента. Вывести исходный массив и результат вычислений.
13. Дан массив, содержащий 12 элементов. Все четные элементы сложить, вывести массив и результат.
14. Дан массив, содержащий 15 элементов. Все положительные элементы возвести в квадрат, а отрицательные умножить на 2. Вывести исходный и полученный массив.
15. Дан массив, содержащий 14 элементов. Все отрицательные элементы заменить на 3. Вывести исходный и полученный массив.
16. Массив задан датчиком случайных чисел на интервале [–33, 66]. Найти наименьший нечетный элемент.
17. Разработать программу, выводящую количество максимальных элементов в массиве из пятидесяти целочисленных элементов.
18. Разработать программу, циклически сдвигающую элементы целочисленного массива влево. Нулевой элемент массива ставится на последнее место, остальные элементы сдвигаются влево на одну позицию. Запрещается использовать второй массив.
19. Дано два массива с неубывающими целыми числами. Напишите программу, формирующую новый массив из элементов первых двух. В результирующем массиве не должно быть одинаковых элементов.
20. Дан массив целых чисел из 30 элементов. Найдите все локальные максимумы. (Элемент является локальным максимумом, если он не имеет соседей, больших, чем он сам).

Многомерные массивы

Цель работы: изучить свойства элемента управления DataGridView. Написать программу с использованием двухмерных массивов.

Двухмерные массивы

Многомерные массивы имеют более одного измерения. Чаще всего используются двумерные массивы, которые представляют собой таблицы. Каждый элемент такого массива имеет два индекса, первый определяет номер строки, второй – номер столбца, на пересечении

которых находится элемент. Нумерация строк и столбцов начинается с нуля. Объявить двумерный массив можно одним из предложенных способов:

тип[,] имя_массива;

тип[,] имя_массива = new тип[размер1, размер2];

тип[,] имя_массива = {{элементы 1-ой строки}, ..., {элементы n-ой строки}};

тип[,] имя_массива = new тип[,] {{элементы 1-ой строки},

..., {элементы n-ой строки}};

8. качестве примера рассмотрим код, который строит «таблицу умножения» – каждая ячейка будет содержать значение, равное произведению номера строки и номера столбца:

9. Объявление двухмерного массива int[,] mul = new int[10,10];

10. Заполнение массива

```
for (int i = 0; i < 10; i++)
for (int j = 0; j < 10; j++)
mul[i, j] = i * j;
```

Элемент управления DataGridView

При работе с двухмерными массивами ввод и вывод информации на экран удобно организовывать в виде таблиц. Элемент управления DataGridView может быть использован для отображения информации в виде двумерной таблицы. Для обращения к ячейке в этом элементе необходимо указать номер строки и номер столбца. Например:

```
dataGridView1.Rows[2].Cells[7].Value = "*";
```

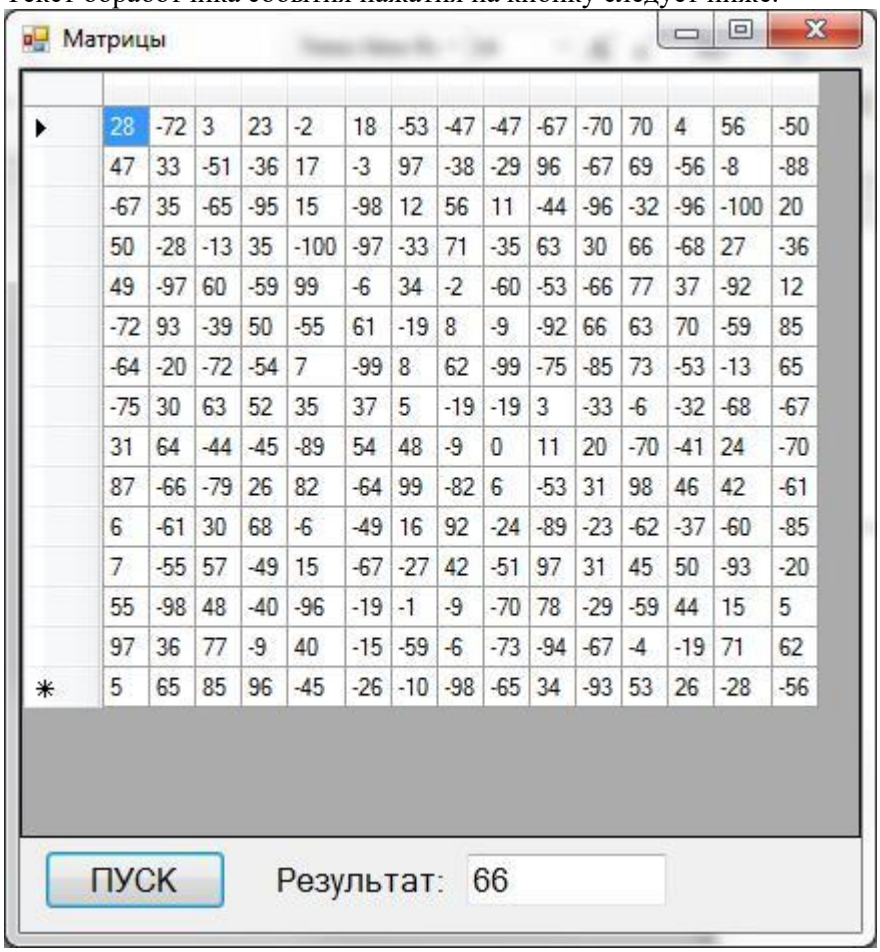
Этот код запишет во вторую строку и седьмой столбец знак звездочки.

Порядок выполнения задания

14.ходе выполнения задания нужно создать программу для определения целочисленной матрицы 15×15. Разработать обработчик кнопки, который будет искать минимальный элемент на дополнительной диагонали матрицы. Результат вывести в текстовое поле.

Окно программы приведено на рис. 5.2.

Текст обработчика события нажатия на кнопку следует ниже.



▶	28	-72	3	23	-2	18	-53	-47	-47	-67	-70	70	4	56	-50
	47	33	-51	-36	17	-3	97	-38	-29	96	-67	69	-56	-8	-88
	-67	35	-65	-95	15	-98	12	56	11	-44	-96	-32	-96	-100	20
	50	-28	-13	35	-100	-97	-33	71	-35	63	30	66	-68	27	-36
	49	-97	60	-59	99	-6	34	-2	-60	-53	-66	77	37	-92	12
	-72	93	-39	50	-55	61	-19	8	-9	-92	66	63	70	-59	85
	-64	-20	-72	-54	7	-99	8	62	-99	-75	-85	73	-53	-13	65
	-75	30	63	52	35	37	5	-19	-19	3	-33	-6	-32	-68	-67
	31	64	-44	-45	-89	54	48	-9	0	11	20	-70	-41	24	-70
	87	-66	-79	26	82	-64	99	-82	6	-53	31	98	46	42	-61
	6	-61	30	68	-6	-49	16	92	-24	-89	-23	-62	-37	-60	-85
	7	-55	57	-49	15	-67	-27	42	-51	97	31	45	50	-93	-20
	55	-98	48	-40	-96	-19	-1	-9	-70	78	-29	-59	44	15	5
	97	36	77	-9	40	-15	-59	-6	-73	-94	-67	-4	-19	71	62
*	5	65	85	96	-45	-26	-10	-98	-65	34	-93	53	26	-28	-56

Рис. 5.2. Окно программы для работы с двухмерным массивом

```
private void button1_Click(object sender, EventArgs e)
```

```
{
dataGridView1.RowCount = 15;                    // Кол-во строк
dataGridView1.ColumnCount = 15; // Кол-во столбцов int[,] a = new int[15,15]; // Инициализируем массив int i,j;
//Заполняем матрицу случайными числами Random rand = new Random();
for (i = 0; i < 15; i++)
for (j = 0; j < 15; j++)
a[i,j] = rand.Next(-100, 100);
1.          Выводим матрицу в dataGridView1 for (i = 0; i < 15; i++)
for (j = 0; j < 15; j++) dataGridView1.Rows[i].Cells[j].Value =
a[i, j].ToString();
2.          Поиск максимального элемента
3.          на дополнительной диагонали
int m = int.MinValue;
for (i = 0; i < 15; i++)
if (a[i, 14 – i] > m) m = a[i, 14 – i];
1.          выводим результат
textBox1.Text = Convert.ToString(m); }
```

Задание на практическую работу

Индивидуальные задания

1. Дана матрица A(3,4). Найти наименьший элемент в каждой строке матрицы. Вывести исходную матрицу и результаты вычислений.
2. Дана матрица A(3,3). Вычислить сумму второй строки и произведение первого столбца. Вывести исходную матрицу и результаты вычислений.
3. Вычислить сумму S элементов главной диагонали матрицы B(10,10). Если S>10, то исходную матрицу преобразовать по формуле $b_{ij} = b_{ij} + 13.5$; если S<=10, то $b_{ij} = b_{ij}^2 – 1.5$. Вывести сумму S и преобразованную матрицу.
4. Дана матрица F(15,15). Вывести номер и среднее арифметическое элементов строки, начинающейся с 1. Если такой строки нет, то вывести сообщение «Строки нет».
5. Дана матрица F(7,7). Найти наименьший элемент в каждом столбце. Вывести матрицу и найденные элементы.
6. Найти наибольший элемент главной диагонали матрицы A(15,15) и вывести всю строку, в которой он находится.
7. Найти наибольшие элементы каждой строки матрицы Z(16,16) поместить их на главную диагональ. Вывести полученную матрицу.
8. Найти наибольший элемент матрицы A(10,10) и записать нули в ту строку и столбец, где он находится. Вывести наибольший элемент, исходную и полученную матрицу.
9. Дана матрица R(9,9). Найти наименьший элемент в каждой строке и записать его на место первого элемента строки. Вывести исходную и полученную матрицы.
10. Вычислить количество H положительных элементов последнего столбца матрицы X(5,5). Если H < 3, то вывести все положительные элементы матрицы, если H>=3, то вывести сумму элементов главной диагонали матрицы.
11. Вычислить и вывести сумму элементов матрицы A(12,12), рас-положенных над главной диагональю матрицы.
12. Найти номер столбца матрицы, в котором находится наименьшее количество положительных элементов.
13. Дан двухмерный массив 20 × 20 целочисленных элементов. Найдите все локальные максимумы. (Элемент является локальным максимумом, если он не имеет соседей, больших, чем он сам).
14. Дана матрица 7 × 7. Найти наибольший элемент среди стоящих на главной и побочной диагоналях и поменять его местами с элементом, стоящим на пересечении этих диагоналей.
15. Задана матрица, содержащая N строк и M столбцов. Седловой точкой этой матрицы назовем элемент, который одновременно является минимумом в своей строке и максимумом в своем столбце. Найдите количество седловых точек заданной матрицы.
16. Дана квадратная матрица 10 × 10. Реализуйте программу для транспонирования матрицы по главной и побочной диагоналям.

17. Требуется совершить обход квадратной матрицы по спирали так, как показано на рисунке: заполнение происходит с единицы из левого верхнего угла и заканчивается в центре числом N^2, где N – порядок матрицы. Реализуйте программу для матрицы 10 × 10.

1	2	3	4	5
16	17	18	19	6
15	24	25	20	7
14	23	22	21	8
13	12	11	10	9

18. Требуется заполнить змейкой квадратную матрицу так, как показано на рисунке: заполнение происходит с единицы из левого верхнего угла и заканчивается в правом нижнем числом N^2, где N – порядок матрицы. Реализуйте программу для матрицы 10 × 10.

1	3	4	10
2	5	9	11
6	8	12	15
7	13	14	16

19. Дана шахматная доска (матрица 8 × 8). Разработать программу, показывающую последовательность ходов конем с произвольной клетки. Конь ходит в соответствии с шахматными правилами, но в произвольную сторону (сгенерировать случайным образом). В клетку, с которой начинается ход, выводится единица. В клетку, в которую идет далее конь, записывается двойка и т. д. Ходить конем на клетки, на которых уже побывал конь, нельзя. Алгоритм останавливает работу, когда конем ходить некуда. Максимальная последовательность ходов – 64.

20. Проверка на симпатичность. Рассмотрим таблицу, содержащую n строк и m столбцов, в каждой клетке которой расположен ноль или единица. Назовем такую таблицу симпатичной, если в ней нет ни одного квадрата 2 на 2, заполненного целиком нулями или целиком единицами. Так, например, таблица 4 на 4, расположенная слева, является симпатичной, а расположенная справа таблица 3 на 3 – не является.

1	0	1	0
1	1	1	0
0	1	0	1
0	0	0	0

0	0	1
0	0	1
1	1	1

Практическая работа №6 «Работа со строками»

Цель работы: изучить правила работы с элементом управления ListBox. Написать программу для работы со строками.

Строковый тип данных

Для хранения строк в языке C# используется тип string. Чтобы объявить (и, как правило, сразу инициализировать) строковую переменную, можно написать следующий код:

```
string a = "Текст";
string b = "строки";
```

Над строками можно выполнять операцию сложения – в этом случае текст одной строки будет добавлен к тексту другой:

```
string c = a + " " + b; // Результат: Текст строки
```

Тип string на самом деле является псевдонимом для класса String, с помощью которого над строками можно выполнять ряд более сложных операций. Например, метод IndexOf может осуществлять поиск подстроки в строке, а метод Substring возвращает часть строки указанной длины, начиная с указанной позиции:

```
string a = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
int index = a.IndexOf("OP"); // Результат: 14 (счет с 0)
string b = a.Substring(3, 5); // Результат: DEFGH
```

Если требуется добавить в строку специальные символы, это можно сделать с помощью escape-последовательностей, начинающихся с обратного слэша:

- \" – Кавычка.
- \\ – Обратная косая черта.
- \n – Новая строка.
- \r – Возврат каретки.
- \t – Горизонтальная табуляция.

Более эффективная работа со строками

Строки типа string представляют собой неизменяемые объекты: после того, как строка инициализирована, изменить ее уже нельзя. Рассмотрим для примера следующий код:

```
string s = "Hello, ";
s += "world!";
```

Здесь компилятор создает в памяти строковый объект и инициализирует его строкой «Hello, », а затем создает другой строковый объект и инициализирует его значением первого объекта и новой строкой «world!», а затем заменяет значение переменной s на новый объект. В результате строка s содержит именно то, что хотел программист, од-нако в памяти остается и изначальный объект со строкой «Hello, ». Конечно, со временем сборщик мусора уничтожит этот бесхозный объект, однако если в программе идет интенсивная работа со строками, то таких бесхозных объектов может оказаться очень много. Как правило, это негативно сказывается на производительности программы и объеме потребляемой ею памяти.

Чтобы компилятор не создавал каждый раз новый строковый объект, разработчики языка C# ввели другой строковый класс: StringBuilder. Приведенный выше пример с использованием этого класса будет выглядеть следующим образом:

```
StringBuilder s = new StringBuilder("Hello, "); s.Append("world!");
```

Конечно, визуально этот код выглядит более сложным, зато при активном использовании строк в программе он будет гораздо эффективнее. Помимо добавления строки к существующему объекту (Append) класс StringBuilder имеет еще ряд полезных методов:

- Insert: вставляет указанный текст в нужную позицию исходной строки
- Remove: удаляет часть строки
- Replace: заменяет указанный текст в строке на другой.

Если нужно преобразовать объект StringBuilder в обычную строку, то для этого можно использовать метод ToString():

```
StringBuilder s = new StringBuilder("Яблоко"); string a = s.ToString();
```

Элемент управления ListBox

Элемент управления ListBox представляет собой список, элементы которого выбираются при помощи клавиатуры или мыши. Список эле-ментов задается свойством Items. Items – это элемент, который имеет свои свойства и свои методы. Методы Add, RemoveAt и Insert используются для добавления, удаления и вставки элементов.

Объект Items хранит объекты, находящиеся в списке. Объект может быть любым классом – данные класса преобразуются для отображения в строковое представление методом ToString(). В нашем случае в качест-ве объекта будут выступать строки. Однако, поскольку объект Items хра-нит объекты, приведенные к типу object, перед использованием необхо-димо привести их обратно к изначальному типу, в нашем случае string:

string a = (string)listBox1.Items[0];

Для определения номера выделенного элемента используется свойство SelectedIndex.

Порядок выполнения индивидуального задания

Задание: Написать программу подсчета числа слов в произвольной строке. В качестве разделителя может быть любое число про-белов. Для ввода строк использовать ListBox. Строки вводятся на этапе проектирования формы, используя окно свойств. Вывод результата ор-ганизовать в метку Label.

Панель диалога будет иметь вид:

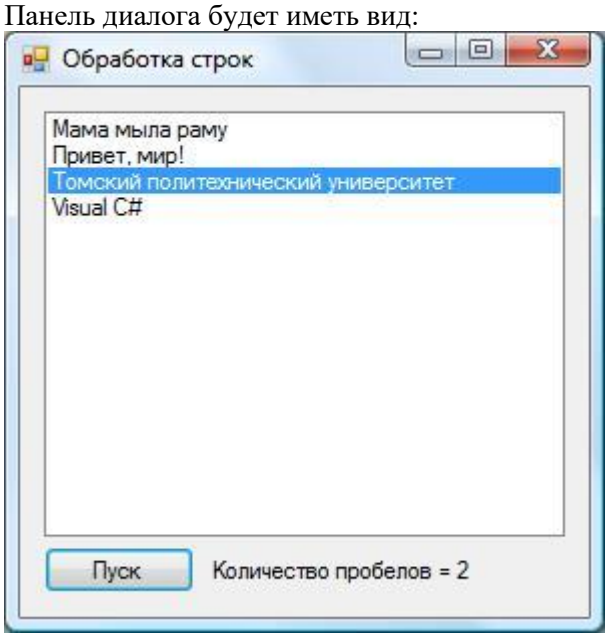


Рис. 6.1. Окно программы обработки строк

Текст обработчика нажатия кнопки «Пуск» приведен ниже.

```
private void button1_Click(object sender, EventArgs e)
{
Получаем номер выделенной строки int index = listBox1.SelectedIndex;
Считываем строку в переменную str
string str = (string)listBox1.Items[index];
Узнаем количество символов в строке int len = str.Length;
Считаем, что количество пробелов равно 0 int count = 0;
Устанавливаем счетчик символов в 0
int i = 0;
1              Организуем цикл перебора всех символов в строке while (i < len)
{
Если нашли пробел, то увеличиваем
счетчик пробелов на 1
if (str[i] == ' ')
count++;
i++;
}
label1.Text = "Количество пробелов = " + count.ToString();
}
```

Задание на практическую работу

Во всех заданиях исходные данные вводить с помощью ListBox. Строки вводятся на этапе проектирования формы, используя окно свойств. Вывод результата организовать в метку Label.

Индивидуальные задания

1. Дана строка, состоящая из групп нулей и единиц. Посчитать количество нулей и единиц.
2. Посчитать в строке количество слов.
3. Найти количество знаков препинания в исходной строке.
4. Дана строка символов. Вывести на экран цифры, содержащиеся в строке.
5. Дана строка символов. Сформировать новую строку, в которую включить все символы исходной строки, стоящие на четных местах. При этом должен быть обратный порядок следования символов по от-ношения к исходной строке.
6. Сформировать и вывести две новых строки на основе входной строки символов. В первую строку включить все символы, стоящие на четных местах, во вторую – символы, стоящие на нечетных местах в исходной строке.
7. Дана строка символов, состоящая из произвольных десятичных цифр, разделенных пробелами. Вывести количество четных чисел в этой строке.
8. Дана строка символов. Вывести на экран количество строчных русских букв, входящих в эту строку.
9. Сформировать и вывести три новых строки на основе входной строки символов. В первую строку включить все цифры, во вторую – все знаки препинания: точки, запятые, двоеточия, точки с запятой, восклицательные и вопросительные знаки, в третью строку – все остальные символы. Например, входная строка содержит: «выходные дни: 1, 2 ян-варя, 8 марта, 1 мая, 9 мая!», после обработки первая строка будет со-держать: «12819», вторая строка: «:,,,!», третья строка: «выходные дни января марта мая мая».
10. Дана строка символов. Вывести на экран только строчные русские буквы, входящие в эту строку.
11. Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. В каждом слове заме-нить первую букву на прописную.
12. Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Удалить первую бук-ву в каждом слове.
13. Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Поменять местами i и j-ю буквы. Для ввода i и j на форме добавить свои поля ввода.
14. Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Заменить все буквы ла-тинского алфавита на знак «+».
15. Дана строка символов, содержащая некоторый текст на рус-ском языке. Заменить все большие буквы «А» на символ «*».
16. Дана строка символов, содержащая некоторый текст. Разработать программу, которая определяет, является ли данный текст палиндромом, т. е. читается ли он слева направо так же, как и справа налево (например, «А роза упала на лапу Азора»).
17. Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Сформировать новую строку, состоящую из чисел длин слов в исходной строке.
18. Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Поменять местами первую и последнюю буквы каждого слова.
19. Поменять местами первое и второе слово в исходной строке.
20. Сформировать новую строку, где поменять местами первое и последнее слово из исходной строки.

Практическая работа № 7 «Работа с данными типа множество»

Цель работы: Овладеть техникой составления программ для работы с множествами, её компиляции и записи на диск под заданным именем.

Операции над множествами в LINQ — это операции запросов, результирующие наборы которых основываются на наличии или отсутствии эквивалентных элементов в одной или другой коллекции (или наборе).

Каждый отдельный элемент множества не идентифицируется и с ним нельзя выполнять какие-либо действия. Действия могут выполняться только над множеством в целом.

Описание множеств и операции над ними Описывают множества, используя ключевое слово set of: Var M: set of Char;

Присвоить значение переменной множественного типа можно так: M:= [‘a’..’y’]

Для того, чтобы определить, принадлежит ли элемент множеству, используют зарезервированное слово in: If ‘s’ in M then…

Над множествами можно выполнять операции:

- Объединения (+);
- Пересечения (*);
- Разности (-);
- Операции отношений (=, <>, , <=, >=);
- Проверка принадлежности элемента множеству.

Методы стандартных операторов запросов, которые выполняют операции над множествами, перечислены ниже:

Таблица 1. Методы

Имя метода	**Описание**	**Синтаксис выражения запроса C#**	**Дополнительные сведения**
Distinct	Удаляет повторяющиеся значения из коллекции.	Не применяется	Enumerable.Distinct Queryable.Distinct
Исключения	Возвращает разность множеств, т. Е. Элементы одной коллекции, которые отсутствуют во второй.	Не применяется	Enumerable.Except Queryable.Except
Пересечение	Возвращает пересечение множеств, т. Е. Элементы, присутствующие в каждой из двух коллекций.	Не применяется	Enumerable.Intersect Queryable.Intersect
Объединение	Возвращает объединение множеств, т. Е. Уникальные элементы, присутствующие в одной из двух коллекций.	Не применяется	Enumerable.Union Queryable.Union

В Pascal отсутствуют средства ввода-вывода элементов множества, поэтому работу

Программы необходимо проверять, выполняя ее в пошаговом режиме и отслеживая изменения

Значений переменных в окне просмотра:

1. С помощью команды ***Debug/Add*** watch задать переменные, значения которых

Необходимо наблюдать.

2. Открыть окно наблюдаемых значений, выбрав команду ***Debug/Watch***, а окно редактора

Уменьшить так, чтобы окна не перекрывали друг друга.

3. Установить курсор на точку программы, до которой она выполняется правильно

4. Запустить программу до этой точки, выбрав команду ***Run/Go to cursor (F4)***, а затем

Выполнять ее по шагам с помощью команды ***Run/Step over (F8),*** наблюдая в окне

Значения переменных.

5. Для прерывания процесса отладки выбрать команду ***Run/Program reset (Ctrl+F2)***

Пример

```
Program Dem_Mno; {Демонстрация операций над множествами}
Type
Digits=set of 0..9;
```

```
Var
D1, d2,d3,d: digits;
Begin
D1:=[2, 4,6,8]; {Заполнение множеств}
D2:=[0..3,5];
D3:=[1,3,5,7,9];
D:=d1+d2; {Объединение множеств d1и d2}
D:=d+d3; {Объединение множеств d и d3}
D:=d-d2; {Разность множеств d и d2}
D:=d*d1; {Пересечение множеств d и d1}
End.
```

Например, используя операцию объединения, к множеству M можно добавить символ ‘z’, представив его как множество из одного элемента: M:= M+[‘z’]. Либо эту же операцию выполняет процедура Include(M, ‘z’). Процедура Exclude предназначена для исключения элемента из множества. Например, Exclude(M, ‘s’) исключит символ ‘s’ из множества M. Константы множественного типа записывают с помощью квадратных скобок и списка эле- ментов: Abc=[‘A’, ‘B’, ‘C’]; {Множество из трех букв} Cifr=[0..9]; {Множество цифр} P=[] – пустое множество Особенности множества: 1) В нем могут содержаться элементы только одного базового типа. 2) Порядок элементов множества не фиксируется: {7, 3,1}. 3) В множестве не может быть одинаковых элементов. 18 2. Выполните следующие упражнения: 1. Заполнить множество NB случайными значениями в пределах от 0 до 49. Количество элементов множества запрашивается с клавиатуры: uses crt; Var nb : set of 1..200; k, n, i: integer; Begin clrscr; nb :=[]; {Создаем пустое множество} randomize; {Подключаем процедуру случайных чисел} Writeln('Введите количество элементов множества''); Readln(k); For i:=1 to k do begin n:=Random(50); nb:=nb+[n]; Write(n:3); end; end. Сохраните программу в свою папку на диск D: P7PR1 2. Дано множество A, содержащее знаки препинания английского алфавита. Вывести множество A на экран: var A: set of char; ch: char; Begin clrscr; A:=['.',',',';',':','!','?']; Writeln('A='); For ch:=#0 to #127 do if ch in A then Write(ch:3); Readln; end. Сохраните программу в свою папку на диск D: P7PR2 3. Выполните самостоятельно 1. Составить программу, которая моделирует «лототрон (5 из 36)» т.е. Случайную выборку 5 шаров из контейнера, содержащего 36 шаров, пронумерованных от единицы до 36. Сохранить программу под именем P7PR3. 2. Дано множество A – множество гласных букв английского алфавита и множество B, состоящее из символов английского алфавита: a, o, !, ?. Вывести на экран исходные множества. Найти множество C, являющимся объединением A и B. Сохранить программу под именем P7PR4.

Задание на практическую работу

1. Опишите множество Pr(1..20) и поместите в него все простые числа в диапазоне 1..20. Составьте блок-схему.
2. Опишите множество Alf(‘a’..’я’) и поместите в него гласные буквы. Составьте блок-схему.
3. Опишите множества M1(1,2) и M2(2,1). Сравните множества M1 и M2 на равенство. Составьте блок-схему.
4. Опишите множества M1(‘a’, ‘b’) M2(‘b’, ’a’, ‘c’). Сравните два этих множества на неравенство. Составьте блок-схему.
5. Опишите множества M1(‘a’, ‘b’,’c’) M2(‘a’, ‘c’). Сравните два этих множества с использованием операции >=. Составьте блок-схему.
6. Опишите множества M1(1, 2, 3) M2(1, 2, 3, 4). Сравните два этих множества с использованием операции <=. Составьте блок-схему.
7. Опишите множества M1(1, 2) M2(5, 6). Получите результирующее множество M3=M1-M2. Определите содержится в M3 элемент 7. Составьте блок-схему.
8. Опишите множества M1(1, 2, 3,4) M2(3, 4, 1). Получите результирующее множество M3=M1-M2. Определите содержится в M3 элемент 7. Составьте блок-схему.
9. Опишите множества M1(1, 2, 3) M2(1, 4, 2, 5). Получите результирующее множество M3=M1*M2. Определите содержится в M3 элемент 2. Составьте блок-схему.
10. Опишите множества M1(1, 2) M2(5, 6). Получите результирующее множество M3=M1+M2. Определите содержится в M3 элемент 2. Составьте блок-схему.

Практическая работа №8 «Файлы последовательного доступа. Типизированные файлы»

Цель работы: Получение навыков в работе с функциями.

Понятие и назначение функций

Любая С++-программа составляется из "строительных блоков", именуемых функциями. Функция — это подпрограмма, которая содержит одну или несколько С++-инструкций и выполняет одну задачу.

Каждая функция имеет имя, которое используется для ее вызова. Своим функциям программист может давать любые имена за исключением имени main(), зарезервированного для функции, с которой начинается выполнение программы.

Функции — это "строительные блоки" С++-программы.

В С++ ни одна функция не может быть встроена в другую. Одна функция может вызывать другую.

В уже рассмотренных примерах программ функция main() была единственной, функция main () — первая функция, выполняемая при запуске программы. Ее должна содержать каждая С++-программа. Функции бывают двух типов. К первому типу относятся функции, написанные программистом (main() — пример функции такого типа). Функции другого типа находятся в стандартной библиотеке С++-компилятора -. Коллекция встроенных функций. Как правило, С++-программы содержат как функции, написанные программистом, так и функции, предоставляемые компилятором.

Эта программа содержит две функции: main() и myfunc().

```
Void myfunc();          // прототип функции myfunc()
Main()
{
Cout << "В функции main() . " ;
Myfunc();               // Вызываем функцию myfunc() •
Cout << "Снова в функции main () ";
}
Void myfunc()
{
Cout << " В функции myfunc() . ";
}
```

Программа работает следующим образом. Вызывается функция main() и выполняется ее первая cout-инструкция. Затем из функции main() вызывается функция myfunc (). Для вызова функции в программе указывается имя функции myfunc, за которым следуют пара круглых скобок и точка с запятой. Затем функция myfunc() выполняет свою единственную cout-инструкцию и передает управление назад функции main(), причем той строке кода, которая расположена непосредственно за вызовом функции. Наконец, функция main() выполняет свою вторую cout-инструкцию, которая завершает всю программу. Результаты работы программы:

В функции main() . В функции myfunc() . Снова в функции main().

Прототип функции

Прототип функции объявляет функцию до ее определения и использования. Прототип позволяет компилятору узнать тип значения, возвращаемого этой функцией, а также количество и тип параметров, которые она может иметь. Компилятору нужно знать эту информацию до первого вызова функции. Поэтому прототип располагается до функции main (). Единственной функцией, которая не требует прототипа, является main(), поскольку она встроена в язык С++.

```
Void myfunc();          // прототип функции myfunc()
```

Функция myfunc () не содержит инструкцию return. Ключевое слово void, которое предваряет как прототип, так и определение функции myfunc(),заявляет о том, что функция myfunc () не возвращает никакого значения. В С++ функции, не возвращающие значений, объявляются с использованием ключевого слова void.

Пример программы с функциями без параметров

Составить программу с функцией, которая рисует на экране строку, состоящую из 80 звездочек.

```
Void line(void);                    //Прототип функции line
Main()                          //Основная функция
{
```

```
Line();                          //Вызов функции line
}
//Определение функции line
Void line(void)
{          int   i;
For (i=0;   i<80;   i + +)
Cout<<"*";
}
```

Задание на практическую работу

1. Составить программу с тремя функциями, выводящими на экран строки символов #, @, $.
2. Составить программу для вычисления площади кольца по значениям внутреннего и внешнего радиусов, используя функцию вычисления площади круга.
3. Даны три целых числа. Определить, сумма цифр которого из них больше. Подсчет суммы цифр организовать через функцию.
4. Создать типизированный файл, содержащий данные о студентах группы: фамилия и инициалы (одно поле записи), год рождения, адрес (улица, дом, квартира), средний балл при поступлении. Переписать в текстовый и вывести в ячейки эл. Таблицы данные о студентах со средним проходным баллом, большим 3,8;
5. Создать типизированный файл, содержащий данные о редких книгах, хранящихся с библиотеке: название, автор (фамилия и инициалы), год издания, место издания, инвентарный номер. Переписать в текстовый файл и вывести в ячейки эл. Таблицы данные о книгах, изданных в Санкт-Петербурге;
6. Создать типизированный файл, содержащий данные о рейсах самолетов: номер рейса, пункт назначения, время в пути (дробное число), тип самолета, время отправления (два поля записи: часы и минуты). Переписать в текстовый файл и вывести в ячейки эл. Таблицы данные о рейсах, вылетающих после десяти вечера, но до полуночи;
7. Создать типизированный файл, содержащий данные о автобусных маршрутах: пункт отправления, пункт назначения, время в пути (дробное число), время отправления (два поля записи: часы и минуты), количество остановок в пути. Переписать в текстовый файл и вывести в ячейки эл. Таблицы данные о рейсах, делающих более трех остановок в пути;
8. Создать типизированный файл, содержащий данные о наименованиях продукции молокозавода, поступивших в продажу: название продукта, дата изготовления (три поля записи: год, месяц и число), срок хранения в днях, закупочная цена (дробное число). Переписать в текстовый файл и вывести в ячейки эл. Таблицы данные о продуктах, выпущенный в июле 2004 года;
9. Создать типизированный файл, содержащий данные о фирмах: название фирмы, фамилия и инициалы владельца (одно поле), адрес (три поля: город, улица, дом), телефон. Переписать в текстовый файл и вывести в ячейки эл. Таблицы данные о владельцах фирм, фамилия которых начинается с буквы М
10. Создать типизированный файл, содержащий данные о автомашинах, находящихся в розыске: марка, цвет, год выпуска, номер, дата угона (три поля: число, месяц, год). Переписать в текстовый файл и вывести в ячейки эл. Таблицы данные о машинах, угнанных в июле 2002 года;
11. Создать типизированный файл, содержащий данные о сотрудниках фирмы: фамилия, имя, отчество, год рождения, должность, год поступления на работу. Переписать в текстовый файл и вывести в ячейки эл. Таблицы данные о сотрудниках фирмы, устроившихся на работу в период с 1999 по 2002 год включительно;
12. Создать типизированный файл, содержащий данные о спортсменах-пловцах: фамилия и инициалы, пол, год рождения, рост, вес, лучшее время, за которое спортсмен проплывает 50 метров. Переписать в текстовый файл и вывести в ячейки эл. Таблицы данные о спортсменах, рост которых больше 175 см;
13. Создать типизированный файл, содержащий данные о клиентах ателье: фамилия и инициалы, адрес (три поля: улица, дом, квартира), вид заказа, стоимость заказа. Переписать в текстовый файл и вывести в ячейки эл. Таблицы данные о клиентах, проживающих на улице Международная.

Практическая работа №9 «Нетипизированные файлы»

Цель работы: Получение навыков в работе с функциями.

Назначение аргументов функции

Функции можно передать одно или несколько значений. Значение, передаваемое функции, называется аргументом. **Аргумент** — это значение, передаваемое функции при вызове.

Рассмотрим программу, которая для отображения абсолютного значения числа использует стандартную библиотечную функцию abs(). Эта функция принимает один аргумент, преобразует его в абсолютное значение и возвращает результат.

```
Main()
{
Cout << abs(-10);
}
```

Здесь функции abs() в качестве аргумента передается число -10. Функция abs() принимает этот аргумент при вызове и возвращает его абсолютное значение, которое в данном случае равно числу 10. Если функция принимает аргумент, он указывается внутри круглых скобок, расположенных сразу после имени функции.

Рассматриваемая программа включает заголовок <stdlib.h>. Он необходим для обеспечения возможности вызова функции abs(). Каждый раз, когда используется библиотечная функция, в программу необходимо включать соответствующий заголовок, который помимо прочей информации, содержит прототип библиотечной функции.

Передаваемые параметры

Параметр — это определяемая функцией переменная, которая принимает передаваемый функции аргумент.

При создании функции, которая принимает один или несколько аргументов, иногда необходимо объявить переменные, которые будут хранить значения аргументов. Эти переменные называются параметрами функции. Например, следующая функция выводит произведение двух целочисленных аргументов, передаваемых функции при ее вызове.

```
Void mul(int x, int y)
{
Cout << x * y << " ";
}
```

При каждом вызове функции mul() выполняется умножение значения, переданного параметру x, на значение, переданное параметру y. Рассмотрим программу, которая демонстрирует использование функции mul().

```
Void mul(int x, int y);              // Прототип функции mul() .
Main()
{
Mul(10, 2 0);    //Вызовы функции с параметрами
Mul(5, 6);
Mul (8, 9);
}
```

Эта программа выведет на экран числа 200, 30 и 72. При вызове функции mul() C++-компилятор копирует значение каждого аргумента в соответствующий параметр. В данном случае при первом вызове функции mul() число 10 копируется в переменную x, а число 20 — в переменную y. При втором вызове 5 копируется в x, а 6 — в y. При третьем вызове 8 копируется в x, а 9 — в y.

3.3. Функции, возвращающие значения

В C++ для возврата значения из функции используется инструкция return. Общий формат этой инструкции таков:

Return значение;

Значение представляет собой значение-результат, возвращаемое функцией.

Исправим функцию mul() так, чтобы она возвращала результат - произведение своих аргументов.

```
Int mul(int x, int y);               // Прототип функции mul() с результатом
Main()
{        int answer;
Answer = mul(10, 11);            //Вызов с возвратом значения
```

```
Cout << "Ответ равен " <<  answer;
}
// Эта функция возвращает значение.
Int mul(int x, int y)
{
Return x * y;                    // Функция возвращает произведение x и y.
}
```

В этом примере функция mul() возвращает результат вычисления выражения x*y с помощью инструкции return. Затем значение этого результата присваивается переменной answer.

Поскольку в этой версии программы функция mul () возвращает значение, ее имя в определении не предваряется словом void. Здесь функция mul() возвращает значение целочисленного типа. Тип значения, возвращаемого функцией, предшествует ее имени, как в прототипе, так и в определении.

При достижении инструкции return функция немедленно завершается, а весь остальной код игнорируется. Функция может содержать несколько инструкций return. Возврат из функции можно обеспечить с помощью инструкции return без указания возвращаемого значения, но такую ее форму допустимо применять только для функций, которые не возвращают никаких значений и объявлены с использованием ключевого слова void.

Пример программы с функциями с параметрами

Составить программу нахождения наибольшего значения из трех величин — max (a, b, c). Для ее решения можно использовать вспомогательный алгоритм нахождения максимального значения из двух, поскольку справедливо равенство: max (a, b, c) = max (max (a, b), c).

```
//Определение вспомогательной функции
Int MAX(int x, int y)
{
If (x>y)
Return x;
Else
Return y;
}
//Основная функция
Main()
{        int a,b,c,d;
Cout<<"Введите  a,b,c:";
Cin>>a>>b>>c;
D=MAX (MAX (a, b), c);
Cout<<"\nmax (a, b, c) ="<<d;
}
```

Задание на практическую работу

1. Составить программу с функцией, выводящей на экран строку нужной длины нужным символом. Функция принимает два параметры – длину строки и символ. Возвращаемых значений нет.
2. Составить программу для вычисления площади кольца по значениям внутреннего и внешнего радиусов, используя функцию вычисления площади круга. Функции передается один параметр – радиус и возвращается результат – вычисленная площадь.
3. Даны три целых числа. Определить, сумма цифр которого из них больше. Подсчет суммы цифр организовать через функцию, которая имеет два входных параметра – складываемые числа и результат – вычисленную сумму.

Практическая работа № 10 «Организация процедур. Организация функций»

Цель работы: Сформировать умения по использованию процедурного языка программирования для построения логически правильных и эффективных программ с использованием стандартных функций и процедур для работы со строками.

Тип **string**, предназначенный для работы со строками символов в кодировке **Unicode**, является встроенным типом **C#**. Ему соответствует базовый класс **System.String** библиотеки **.Net**. Тип **string** относится к ссылочным типам, хотя работа с ним во многом напоминает работу с размерными типами. С объектом типа **string** можно работать посимвольно, т.е. Поэлементно. Класс **string** обладает богатым набором методов для сравнения строк, поиска в строке и других действий со строками. Все методы возвращают ссылку на новую строку, созданную в результате преобразования копии исходной строки. Для того чтобы сохранить данное преобразование, нужно установить на него новую ссылку. Область применения типа **string** – это поиск, сравнение, извлечение информации из строки.

Создайте новое консольное приложение с именем **Zad10**. Составьте программу для нахождения количества вхождений символа в строке и протестируйте её.

```
static void Main(string[] args)
{
    string a = "кол около колокола";
    Console.WriteLine("Дана строка: {0}", a);
    char b = 'o';
    int k = 0;
    for (int x=0; x<a.Length; x++)
    {
        if (a[x]==b)
        {
            k++;
        }
    }
    Console.WriteLine("Символ {0} содержится в ней {1} раз", b, k);
}
```

Рисунок 10.1. Код программы

3. Попробуем заменить все вхождения буквы о на букву а.

```
static void Main(string[] args)
{
    string a = "кол около колокола";
    Console.WriteLine("Дана строка: {0}", a);
    char b = 'o';
    char c = 'a';
    int k = 0;
    for (int x=0; x<a.Length; x++)
    {
        if (a[x]==b)
        {
            a[x]=c;
        }
    }
    Console.WriteLine("Символ {0} содержится в ней {1} раз", b, k);
}
```

Рисунок 10.2. Код программы

Запустить программу будет не возможно. Компилятор запрещает напрямую изменять значение строки.

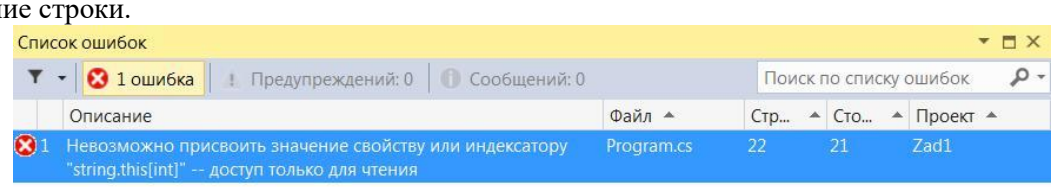


Рисунок 10.3. Ошибка

Строковый тип **stringbuilder** определен в пространстве имен **System.Text** и предназначен для создания строк, значение которых можно изменять. Объекты данного класса всегда создаются с помощью явного вызова конструктора класса, т.е. Через операцию **new**. С объектами класса **stringbuilder** можно работать посимвольно. Внесите изменения в созданную ранее программу следующим образом:

```
static void Main(string[] args)
{
    StringBuilder a = new StringBuilder ("кол около колокола");
    Console.WriteLine("Дана строка: {0}", a);
    char b = 'o';
    char c = 'a';
    int k = 0;
    for (int x=0; x<a.Length; x++)
    {
        if (a[x]==b)
        {
            k++;
        }
    }
    Console.WriteLine("Символ {0} содержится в ней {1} раз", b, k);
}
```

Рисунок 10.4 Код программы

Протестируйте работу программы. Она ничем не отличается от аналогичной программы с типом **string**.

Измените программу так, чтобы все вхождения буквы о были заменены на букву а. Возникает ли ошибка при работе данной программы.

```
static void Main(string[] args)
{
    StringBuilder a = new StringBuilder ("кол около колокола");
    Console.WriteLine("Дана строка: {0}", a);
    char b = 'o';
    char c = 'a';
    int k = 0;
    for (int x=0; x<a.Length; x++)
    {
        if (a[x]==b)
        {
            a[x]=c;
        }
    }
    Console.WriteLine("Преобразованная строка: {0}", a);
}
```

Риунок 10.5. Код программы

Завершите работу с программой.

Практическая работа №11 «Применение рекурсивных функций»

Цель работы: Получение навыков в работе с рекурсивными функциями.

Понятие рекурсивной функции

Рекурсией называется ситуация, когда подпрограмма вызывает сама себя. Функция может содержать вызов других функций. В том числе процедура может вызвать саму себя. Пример рекурсивной функции:

```
Int Rec(int a) { If (a>0) Rec(a-1); Cout<<a; }
```

Функция Rec вызывается с параметром a = 3. В ней содержится вызов функции Rec с параметром a = 2. Предыдущий вызов еще не завершился, поэтому создается еще одна функция и до окончания ее работы первая свою работу не заканчивает. Процесс вызова заканчивается, когда параметр a = 0. В этот момент одновременно выполняются 4 экземпляра функции. Количество одновременно выполняемых функций называют **глубиной рекурсии**. Четвертая вызванная процедура (Rec(0)) напечатает число 0 и закончит свою работу. После этого управление возвращается к функции, которая ее вызвала (Rec(1)) и печатается число 1. И так далее пока не завершатся все вызовы. Результатом исходного вызова будет печать четырех чисел: 0, 1, 2, 3.

Пример программы с рекурсивной функцией

Если функция вызывает сама себя, то, по сути, это приводит к повторному выполнению содержащихся в ней инструкций, что аналогично работе цикла.

Для примера сымитируем работу цикла for. Для этого потребуется переменная счетчик шагов, которую можно реализовать как параметр функции.

```
//имитация работы цикла с помощью рекурсии
Void   loopimit(int i, int n)
{
Cout<<"\n"<<"  1 n "<<" i "<<i;
//Здесь может располагаться первый блок инструкций
If (i<=n)
Loopimit(i+1, n);
Cout<<"\n"<<"  2 n "<<" i "<<i;
//Здесь может располагаться второй блок инструкций
}
Main()
{          int i=0, n;
Cout<<"n=";
Cin>>n;
Loopimit(i, n);
}
```

Результат работы программы:

```
N=5
1 n  i 0
1 n  i 1
1 n  i 2
1 n  i 3
1 n  i 4
1 n  i 5
1 n  i 6
2 n  i 6
2 n  i 5
2 n  i 4
2 n  i 3
2 n  i 2
2 n  i 1
2 n  i 0
```

Задание на практическую работу

1. Разработать рекурсивную функцию вычисления факториала.
2. Разработать рекурсивную функцию, которая для двух заданных натуральных чисел определяет их наибольший общий делитель.
3. Разработать рекурсивную функцию, определяющую число Фибоначчи с номером n.

Практическая работа №12 «Программирование модуля»

Цель работы: Получение навыков в работе с рекурсивными функциями.

Понятие рекурсивной функции

Рекурсией называется ситуация, когда подпрограмма вызывает сама себя. Функция может содержать вызов других функций. Пример рекурсивной функции:

```
Int Rec(int a)
{
If (a>0) Rec(a-1);
Cout<<a;
}
```

Рассмотрим принцип работы рекурсивной функции на примере вызова созданной функции Rec(3).

Функция Rec вызывается с параметром a = 3. В ней содержится вызов функции Rec с параметром a = 2. Предыдущий вызов еще не завершился, поэтому создается еще одна функция и до окончания ее работы первая свою работу не заканчивает. Процесс вызова заканчивается, когда параметр a = 0. В этот момент одновременно выполняются 4 экземпляра функции. Количество одновременно выполняемых процедур называют **глубиной рекурсии**. Четвертая вызванная функция (Rec(0)) напечатает число 0 и закончит свою работу. После этого управление возвращается к функции, которая ее вызвала (Rec(1)) и печатается число 1. И так далее пока не завершатся все вызовы. Результатом исходного вызова будет печать четырех чисел: 0, 1, 2, 3.

Задача о Ханойских башнях

Существует древнеиндийская легенда, согласно которой в городе Бенаресе под куполом главного храма, в том месте, где находится центр Земли, на бронзовой площадке стоят три алмазных стержня. В день сотворения мира на один из этих стержней было надето 64 кольца. Бог поручил жрецам перенести кольца с одного стержня на другой, используя третий в качестве вспомогательного. Жрецы обязаны соблюдать условия:

1. Переносить за один раз только одно кольцо;

2. Кольцо можно переносить с одного стержня (x) на другой (y), только если оно имеет меньший диаметр, чем верхнее кольцо, находящееся на стрежне y, или стержень y свободен.

Согласно легенде, когда, соблюдая все условия, жрецы перенесут все 64 кольца, наступит конец света. Минимальное число ходов, необходимое для решения головоломки, равно $2^n - 1$, где n — число дисков. Если считать, что на перенос одного кольца потребуется одна секунда, то на перенос всех 64 колец потребуется более 5 млрд веков.

Если башня состоит из одного диска, то она переносится за один ход: 1->3.

Башня из двух дисков переносится за три хода: 1—>2, 1—>3, 2—>3.

Для переноса башни из трех дисков потребуется уже семь ходов: 1->3, 1->2, 3->2, 1->3, 2->1, 2->3, 1->3.

Чтобы перенести башню из четырех дисков с первого стержня на третий, необходимо действовать по плану:

1) перенести башню из трех верхних дисков с первого стержня на второй (7 ходов);

2) самый большой диск перенести с первого стержня на третий (1 ход);

3) перенести башню из трех дисков со второго стержня на третий (7 ходов).

Всего на перенос потребуется 15 ходов.

Рассуждая аналогичным образом, сосчитаем число ходов, необходимых для переноса башни из пяти дисков: 15 + 1 + 15 = 2 • 15 + 1 = 31.

Рассмотренный алгоритм решения задачи «Ханойская башня» обладает рекурсивным свойством: в ходе его выполнения для башни, состоящей из n колец, используется алгоритм для чуть более простой ситуации — переноса башни, состоящей из n - 1 кольца. В свою очередь, в алгоритме для башни из n - 1 кольца используется этот же алгоритм для n - 2 колец и т. Д.

Пример программы с рекурсивной функцией

```
//ханойские башни
Void hb(int n,int x,int y,int z)
{
If (n>0)
{hb (n-1, x, z, y);
Cout<<"\n<P> "<< x<< " -> "<<y;
Hb (n-1, z, y, x);
```

```
}
}
Main()
{         int st1, st2, st3, st4;
//Введите число колец st1
Cout<<"\nkol rkolez ";
Cin>>st1;
//Введите начальный стержень st2
Cout<<"\nnum beg ";
Cin>>st2;
//Введите конечный стержень st3
Cout<<"\nnum end ";
Cin>>st3;
//Введите вспомогательный стержень st4
Cout<<"\nnum wrem ";
Cin>>st4;
Hb(st1,st2,st3,st4);
}
```

Результаты работы программы

```
Kol rkolez 3
Num beg 1
Num end 3
Num wrem 2
<P> 1 -> 3
<P> 1 -> 2
<P> 3 -> 2
<P> 1 -> 3
<P> 2 -> 1
<P> 2 -> 3
<P> 1 -> 3
```

Задание на практическую работу

1. Напишите рекурсивную функцию, определяющую количество единиц в двоичном представлении натурального числа.
2. Напишите рекурсивную функцию, которая по заданным натуральным числам m и n выводит все различные представления числа «в виде суммы m натуральных слагаемых». Представления, различающиеся лишь порядком слагаемых, считаются одинаковыми.
3. Напишите рекурсивную функцию, которая переворачивает элементы массива в обратном порядке.

Практическая работа №13 «Создание библиотеки подпрограмм»

Цель работы: сформировать умения по использованию процедурного языка программирования для построения логически правильных и эффективных программ с использованием библиотеки подпрограмм

Существует два вида библиотек статические .lib и динамические .dll,

Открываем visual studio у меня visual studio 2013

Выбираем: “Создать новый проект”, в открывшемся окне выбираем “консольное приложение виндовс 32”, также вводим название проекта mynewlib и указываем путь где мы сохраним проект.

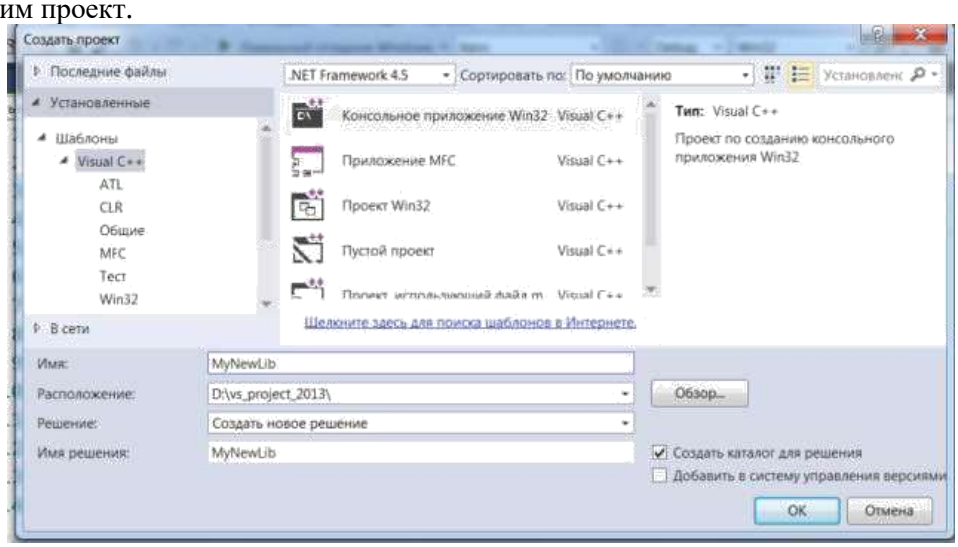


Рисунок 13.1. Создание проекта

Дальше нажимаем ок и мы переходим к следующему окошку.

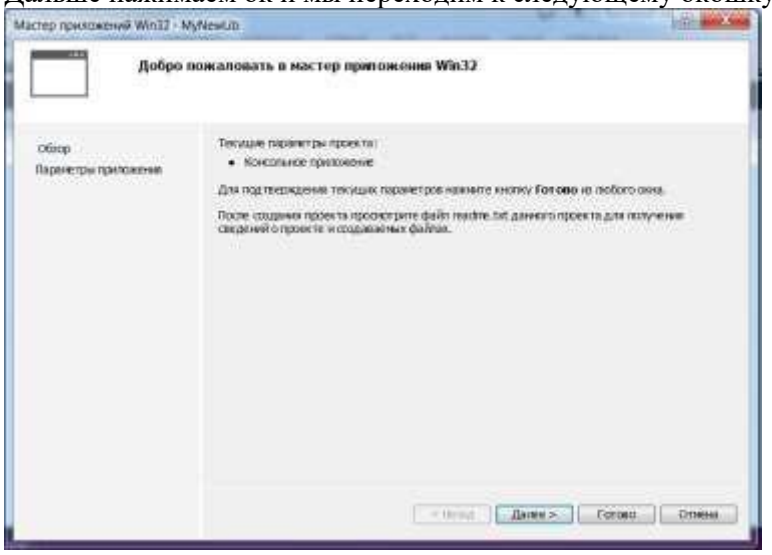


Рисунок 13.2. Создание проекта – этап обработки

В новом окне выбираем “Статическая библиотека” и убираем галочку “предварительно откомпилированные заголовок”, далее нажимаем «готово»

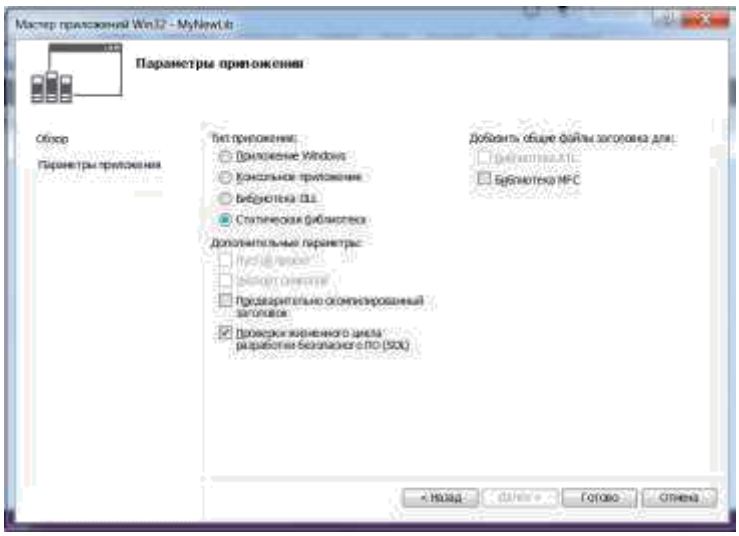


Рисунок 13.3. Выбор библиотеки

Таким образом, у нас создается новый проект в котором будут создаваться не программы exe а статические библиотеки lib.

Дальше в этот проект mynewlib добавляем два файла один заголовочный function.h:

```
#ifndef FUNCTION_H
#define FUNCTION_H
Void func_hellow();
Void func_privet();
#endif
```

Файл function.cpp:

```
#include <iostream>
Using std::cout;
Using std::endl;
#include "function.h";
Void func_hellow()
{
Cout <<"Hellow world gacpada"<<endl;
}
Void func_privet()
{
Cout <<"privet eb tu bl9 mazafaka bich"<<endl;
```

Включаем их в проект, а дальше компилируем нажимаем клавиши ctrl+alt+F7 и у нас создается в папке Debug или Release смотря какой режим выбрать. (мы выбраем режим Debug) и в папке Debug создался файл библиотеки mynewlib.lib

Для того, чтобы создавать качественные библиотеки нам нужно каждую функцию записать в отдельный файл, это для того, чтобы в mynewlib.lib было создано несколько объектных файлов .obj в данном случае мы могли бы файл function.cpp разбить на два файла допустим файл f1.cpp в котором записать определение функции func_hellow() и на файл f2.cpp в нем записать определение второй функции func_privet(), тогда у нас будет правильная библиотека и если мы в проекте будем использовать одну функцию, допустим func_hellow(), а func_privet() нет, то объектный код .obj с функцией func_privet() не добавиться , а только .obj func_hellow() будет добавляться, я так думаю это фажный момент, возможно ниже попробуем смоделировать два разных варианта создания библиотек. Просто при использовании библиотеки в которой каждой функции принадлежит отдельный объектный код, только включаются те функции которые используются, а если все функции в одном .obj то естественно включатся все функции и даже те которые мы не используем.

Нужно подключить библиотеку к нашему проекту, для этого:

- создаем новый проект пустой, проект будет пусть консольный вин 32 и создаем в нем файл

Main.cpp со следующим кодом:

```
#include <iostream>
Using std::cout;
Using std::endl;
```

```
#include "function.h"
Int main()
{
Func_hellow();
Func_privet();
Return 0;
}
```

Таким образом, мы включили в проект заголовочный файл function.h в котором находятся определения функций

Вызываем функции, (если мы сейчас это все скомпилируем, то у нас вылезет ошибка так как мы реально не добавили файл function.h и не добавили саму библиотеку в проект)

- Копируем файлы mynewlib.lib и файл function.h в папку с файлами нашего проекта – это та папка где находится наш файл main.cpp.
- С помощью горячих клавиш Shift+Alt+A

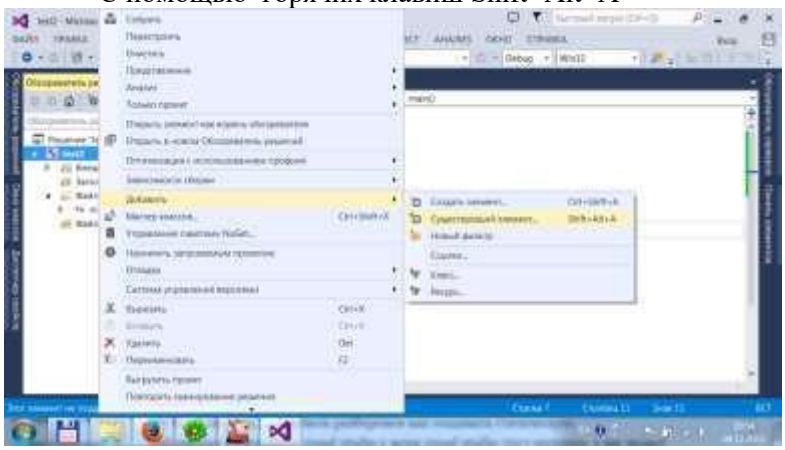


Рисунок 13.4. Вызов функии

Дальше появится окно.

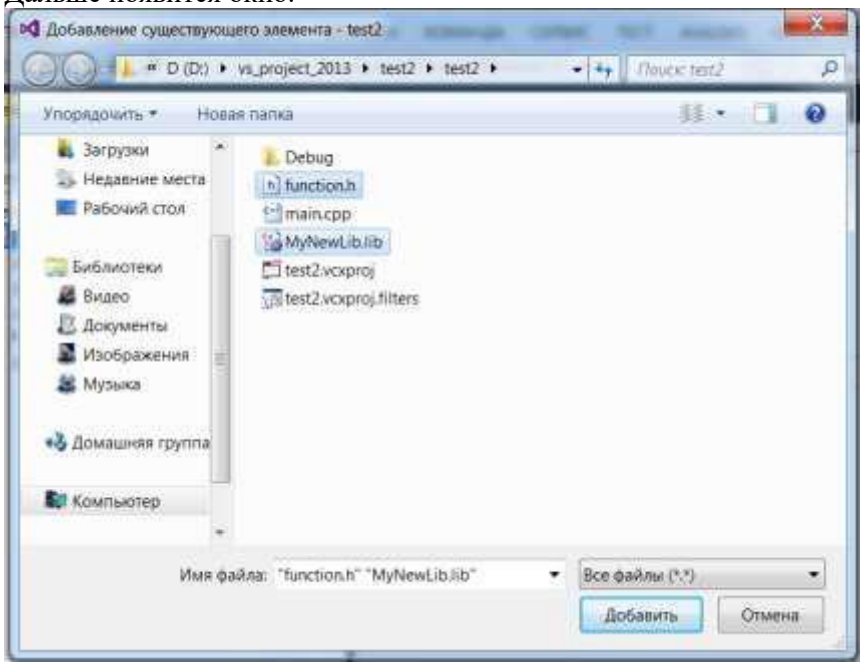


Рисунок 13.5. Выбор функции

Выбираете файлы и нажимаете добавить и все файлы добавлены, при открытии обозревателя решений у вас они должны быть видны из него.

Безымянный

Все из рисунка видно что статическая библиотека mynewlib.lib у нас добавлена и заголовочный файл с определениями функций из библиотеки mynewlib.lib включен это файл function.h, теперь нажимаем Ctrl+F5 и все компилируется.

Практическая работа №14 «Классы ООП: виды, назначение, свойства, методы, события. Объявления класса»

Цель работы: Закрепить понятие класса и объекта.

Понятие класса и объекта

Класс — это новый структурированный тип, включающий в себя в качестве элементов (членов класса):

- данные (свойства объекта) – переменные нужного типа;
- функции (методы объекта), применяемые по отношению к этим данным.

Класс используется для построения объектов. **Объекты** - это экземпляры класса. Синтаксис объявления класса подобен синтаксису объявления структуры, когда под одним именем объединялись несколько полей данных разного типа.

```
Class имя_класса
{
Закрытые свойства и методы
Public:
Открытые свойства и методы
};
```

Определив класс, можно создать объект этого "классового" типа, используя имя класса. Переменные типа класс называются объектами.

После создания объекта класса можно обращаться к открытым членам класса, используя составное имя, которое образуется из имени объекта и оператора точка (.), аналогично тому, как осуществляется доступ к членам структуры.

Объект. Член_класса

Методы класса

Метод - это функция, которая входит в состав окласса и выполняет действия над свойствами объекта. Обычно определение методов класса выносится перед функцией main().

Для определения метода надо указать, к какому классу он относится. Это связано с тем, что в языке C++ разные классы могут иметь методы с одинаковыми именами. Для этого используется операция :: - операция принадлежности классу или оператором разрешения области видимости.

Имя_класс::имя_функции называется полным или квалификационным именем функции – члена класса.

При определении функции-члена пользуйтесь следующей основной формой:

```
Тип_результата  имя __класса :: имя__функции (список__параметров)
{
        // тело функции
}
```

Здесь имя_класса — это имя того класса, которому принадлежит определяемый метод.

Доступ к свойствам объекта (для чтения их значений или для записи в них значения) можно осуществлять только через методы. Обращаться к свойствам объекта без помощи метода может вызвать трудно обнаруживаемые ошибки, ведь у каждого объекта свои значения свойств.

Метод – это функция, но только входящая в состав класса. Поэтому при вызове метода, как и при вызове функции, происходит скачок к телу вызванной функции и при необходимости передачу ей параметров и получение от нее результатов ее работы. Однако, имеются и отличия методов от просто функций:

1. Чтобы вызвать метод в той части программы, которая не является частью класса, надо использовать составное имя из объекта и операции точка для вызова метода. Для обычной функции составное имя не используется.

2. Чтобы из одного метода вызвать другой метод этого же класса операция точка не используется. В этом случае компилятор уже точно знает, какой объект подвергается обработке.

3. Благодаря составному имени при вызове метода, нет необходимости указывать через точку обращение к свойствам объекта, как это надо делать для структур. Метод будет работать со свойствами того объекта, который в данный момент вызвал метод.

Пример программы с классом и объектом

```
//Создать класса – круг со свойством:
//r - радиус окружности
//Методы:
```

```
//  1. Инициализация окружности
//  2. Вычисление площади
Class krug                              //Класс
{
Float r;                    //Скрытые свойства класса
Public:                     //Открытая часть класса
Void init (float rad);
Float plosh();
};
Main()
{       krug k;
Float r, s;
Cout<<”Введите радиус круга”;
Cin>>r;
K.init (r);
S=k.plosh();
Cout<<”Площадь круга=”<<s<<endl;
}
```

Задание на практическую работу

Написать программу с использование объектов, выполнив следующую последовательность действий:

1. Создать новый тип - класс, включив в его состав необходимые свойства и методы. Описать класс – овал со свойствами:
   a. Радиусы по горизонтали и вертикали в пикселях;
   b. Методы:
   c. Инициализация свойств объекта;
   d. Изменить значения радиусов овала;
   e. Расчитать плщадь овала
   f. Расчмтать длину дуги овала.
   g. Реализовать методы созданного класса.
   h. Описать объекты созданного класса.
2. Создать один экземпляр класса – одиночный объект, и к нему применить запрограммированные методы.
3. Создать несколько объектов созданного класса, и к каждому объекту применить методы.
4. Сформировать массив из 5 объектов. Применить к каждому элементу такого массива методы.

Практическая работа №15 «Создание наследованного класса»

Цель работы: Ввести понятие наследования и создания на этом понятии родственных классов

Понятие о наследовании

Наследование — один из трех фундаментальных принципов объектно-ориентированного программирования, поскольку именно благодаря ему возможно создание иерархических классификаций. Используя наследование, можно создать общий класс, который определяет характеристики, присущие множеству связанных элементов. Этот класс затем может быть унаследован другими, узкоспециализированными классами с добавлением в каждый из них своих, уникальных особенностей.

В стандартной терминологии языка C++ класс, который наследуется, называется **базовым**. Класс, который наследует базовый класс, называется **производным**. Производный класс можно использовать в качестве базового для другого производного класса. Таким путем и строится многоуровневая иерархия классов.

Основное достоинство наследования состоит в том, что оно позволяет создать базовый класс, который затем можно включить в состав более специализированных классов. Таким образом, каждый производный класс может служить определенной цели и при этом оставаться частью общей классификации.

Пример с наследованием

Рассмотрим класс auto, который в самых общих чертах определяет дорожное транспортное средство. Его члены данных позволяют хранить количество колес и число пассажиров, которое может перевозить транспортное средство.

```
Class automob
{
Int kol;
Int pass;
Public:
Void set_kol(int num)   { kol = num; }
Int get_kol()                    { return kol; }
Void set_pass(int num) { pass = num; }
Int get_pass()              { return pass; }
};
```

Это общее определение дорожного транспортного средства можно использовать для определения конкретных типов транспортных средств. Например, в следующем фрагменте путем наследования класса auto создается класс grus (грузовых автомобилей).

```
Class grus : public automob
{
Int cargo;
Public:
Void set_cargo(int size){ cargo = size; }
Int get_cargo()  { return cargo; }
Void show();
};
```

Тот факт, что класс grus наследует класс auto, означает, что класс grus наследует все содержимое класса auto. К содержимому класса auto класс grus добавляет член данных cargo, а также функции-члены, необходимые для поддержки члена cargo.

Общий формат для обеспечения наследования имеет следующий вид.

```
Class имя_производного_класса: доступ имя__базового_класса
{
Тело нового класса
};
```

Здесь элемент **доступ** необязателен. При необходимости он может быть выражен одним из спецификаторов доступа: public, private или protected. Чаще всего используется спецификатор public. Это означает, что все public-члены базового класса также будут public-членами производного класса. Следовательно, в предыдущем примере члены класса grus имеют доступ к открытым функциям-членам класса auto, как будто они (эти функции) были объявлены в теле

класса grus. Однако класс grus не имеет доступа к private-членам класса auto. Например, для класса grus закрыт доступ к члену данных kol.

Пример программы с наследованием классов

Рассмотрим программу, которая использует механизм наследования для создания двух подклассов класса auto: grus и legk.

```
// Определяем базовый класс транспортных средств,
Enum type {car, van, wagon};
Class automob
{int kol;
Int pass;
Public:
Void set_kol(int num)   { kol = num; }
Int get_kol()           { return kol; }
Void set_pass(int num) { pass = num; }
Int get_pass()    { return pass; }
};
// Определяем класс грузовиков,
Class grus : public automob
{
Int cargo;
Public:
Void set_cargo(int size){ cargo = size; }
Int get_cargo()  { return cargo; }
Void show();
};
// Определяем класс автомобилей.
Class legk : public automob
{
Enum type car_type;
Public:
Void set_type(type t)     { car_type = t; }
Enum type get_type()    { return car_type; }
Void show();
};
Void grus::show()
{
Cout <<"Koles: "<<get_kol()<<"\n";
Cout<<"Passajir: "<<get_pass()<<"\n";
Cout<<"Grusopodjemnost: "<<cargo<<"\n";
}
Void legk::show ()
{
Cout<<"Koles: "<<get_kol()<<"\n";
Cout<<"Passajir: "<<get_pass()<< "\n";
Cout<<"Tip kusova: ";
Switch(get_type())
{
Case van: cout <<"legkovoi\n"; break;
Case car: cout <<"sedan\n"; break;
Case wagon: cout << "kabriolet\n";
}
}
Main()
{          grus t1, t2;
Legk c;
T1.set_kol(18); t1.set_pass(2);   t1.set_cargo(3200);
T2.set_kol(6);   t2.set_pass(3);   t2.set_cargo(1200);
```

```
T1.show();  cout<<"\n";  t2.show();  cout<<"\n";
C.set_kol(4);  c.set_pass(6);  c.set_type(van);
C.show();
}
```

При выполнении эта программа генерирует такие результаты.

```
Колес: 18
Пассажиров: 2
Грузовместимость в кубических футах: 3200
Колес: 6
Пассажиров: 3
Грузовместимость в кубических футах: 1200
Колес: 4 пассажиров: 6 тип: автофургон
```

Оба класса grus и legk включают функцию-член show (), которая отображает информацию об объекте. Эта функция демонстрирует еще один аспект объектно-ориентированного программирования — полиморфизм. Поскольку каждая функция show() связана с собственным классом, компилятор может легко "понять", какую именно функцию нужно вызвать для данного объекта.

Задание на практическую работу

1. На основе базового класса «Автомобиль», создать производный класс «Общественный транспорт».
2. На основе базового класса «Общественный транспорт», создать производный класс «Автобус».
3. На основе базового класса «Общественный транспорт», создать производный класс «Трамвай».

Практическая работа №16 «Перегрузка методов»

Цель работы: Закрепить понятие класса, объекта, ввести понятие конструктора, деструктора.

Назначение конструкторов

Как правило, некоторую часть объекта, прежде чем его можно будет использовать, необходимо инициализировать. Например, рассмотрим класс queue. Прежде чем класс queue можно будет использовать, переменным rloc и sloc нужно присвоить нулевые значения. В C++ предусмотрена реализация этой возможности при создании объектов класса. Такая автоматическая инициализация выполняется благодаря использованию конструктора.

Конструктор — это специальная функция, которая является членом класса, и имя которой совпадает с именем класса.

```
Class queue                // Определение класса queue.
{int q[100];
Int sloe, rloc;
Public:
Queue (); // конструктор
Void qput(int i);
Int qget();
};
```

В объявлении конструктора queue() отсутствует тип возвращаемого значения. В C++ конструкторы не возвращают значений, при этом нельзя указывать даже тип void. Теперь приведем код функции queue().

```
// Определение конструктора.
Queue::queue()
{sloe = rloc = 0;
Cout << "Очередь инициализирована.\n";
}
```

В данном случае при выполнении конструктора выводится сообщение «Очередь инициализирована», которое служит исключительно иллюстративным целям. На практике же в большинстве случаев конструкторы не выводят никаких сообщений.

Конструктор объекта вызывается при создании объекта. Это означает, что он вызывается при выполнении инструкции объявления объекта. Конструкторы глобальных объектов вызываются в самом начале выполнения программы, еще до обращения к функции main(). Что касается локальных объектов, то их конструкторы вызываются каждый раз, когда встречается объявление такого объекта.

Деструкторы

Деструктор — это функция, которая вызывается при разрушении объекта.

Во многих случаях при разрушении объекту необходимо выполнить некоторое действие или даже некоторую последовательность действий. Локальные объекты создаются при входе в блок, в котором они определены, и разрушаются при выходе из него. Глобальные объекты разрушаются при завершении программы. Существует множество факторов, обусловливающих необходимость деструктора. Например, объект должен освободить ранее выделенную для него память. В C++ именно деструктору поручается обработка процесса дезактивизации объекта. Имя деструктора совпадает с именем конструктора, но предваряется символом "~" (тильда). Подобно конструкторам деструкторы не возвращают значений, а следовательно, в их объявлениях отсутствует тип возвращаемого значения.

Рассмотрим уже знакомый нам класс queue, но теперь он содержит конструктор и деструктор. (Справедливости ради отметим, что классу queue деструктор, по сути, не нужен, а его наличие здесь можно оправдать лишь иллюстративными целями).

```
// Определение класса queue,
Class queue
{
Int q[100];
Int sloc, rloc;
Public:
Queue();  // конструктор
~queue(); // деструктор
```

```
Void qput(int i);
Int qget();
};
// Определение деструктора
Queue::~queue()
{       cout << "Очередь разрушена.\n"; }
```

Параметризованные конструкторы

Конструктор может иметь параметры. С их помощью при создании объекта членам данных (переменным класса) можно присвоить некоторые начальные значения, определяемые в программе. Это реализуется путем передачи аргументов конструктору объекта.

Чтобы передать аргумент конструктору, необходимо связать этот аргумент с объектом при объявлении объекта. C++ поддерживает два способа реализации такого связывания. Вот как выглядит первый способ.

```
Queue  a = queue(101);
```

В этом объявлении создается очередь с именем a, которой передается значение (идентификационный номер) 101. Но эта форма (в таком контексте) используется редко, поскольку второй способ имеет более короткую запись и удобнее для использования. Во втором способе аргумент должен следовать за именем объекта и заключаться в круглые скобки. Например, следующая инструкция эквивалентна предыдущему объявлению.

```
Queue a(101);
```

Это самый распространенный способ объявления параметризованных объектов. Опираясь на этот метод, приведем общий формат передачи аргументов конструкторам.

Тип_класса имя_переменной(список_аргументов);

Здесь элемент список_аргументов представляет собой список разделенных запятыми аргументов, передаваемых конструктору.

Создание в программе перегружаемых конструкторов

Наиболее частое использование перегрузки конструктора — это обеспечение возможности выбора способа инициализации объекта. Например, в следующей программе объекту ol дается начальное значение, а объекту o2 — нет. Если вы удалите конструктор с пустым списком аргументов, программа не будет компилироваться, поскольку у неинициализируемого объекта типа samp не будет конструктора. И наоборот, если вы удалите конструктор с параметром, программа не будет компилироваться, поскольку не будет конструктора у инициализируемого объекта типа samp. Для правильной компиляции программы необходимы оба конструктора.

```
Class myclass
{
Int x;
Public:
// перегрузка конструктора двумя способами
Myclass()       { x = 0; }                  // нет инициализации
Myclass(int n)  { x = n; }          // инициализация
Int getx() { return x; }
};
Main()
{       myclass o1(10);             // объявление с начальным значением
Myclass o2;                         // объявление без начального значения
Cout<<"o1:  "<<o1.getx()<<'\n';
Cout<<"o2:  "<<o2.getx()<<'\n';
}
```

Перегрузка конструктора позволяет программисту выбрать наиболее подходящий метод инициализации объекта. Рассмотрим пример, в котором создается класс для хранения календарной даты. Конструктор date() перегружается двумя способами. В первом случае данные задаются в виде строки символов, в другом — в виде трех целых.

```
Class date
{
Int day, month, year;
Public:
Date(char *str) ;
```

```
Date (int m, int d, int y) { day = d; month = m; year = y; }
Void show()                {cout<<month<<'/'<<day<<'/'<<year<<"\n"; }
};
Date::date(char *str)
{         sscanf(str, "%d%*c%d%*c%d",  &month, &day, &year); }
Main()
{// использование конструктора для даты в виде строки
Date sdate("11/1/92");
// использование конструктора для даты в виде трех целых
Date idate(11, 1, 92);
Sdate.show();
Idate.show();
}
```

Преимущество перегрузки конструктора date(), как показано в программе, в том, что вы можете выбрать ту версию инициализации, которая лучше всего подходит к текущей ситуации. Например, если объект типа date создается в результате пользовательского ввода, то проще использовать строковую версию. Однако если объект типа date строится путем каких-то простых внутренних расчетов, то версия с тремя целыми параметрами становится, вероятно, более привлекательной.

Пример программы с конструкторами и деструкторами

Программа реализации очереди, в которой демонстрируется использование конструктора и деструктора.

```
// Определение класса queue,
Class queue
{
Int q[100];
Int sloc, rloc;
Public:
Queue();                    // непараметризованный конструктор
Queue(int id);   // параметризованный конструктор
~queue(); // деструктор
Void qput(int i);
Int qget() ;
};
// Определение конструктора без параметров
Queue::queue ()
{ sloc = rloc = 0; cout << "Очередь инициализирована.\n";}
//Определение конструктора с параметрами
Queue :: queue (int id)
{
Sloc = rloc = 0;
Who = id;
Cout << "Очередь " << who << " инициализирована.\n";
}
// Определение деструктора.
Queue::~queue()
{ cout <<"Очередь разрушена.\n"; }
// Занесение в очередь целочисленного значения.
Void queue::qput(int i)
{
If(sloc==100)   { cout << "Очередь заполнена.\n";       return; }
Sloc++;
Q[sloc] = i;
}
// Извлечение из очереди целочисленного значения,
Int queue::qget()
{
```

```
If(rloc == sloc) {cout << "Очередь пуста.\n";          return 0; }
Rloc++;
Return q[rloc];
}
Main ()
{       queue a, b;              // Создание объектов конструктором без параметров
A.qput(10);          b.qput(19);          a.qput(20);          b.qput(1);
Cout << a.qget()<<" ";          cout << a.qget()<<" ";
Cout << b.qget()<<" ";          cout << b.qget()<<"\n";
Queue  a(1), b(2);       // Создание объектов конструктором с параметрами
A.qput (10);     b.qput (19);
A.qput(20);          b.qput(1);
Cout << a.qget()<<" ";          cout<< a.qget()<<" ";
Cout << b.qget()<<" ";          cout << b.qget()<<"\n";
}
```

При выполнении этой программы получаются такие результаты.

Очередь инициализирована.
Очередь инициализирована. 10 20 19 1
Очередь разрушена. Очередь разрушена.
Очередь 1 инициализирована.
Очередь 2 инициализирована.
10 20
19 1
Очередь 2 разрушена.
Очередь 1 разрушена.

В примере с использованием класса queue при создании объекта передается только один аргумент. В общем случае возможна передача двух аргументов и более. В отличие от конструкторов, деструкторы не могут иметь параметров. Причину понять нетрудно: не существует средств передачи аргументов объекту, который разрушается.

Задание на практическую работу

1. Разработайте программу, содержащую класс с двумя открытыми и двумя закрытыми целочисленными переменными. Создайте два объекта этого класса. Используйте для инициализации закрытых переменных класса параметризованные конструкторы. Дополните программу функциями расчета суммы всех переменных объекта.
2. Создайте функции, определяющие максимальное и минимальное значение переменных каждого объекта. Используйте коструктор.
3. Определите, в каком объекте сумма переменных максимальна (минимальна). Используйте перегрузку конструкторов.

Практическая работа №17 «Использование указателей для организации связанных списков»

Цель работы: изучить основные понятия, относящиеся к классам и объектам, освоить динамическое создание объектов в программном коде.

Классы и объекты

Объектно-ориентированном подходе существуют понятия класс и Объект.

Класс – это программная единица, которая задает общий шаблон для конкретных объектов. Класс содержит все необходимые описания переменных, свойств и методов, которые относятся к объекту. Примером класса в реальной жизни является понятие «автомобиль»: как правило, автомобиль содержит некоторое количество колес, две-рей, имеет какой-то цвет, но эти конкретные детали в классе не описываются.

Объект – это экземпляр класса. Свойства объекта содержат конкретные данные, характерные для данного экземпляра. В реальной жизни примером объекта будет конкретный экземпляр автомобиля с 4 колесами, 5 дверками и синего цвета.

Динамическое создание объектов

Чаще всего для размещения на форме кнопки, поля ввода или других управляющих элементов используется дизайнер среды Visual Studio: нужный элемент выделяется в панели элементов и размещается на форме. Однако иногда создавать элементы нужно уже в процессе выполнения программы. Поскольку каждый элемент управления представляет собой отдельный класс, его помещение на форму программным способом включает несколько шагов:

11. Создание экземпляра класса.
12. Привязка его к форме.
13. Настройка местоположения, размеров, текста и т. П.

Например, чтобы создать кнопку, нужно выполнить следующий код (его следует разместить в обработчике сообщения Load или в каком-либо другом методе):

```
Button b = new Button();
```

Здесь объявляется переменная b, относящаяся к классу Button, как и в предыдущих лабораторных работах. Однако дальше идет нечто но-вое: с помощью оператора new создается экземпляр класса Button, ссылка на него присваивается переменной b. При этом выполняется целый ряд дополнительных действий: выделяется память под объект, инициализируются все свойства и переменные.

Далее нужно добавить объект на форму. Для этого служит свойство Parent, которое определяет родительский элемент, на котором будет размещена кнопка: B.Parent = this;

Ключевое слово this относится к тому объекту, в котором разме-щен выполняемый в данный момент метод. Поскольку все методы в лабораторных работах размещаются в классе формы, то и this относится к этому конкретному экземпляру формы.

Вместо формы кнопку можно поместить на другой контейнер. На-пример, если на форме есть элемент управления Panel, то можно поместить кнопку на него следующим образом:

```
B.Parent = panel1;
```

Чтобы задать положение и размеры кнопки, нужно использовать свойства Location и Size:

```
B.Location = new Point(10, 20);
B.Size = new Size(200, 100);
```

Обратите внимание, что Location и Size – это тоже объекты. Хотя внутри у Location содержатся координаты x и y, задающие левый верхний угол объекта, не получится поменять одну из координат, нужно менять целиком весь объект Location. То же самое относится и к свойству Size.

На самом деле, каждый раз, когда на форму помещается новый элемент управления или вносятся какие-то изменения в свойства элементов управления, Visual Studio генерирует специальный служебный код, который проделывает приведенные выше операции по созданию и настройке элементов управления. Попробуйте поместить на форму кнопку, изменить с нее какие-нибудь свойства, а затем найдите в обозревателе решений ветку формы Form1, разверните ее и сделайте двойной щелчок по ветке Form1.Designer.cs. Откроется файл с текстом программы на языке C#, которую среда создала автоматически. Менять этот код вручную крайне не рекомендуется! Однако можно его изучить, чтобы понять принципы создания элементов управления в ходе выполнения программы.

Область видимости

Переменные, объявленные в программе, имеют область видимости. Это значит, что переменная, описанная в одной части программы, не обязательно будет видна в другой. Вот наиболее часто встречающиеся ситуации:

Переменные, описанные внутри метода, не будут видны за пределами этого метода. Например:

```
Void methoda()
{
Описываем переменную delta int delta = 7;
}
Void methodb()
{
Ошибка: переменная delta в этом методе неизвестна! Int gamma = delta + 1;
}
```

Переменные, описанные внутри блока или составного операто-ра, видны только внутри этого блока. Например:

```
Void Method()
{
If (a == 7)
{
Int b = a + 5;
}
//              Ошибка: переменная b здесь уже неизвестна! Messagebox.Show(b.tostring());
}
```

Переменные, описанные внутри класса, являются глобальными и доступны для всех методов этого класса, например:

```
Class Form1 : Form
{
Int a = 5;
Void Method()
{
//                      Переменная a здесь действительна
Messagebox.Show(a.tostring());
}
}
```

Операции is и as

Часто бывает удобно переменные разных классов записать в один список, чтобы было легче его обрабатывать. Чтобы проверить, к какому классу принадлежит какой-либо объект, можно использовать оператор is: он воз-вращает истину, если объект принадлежит указанному классу. Пример:

```
Button b = new Button();
If (b is Button)
Messagebox.Show("Это кнопка!"); else
Messagebox.Show("Это что-то другое...");
```

Как правило, в общих списках объекты хранятся в «обезличенном» состоянии, так, чтобы у всех у них был лишь минимальный общий для всех набор методов и свойств. Для того чтобы получить доступ к расширенным свойствам объекта, нужно привести его к исходному классу с помощью операции приведения as: (someobject as Button).Text = "Это кнопка!";

Следует помнить, что операция приведения сработает только в том случае, если объект изначально принадлежит тому классу, к которому его пытаются привести (или совместим с ним), в противном случае оператор as выбросит исключение и остановит выполнение программы. Поэтому более безопасный подход состоит в комбинированном применении операторов as и is: сначала проверяем совместимость объекта и класса, и только потом выполняем операцию приведения:

```
If (someobject is Button)
(someobject as Button).Text = "Это кнопка!";
```

В качестве практического примера использования этих операций рассмотрим пример программы, которая перебирает все элементы управления на форме, и у кнопок (но не у других элементов управления!) Заменяет текст на пять звездочек «*****»:

```
Private void Form1_Load(object sender, eventargs e)
{
Перебираем все элементы управления foreach (Control c in this.Controls)
If (c is Button) // Кнопка?
(c as Button).Text = "*****"; // Да!
}
```

Сведения, передаваемые в событие

Когда происходит какое-либо событие (например, событие Click при нажатии на кнопку), в обработчик этого события передаются дополнительные сведения об этом событии в параметре e.

Например, при щелчке кнопки мыши на объекте возникает событие mouseclick. Для этого события параметр e содержит целый ряд переменных, которые позволяют узнать информацию о нажатии:

- Button – какая кнопка была нажата;
- Clicks – сколько раз была нажата и отпущена кнопка мыши;
- Location – координаты точки, на которую указывал курсор в момент нажатия, в виде объекта класса Point;
- X и Y – те же координаты в виде отдельных переменных.

Задание на практическую работу

Если в индивидуальном задании используется элемент Panel, измените его цвет, чтобы он визуально выделялся на форме. Если используется элемент Label, не забудьте присвоить ему какой-либо текст, иначе он не будет виден на форме.

Индивидуальные задания

1. Разработать программу, динамически порождающую на окне кнопки. Левый верхний угол кнопки определяется местоположением курсора при щелчке. Вывести надпись на кнопке с координатами ее левого верхнего угла.
2. Разработать программу, динамически порождающую на окне кноп-ки и поля ввода. Левый верхний угол элемента управления определяется местоположением курсора при щелчке. Кнопка порождается, если курсор находится в левой половине окна, в ином случае порождается поле ввода.
3. На форме размещен элемент управления Panel. Написать про-грамму, которая при щелчке мыши на элементе управления Panel добавляет в него кнопки Button, а при щелчке на форме в нее добавляются поля ввода textbox.
4. На форме размещены 3 панели (элемент управления Panel). Написать программу, которая при щелчке мыши на первой панели добавляет во вторую панель кнопки Button, при щелчке на второй панели добавляет в третью панель поля ввода textbox, а при щелчке на третьей панели добавляет на первую панель метки Label.

и Написать программу, добавляющую на форму кнопки. Кнопки добавляются в узлы прямоугольной сетки. Расстояния между кнопками и расстояния между крайней кнопкой и границей окна должны быть равны как по горизонтали, так и по вертикали.

6. Разработать программу, при щелчке мыши динамически порождающую на окне кнопки или поля ввода. Каждый четный элемент управления является кнопкой, нечетный – полем ввода. Левый верхний угол кнопки определяется местоположением курсора при щелчке. Для поля ввода положение курсора определяет координаты правого нижнего угла.
7. Создать программу с кнопкой, меткой и полем ввода. При щелчке на соответствующий элемент на форме динамически должен создаваться подобный ему элемент. Предусмотреть возможность вывода количества кнопок, меток и полей ввода.
8. Создать программу, добавляющую различные элементы управления на форму и на панель Panel. Тип элементов управления выбирается случайным образом. Предусмотреть возможность вывода информации о количестве элементов по типам и информацию о расположении элементов.
9. Разработать программу, добавляющую на форму последовательность элементов управления случайной длины. Тип элементов управления задается случайным образом. Предусмотреть возможность вывода информации о количестве элементов по типам.

10. Написать программу, динамически порождающую на окне кнопки или метки. Левый верхний угол элемента управления определяется местоположением курсора при щелчке. При нажатии правой кнопки мыши на форме с нее удаляются все кнопки.

11. Написать программу, динамически порождающую на окне поочередно кнопки или поля ввода. Левый верхний угол элемента управления определяется местоположением курсора при щелчке. При нажатии правой кнопки мыши на форме с нее удаляются все порожденные элементы.

12. Разработать программу с двумя кнопками на форме. При нажатии на первую на форму добавляется одна панель Panel. При нажатии на вторую кнопку в каждую панель добавляется поле ввода.

13. Разработать программу с двумя кнопками на форме. При нажатии на первую на форму добавляется одна кнопка или поле ввода. При нажатии на вторую кнопку каждое поле увеличивается по вертикали в два раза.

14. Написать программу с кнопкой и тремя полями ввода. При нажатии на кнопку программа анализирует содержимое первого поля и динамически порождает элемент управления. Если в первом поле ввода содержится буква «К», то на форму добавляется кнопка, если «П» – поле ввода, если «М» – метка. Во втором и третьем поле ввода содержатся координаты левого верхнего угла будущего элемента управления.

15. Разработать программу, добавляющую на форму метки с текстом. Местоположение и размеры меток определяются в программе динамически через поля ввода. В заголовок окна, анализируя размер всех меток, вывести количество маленьких и больших меток. Маленькой меткой считается метка размером менее 50 пикселей по горизонтали и вертикали.

16. Создать программу с двумя кнопками на форме, динамически порождающую на окне метки или поля ввода. При нажатии на первую кнопку каждая метка увеличивается по горизонтали в два раза. При нажатии на вторую кнопку каждое поле уменьшается по вертикали в два раза.

17. Разработать программу, динамически порождающую на окне кнопки и поля ввода. Координаты элемента управления определяются случайным образом. Элементы управления не должны накладываться друг на друга. Если нет возможности добавить элемент управления (нет места для размещения элемента), то предусмотреть вывод информации об этом.

18. Разработать программу, динамически порождающую на окне кнопки и поля ввода. Координаты элемента управления определяются случайным образом. При наведении курсора на элемент управления он должен быть удален с формы.

19. Разработать программу, динамически порождающую при щелчке на окне различные элементы (поля ввода, кнопки, метки). Тип элементов определяется с помощью радиокнопок. Все элементы располагаются горизонтально в ряд. При достижении правой границы окна начинается новый ряд элементов.

20. Разработать программу, динамически порождающую или поле ввода (при нажатии на окне левой кнопкой мыши), или кнопку (при нажатии на окне правой кнопкой мыши). Все элементы располагаются наискосок, начиная с левого верхнего угла окна. Реализовать обработчик события изменения размера окна, в котором удалить все порожденные элементы.

Практическая работа №18 «Изучение интегрированной среды разработчика»

Цель работы: Получение навыков работы в визуальной среде программирования. Выполнения основных этапов разработки приложений.

Интегрированная среда разработчика C++ Builder

Среда C++ Builder визуально реализуется в виде нескольких окон, одновременно раскрытых на экране монитора. Среда включает все необходимое для проектирования, запуска и тестирования приложений. Вид среды может меняться в зависимости от настройки и номера версии. Среда C++ Builder визуально реализуется в виде нескольких окон, одновременно раскрытых на экране монитора. Количество, расположение, размер и вид окон может меняться программистом в зависимости от его предпочтений. При запуске C++ Builder можно увидеть картинку, как на Рисунке 8.1.

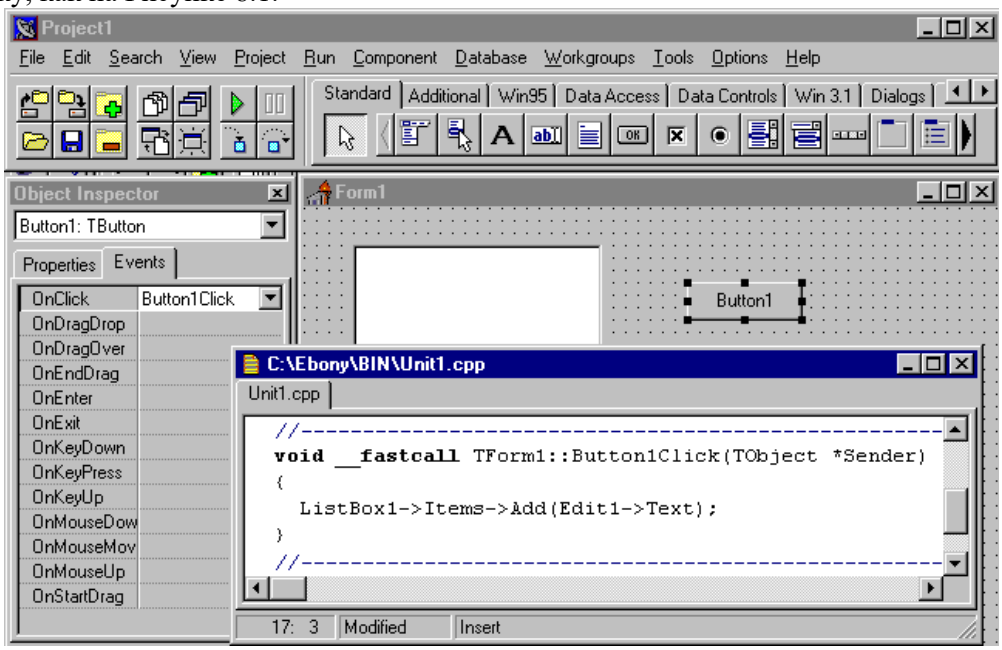


Рисунок 18.1. Среда Builder с открытыми окнами

Главное окно всегда присутствует на экране и предназначено для управления процессом создания программы. Основное меню содержит все необходимые средства для управления проектом. Пиктограммы облегчают доступ к наиболее часто применяемым командам основного меню. Через меню компонентов осуществляется доступ к набору стандартных сервисных программ среды C++ Builder, которые описывают некоторый визуальный элемент (компонент), помещенный программистом в окно формы. Каждый компонент имеет определенный набор свойств (параметров), которые программист может задавать, например, цвет, заголовок окна, надпись на кнопке, размер и тип шрифта и другие.

Окно инспектора объектов (вызывается с помощью клавиши F11) предназначено для изменения свойств выбранных компонентов и состоит из двух страниц. Страница Properties (Свойства) предназначена для изменения необходимых свойств компонента, страница Events (События) - для определения реакции компонента на то или иное событие (например, нажатие определенной клавиши или щелчок по кнопке мыши).

Окно формы представляет собой проект Windows-окна программы. В это окно в процессе написания программы помещаются необходимые компоненты. Причем при выполнении программы помещенные компоненты будут иметь тот же вид, что и на этапе проектирования.

Окно текста программы предназначено для просмотра, написания и редактирования текста программы. В системе C++ Builder используется язык программирования C++. При первоначальной загрузке в окне текста программы находится текст, содержащий минимальный набор операторов для нормального функционирования пустой формы в качестве Windows-окна.

Программа в среде C++ Builder составляется как описание алгоритмов, которые необходимо выполнить, если возникает определенное событие, связанное с формой (например, щелчок по кнопке мыши - событие onclick, создание формы - oncreate). Для каждого обрабатываемого в форме события с помощью страницы Events инспектора объектов в тексте программы организуется функция, между символами { и }, в которой программист записывает на языке C++ требуемый алгоритм.

Структура программ C++ Builder

Программа в C++ Builder состоит из множества модулей, которые объединяются в один проект с помощью файла проекта (файл с расширением .bpr). Файл проекта автоматически создается и обрабатывается средой C++ Builder и не предназначен для редактирования. Объявления классов, функций и переменных находятся в заголовочном файле (расширение .h), текст программы, написанный на языке C++, - в файле исходного текста (расширение .cpp). Описание окна формы находится в файле с расширением .dfm. Файл проекта может быть только один, файлов с другими расширениями может быть несколько.

Простейшее приложение C++ Builder

Простейшее приложение - заготовка, обеспечивающая программиста самими необходимыми действиями с окном. Заготовка есть завершенное рабочее приложение, которое пока ничего не делает. Сразу после запуска Builder создается простейшее приложение, которое представляет собой окно с основными элементами окна Windows: заголовок; кнопки сворачивания, разворачивания, закрытия окна; изменение размера окна.

Этапы создания простейшего приложения

1. Создать отдельную папку для файлов приложения.
2. Запустить среду C++ Builder.
3. Выполнить команду: File/New/Application.
4. Автоматически создаются файлы проекта Project1.cpp и форма с заголовком Form1.
5. Сохранение всех файлов проекта командой: File/Project As. Откроется диалоговое окно, в котором в качестве имени файла модуля формы предлагается имя Unit1.cpp. При этом в окне диалога надо выбрать диск и каталог в нем, можно изменить имя модуля, например, Main.cpp. При этом имя файла формы совпадет с именем модуля (но с расширением .dfm) и будет Main.dfm. После запоминания файла модуля формы Builder предлагает запомнить файл проекта с именем по умолчанию Project1.dpr. Это имя также можно изменить на другое, связанное с назначение проекта. Имена файла проекта и формы не должны совпадать.
6. Запуск проекта выполняется командой Run/Run или зеленой стрелкой на Панели инструментов. Компилятор переводит все тексты в машинный код (расширение .obj), а компоновщик объединяет все файлы в единый модуль (расширение .exe), который может быть запущен на выполнение.

Задание на практическую работу

Необходимо освоить основные этапы и действия со средой на примере создания простейшего приложения. Для этого надо создать отдельную папку, в которую будут помещаться все файлы, относящиеся к одному приложения.

1. Создать простейшее приложение C++ Builder в отдельной папке. Для файлов приложения подобрать подходящие имена. Просмотреть размер получившего приложения командой: Project/Information for Project. Примерный размер простейшего приложения – 25 Кбайт.
2. Просмотреть содержимое и изучить структуру следующих файлов:
   - Файла модуля: Unit1.cpp; при необходимости переключиться на текст клавишей F12;
   - Файла проекта Project1.cpp командой: Project/View Source; откроется отдельная закладка в окне Редактора;
   - Файла формы Unit1.dmf командой локального меню: View as Text; вернуться назад в представление формы как окна можно командой локального меню: View as Form.

Временно выйти из среды и просмотреть содержимое папки. Изучить состав образованных файлов приложения и их назначение.

3. Задание иконки разработанного приложения. Для этого требуется выполнить два этапа:

1. Создание иконки (файла с расширением .ico). Если файл с таким расширением есть, то этот этап можно опустить. Для создания иконки можно воспользоваться встроенным в среду Графическим редактором (вызвать графический редактор командой: Tools/Image Iditor; выбрать в нем пункт меню File/New/Icon File (ico); задать свойства иконки: размер 32x32, цвет=16; рисовать иконку по пикселям; сохранить файл иконки в папке с файлами проекта командой: File/Save As…; выйти из Image Iditor)
2. Подключение иконки к приложению (выполнить команду: Project/Options…; в открывшемся окне перейти на закладку Application; выбрать иконку кнопкой Load Icon…; в появившемся окне выбрать путь к файлу иконки; нажать кнопку Ok)

Практическая работа №19 «События компонентов (элементов управления), их сущность и назначение»

Цель работы: Получение навыков работы в визуальной среде программирования. Выполнения основных этапов разработки приложений. Изучение возможностей окна Инспектора объектов для задания свойств форме на этапе проектирования приложения. Знакомство со способами написания обработчиков событий, назначением и типами параметром. Изучение возможностей окна Инспектора объектов для задания шаблонов обработчиков событий формы. Способы вызова событий.

Этапы разработки проложений

Процесс разработки приложений в C++ Builder содержит следующие этапы:

1. Построение визуального интерфейса: выбор нужных компонентов и размещение их на форме (кнопки, меню и т.д.).

2. Установка с помощью она Инспектора объектов свойств формы и ее элементов управления.

3. Присоединение кода на языке C++ к компонентам для обработки событий, возникающих от пользователя (с помощью мыши или клавиатуры) или системы (таймер и др.).

4. Компиляция исходного кода C++ и ресурсов формы в исполнимый exe -файл, который может запускаться из среды C++ Builder или из операционной системы Windows как отдельное приложение.

Форма

Одним из важнейших компонентов C++ Builder является форма. **Визуально форма** - это специальное окно, которое составляет основу приложения Buider, и является контейнером для других компонентов. Типичная форма представляет собой прямоугольное окно и может содержать следующие элементы: рамку, заголовок, главное меню под заголовком, строку состояния в нижней части, вертикальную и горизонтальную полосы прокрутки.

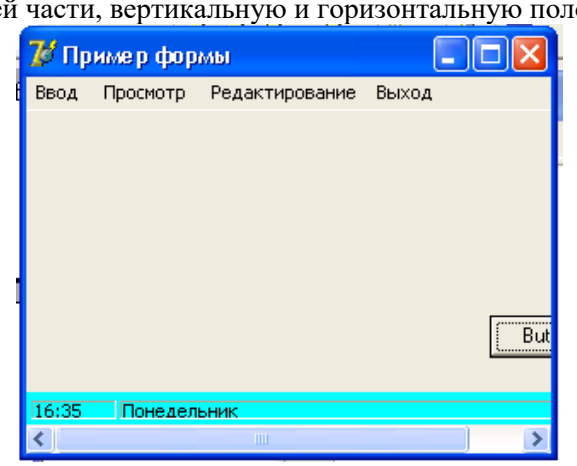


Рисунок 19.1. Вид формы с основными элементами

Все составляющие формы можно менять с помощью свойств формы. Остальная часть пространства формы называется клиентской областью. В ней размещаются различные элементы управления, выводиться текст и графика и т.д.

Программно форма - это объект стандартного класса tform. Как любой объект форма имеет свои свойства, которые можно задавать как на этапе проектирования приложения с помощью окна Инспектора объектов, так и на этапе выполнения приложения из текста программы.

Основные свойства формы

Alphablend – разрешает прозрачность формы. Если свойство равно true, то форма может быть прозрачна.

Alphablendvalue – задает степень прозрачности формы от 0 до 255. Значение 0 свойства означает полностью прозрачна; 255 – полную непрозрачность; полупрозрачная – промежуточное значение.

Autoscroll – простое логическое – определяет, будут ли автоматически появляться полосы прокрутки, если при заданном размере окна не все компоненты помещаются на нем. Если true (по умолчанию), то будут.

Bordericons – задает набор кнопок в правом углу заголовка.

Borderstyle - определяет внешний вид и поведение рамки окна формы.

Borderwidth – ширина рамки.

Caption – задает заголовок окна.

Clientheight, clientwidth – определяет высоту и ширину в пикселях клиентской области формы.

Color – цвет формы.

Icon – определяет пиктограмму, отображаемую в заголовке окна формы.

Left, Top – определяют в пикселях координаты левого угла формы относительно экрана монитора.

Windowstate - состояние формы после запуска.

События

Событие – это действие со стороны пользователя или самой системы относительно запущенного на выполнение приложения. С помощью событий происходит взаимодействие пользователя и системы с запущенным приложением. Метод создания программ на основе возникающих событий, называется **событийно-ориентированным**. **Обработчиком события** - это функция, содержащая команды на языке программирвоания C++, которые должны выполниться при возникновении события, с которым обработчик связан.

3.5.Создание обработчика главного события

У каждого компонента есть **главное событие** – событие, которое с этим компонентом происходит обязательно или наиболее часто. Например, для формы – это событие **oncreate** – создание формы, для кнопки – это onclick – щелчок по кнопке и т.д. Для создания обработчика главного события надо:

- активизировать компонент на форме, для которого создается обработчик;
- выполнить двойной щелчок по компоненту.

Среда C++ Builder подготовит заготовку будущего события.

Создадим обработчик главного события формы – oncreate.

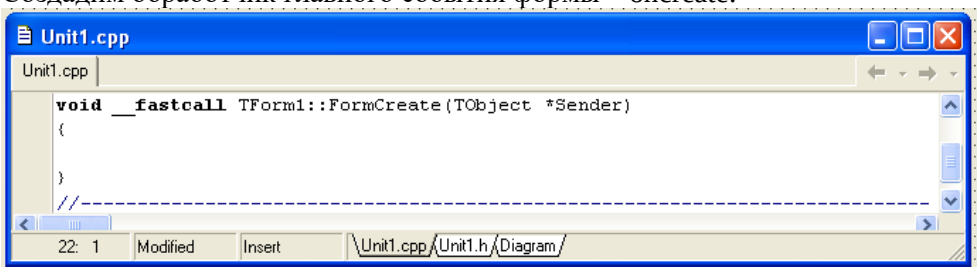


Рисунок 19.2. Окно текста программы с заготовкой главного события формы

Обработчик имеет имя formcreate, один параметр Sender типа *tobject, нет возвращаемого значения, обработчик принадлежит классу tform1. Курсор установиться между {}, где следует написать команды обработчика.

Создание обработчика произвольного события

Для создания заготовки любого события используется окно Инспектора объектов. Требуется выполнить следующие действия:

1. Активизируется компонент на форме, для которого пишется обработчик.
2. В окне Инспекторе объектов перейти на вкладку Events (События). На ней расположен список событий, распознаваемых выбранным компонентом.
3. Найти в списке имя нужного события.
4. Выполнить двойной щелчок в пустой строке справа от события.
5. Из окна Инспектора объектов система вернется в окно Редактора, в котором в позиции курсора вписываются команды события.

Для случая создания шаблона обработчика события щелчка по форме:

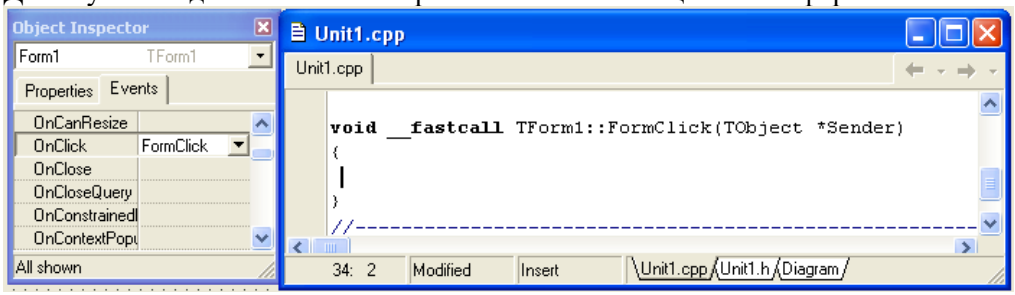


Рисунок 19.3. Создание шаблона произвольного обработчика события

Задание на практическую работу

Необходимо освоить основные свойства формы и способы задания им значений в требуемом диапазоне, а так же освоить основные приемы создания обработчиков событий формы. Для этого надо создать отдельную папку, в которую будут помещаться все файлы, относящиеся к одному приложения. Установить следующие свойства формы с использованием окна Инспектора объектов:

1. Выравнивание формы различными вариантами (свойство Align);
2. Оформить рамку формы различными вариантами (свойство borderstyte);
3. Задать заголовок окна формы (свойству Caption присвоит значение «Моя первая форма»);
4. Подобрать цвет формы (свойство Color);
5. Выбрать вид указателя формы из списка стандартных указателей (свойство Cursor);
6. Поэкспериментировать со свойством прозрачности формы (свойства alphablend и alphablendvalue);
7. Изображение всплывающей подсказки формы (в свойство Hint записать текст подсказки – «Это форма», в свойство showhint записать значение true);
8. Указать состояние формы (свойство windowstaty);
9. Создать иконку (значок) для формы. Для этого надо выполнить два этапа:

Создание иконки. Для этого требуется выполнить:

- вызвать графический редактор командой Tools\Image Iditor.
- выбрать в нем пункт меню File\New\Icon File(ico);
- задать свойства иконки (размер 32x32, цвет=16);
- рисовать иконку по пикселям;
- сохранить файл иконки в папке со всеми файлами данного проекта командой File\Save As…; выйти из Image Editor.

Подключение иконки к форме. Для этого надо выполнить:

- выбрать свойство Icon формы, дважды щелкаем напротив этого свойства на кнопке […]. При этом появляется окно Picture Editor;
- щелкнуть по кнопке Load…. Появиться окно Load Picture. Выбрать в нем нужную иконку;
- щелкнуть по кнопке «Открыть» в окне «Load Picture». При этом происходит возврат в окно Picture Editor, в котором высвечивается нужная иконка;
- щелкнуть по кнопке Ок. Нужная иконка появиться в заголовке.

Сохранить все файлы проекта командой меню File\Save All.

10. В событии onmousedown левая кнопка мыши случайно меняет цвет формы, правая кнопка мыши сворачивает окна приложения до иконки.
11. Событие ondblclick вызывает закрытие формы методом Close, а так как форма является главной, то ее закрытие завершает работу всего приложения.
12. Событие onmousemove выводит текущие координаты мыши в заголовке формы.
13. В событии oncreate активизировать датчик случайных чисел.
14. В событии onkeydown при нажатии клавиш [Alt]+[X] осуществить выход из приложения.

Событие onkeydown анализирует код нажатой клавиши и выполняет необходимые действия для формы в зависимости от нажатой клавиши, а именно:

- при нажатии клавиш управления курсором форма должна перемещаться в указанном направлении (свойства Left и Top для формы);
- при одновременном нажатии клавиши [Shift] и клавиш управления курсором форма должна увеличиваться в размерах в нужном направлении (свойства Left, Top, Width, Height для формы);
- при нажатии клавиши [Ctrl] и клавиш управления курсором, происходит уменьшении размер формы в нужном направлении.

События onmousewheeldown и onmousewheelup меняют прозрачность формы (свойства alphablend и alphablendvalue для формы).

Практическая работа №20 «Создание проекта с использованием компонентов ввода и отображения чисел, дат и времени»

Цель работы: Закрепление на конкретных примерах полученных теоретических знаний при изучении свойств, методов и событий стандартных компонентов: Button – командная кнопка и bitbnt – графическая кнопка.

Компонент Командная кнопка (Button)

Основное назначение данного компонента – управлять ходом работы приложения путем щелчка по нему мышью. Компонент находиться на странице Standard Палитры компонентов. Отменим дополнительные особенности кнопки.

Bool Cancel – свойство определяет, будет ли выполняться обработчик события onclick при нажатии на клавишу [Esc]. Если true, то будет.

Bool Default – свойство определяет, будет ли выполняться обработчик события onclick при нажатии на клавишу [Enter]. Если true, то будет.

Void Click() – программный метод вызова события onclick.

Void setfocus() – метод передачи компоненту фокуса компоненту.

Onclick – главное событие кнопки.

Компонент Кнопка с картинкой (bitbtn)

Данный компонент - потомок кнопки Button. Добавил возможность отображать не только заголовок, но и растровую картинку на своей поверхности, следовательно, к свойствам кнопки Button добавились свойства, отвечающие за картинку на кнопке. Располагается на вкладке Additional.

Отменим дополнительные свойства графической кнопки.

Glyph – битовая матрица растрового рисунка (глиф) на кнопке (по умолчанию None - отсутствие рисунка). Подготовить файл рисунка формата bmp можно в графическом редакторе. Свойство Glyph вызывает окно, в котором выбирается путь к файлу картинки. Задание этого свойства можно выполнить и программно строчкой:

Kind – задает вид кнопки со стандартными картинками размера 16x16 пикселей Если свойство имеет значение bkclose, то при щелчке по такой кнопке происходит выход из приложения без кода программы.

Tbuttonlayout Layout - задает положение глифа относительно поясняющего его текста на поверхности кнопки.

Margin – расстояние в пикселях между краем кнопки и глифом. Край кнопки задается свойством Layout. Если свойство имеет значение –1, что означает центрирование глифа и заголовка на кнопке.

Spacing – расстояние в пикселях между глифом и заголовком. По умолчанию равно 4. При значении 0, заголовок и картинка будут вплотную друг к другу. Если значение свойства равно –1, то заголовок центрирован между глифом и краем кнопки.

Задание на практическую работу

1. Разработать приложение «Калькулятор» с использованием изученных элементов интерфейса. Создать программный код разделяемых событий для кнопок с цифрами и знаками операций. Приложение должно содержать проверки возможных ошибок при вводе исходных значений. В программе использовать две глобальные вещественные переменные, в которые будут сохраняться операнды, над которыми будут выполняться действия, а так же символьную переменную для хранения символа выбранной операции.
2. Создать простейший вид калькулятора. Калькулятор должен иметь панель для отображения числа, десять цифровых кнопок, кнопку разделителя целой и дробной частей, кнопку «=», кнопку смены знака и четыре кнопки с основными математическими действиями
3. Дополнительно к разработанному ранее калькулятору добавляются кнопки памяти.
4. Разработать инженерный калькулятор с вычислением стандартных математических функций: квадратного корня, возведения в квадрат, тригонометрических функций, возведения в степень и другие.

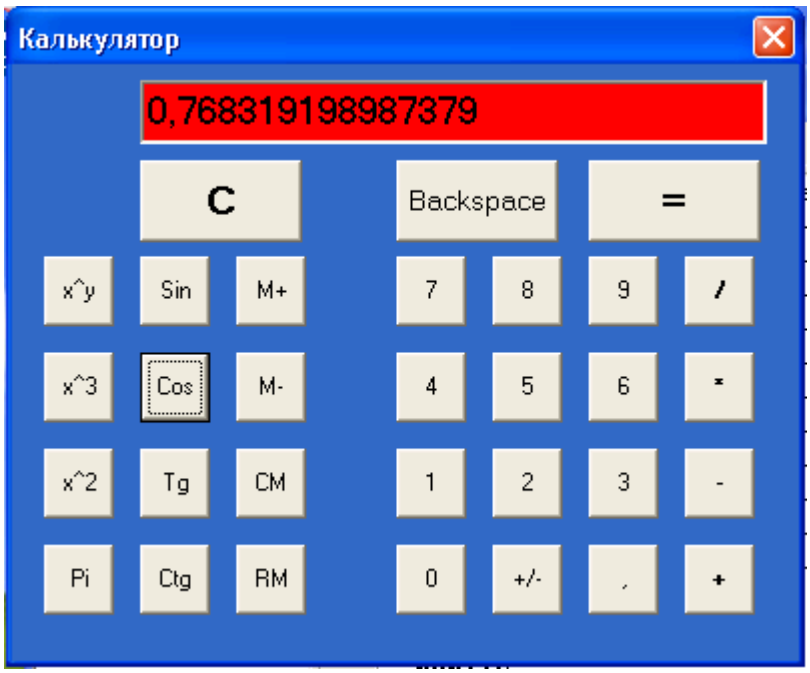


Рисунок 20.1. Приложение «Калькулятор»

Практическая работа №21 «Создание проекта с использованием компонентов для работы с текстом»

Цель работы: сформировать умения по использованию компонентов для работы с текстом, сформировать умения по созданию приложений с компонентами для работы с текстом.

Запустите интегрированную среду разработчика.Создайте новый проект.

1. Создайте в папке своей группы новую папку и назовите её **Label**. Сохраните проект в созданную ранее папку, выполнив команду **File - Save Project as...**

Разместите на форме компонент **Label** и кнопку **Button** вкладки палитры компонентов **Standard**.

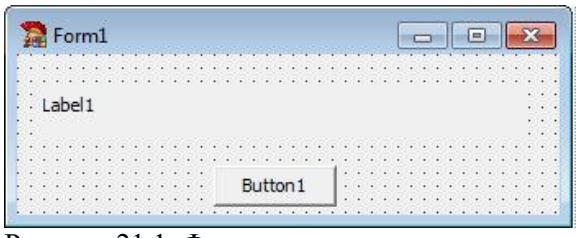


Рисунок 21.1. Форма

Компоненты ввода — вывода данных можно условно разделить на несколько различных блоков: компоненты вывода текстовой информации на экран; однострочные поля ввода текстовой и числовой информации; многострочные поля ввода.

Для вывода определенной информации на экран, кроме уже ранее используемого компонента **Label**, есть и другие компоненты. Текст, который будет отображен, можно задавать как на этапе разработки формы, так и в процессе выполнения программы, присвоив значение свойству **Caption**.

Задайте для формы заголовок **«Работа с компонентом Label»**.

Выделите надпись **Label1**, найдите в **Object Inspector** свойство **Caption** и вставьте новое название надписи **Моя первая программа на языке Delphi.**

Выделите кнопку **Button1**, найдите в **Object Inspector** свойство **Caption** и вставьте новое название кнопки **Output**.

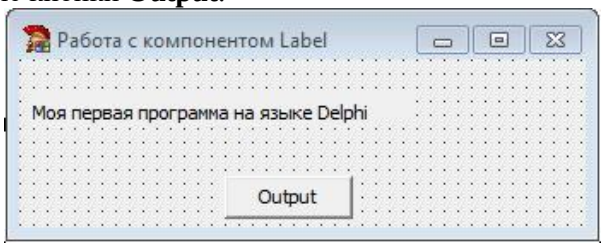


Рисунок 21.2. Работа с меткой

Перейдите в **Object Inspector** на страницу **Events**, найдите событие **onclick** и справа от него дважды щелкните мышкой. Оказавшись в коде программы, но теперь в процедуре кнопки **Button1**, напишите следующий программный код:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
Label1.Visible:=not label1.Visible;
if Label1.Visible then
 Label1.Font.Size:=Label1.Font.Size+1;
end;
```

Рисунок 21.3. Код процедуры

Этой программе при каждом очередном нажатии происходит изменение свойства **Visible**, вследствие чего надпись то появляется, то исчезает с экрана, а также происходит увеличение свойства **Size**.

Для вывода определенной информации на экран, кроме уже описанного компонента **Label**, может быть также использован компонент **Panel** с той же самой вкладки или **statictext** со страницы **Additional**. Они имеют незначительные отличия от компонента **Label**.

Остальные компоненты позволяют вводить и редактировать информацию, включая возможность выделения, копирования, удаления и вставки фрагментов текста. Отметим общие для всех редакторов методы: **Clear** — удалить весь текст, помещенный в редакторе; **clearselect** — удалить выделенный фрагмент текста; **copytoclipboard** — копировать в буфер выделенный

фрагмент, **cuttoclipboard** — удалить из текста выделенный фрагмент и поместить его в буфер, **pastefromclipboard** — копировать текст из буфера в то место редактора, где в данный момент находится курсор.

Для всех редакторов определено дополнительное событие **onchange**, возникающее, когда изменяется текст, находящийся в редакторе.

Для ввода или вывода одной строки могут использоваться компоненты **Edit** со страницы **Standart** и **maskedit** со страницы **Additional**. Основное свойство данных компонентов — это строка, которая либо вводится, либо выводится. Данное свойство имеет имя **Text** и тип **String**, доступное как во время подготовки, так и время выполнения. Логическое свойство **readonly** позволяет запретить изменения,

А целочисленное свойство **gettextlen** выдает текущую длину строки

Сохраните изменения и запустите проект. Протестируйте его работу.

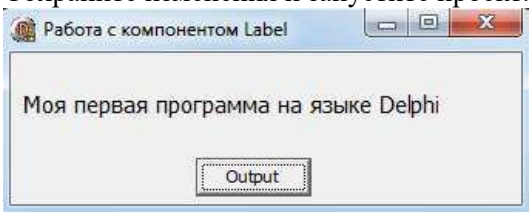


Рисунок 21.4. Окно программы

2. Разработать программу, которая при вводе текста в первый компонент **Edit1**, во втором компоненте **Edit2** отображает реальную длину вводимой строки. Кроме этого, при выходе из компонента **Edit1** его содержимое копируется в буфер обмена и удаляется, а при возвращении в программу появляется снова.

Создайте в папке своей группы новую папку и назовите её **Edit1**. Сохраните проект в созданную ранее папку, выполнив команду **File - Save Project as...**

Разместите на форме два компонент **Edit** вкладки палитры компонентов **Standard**.

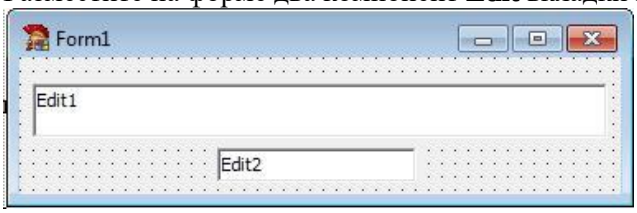


Рисунок 21.5. Обработка текстового поля

Задайте для формы заголовок **«Работа с компонентом Edit»**.

Выделите текстовое поле **Edit1**, найдите в **Object Inspector** свойство **Text** и оставьте его пустым. Аналогичные действия выполните со вторым текстовым полем.

Выделите компонент **Edit1**, в **Object Inspector** перейдите на вкладку **Events** и найдите событие **onchange** и справа от него дважды щелкните мышкой. Оказавшись в коде программы, но теперь в процедуре текстового поля **Edit1**, напишите следующий программный код:

```
procedure TForm1.Edit1Change(Sender: TObject);
begin
Edit2.Text:= IntToStr(Edit1.GetTextLen);
end;
```

Рисунок 21.6. Код процедуры

Выделите компонент **Edit1**, в **Object Inspector** перейдите на вкладку **Events** и найдите событие **onenter** и справа от него дважды щелкните мышкой. Оказавшись в коде программы, но теперь в процедуре текстового поля **Edit1**, напишите следующий программный код:

```
procedure TForm1.Edit1Enter(Sender: TObject);
begin
Edit1.PasteFromClipboard;
end;
```

Рисунок 21.7. Код процедуры

Выделите компонент **Edit1**, в **Object Inspector** перейдите на вкладку **Events** и найдите событие **onexit** и справа от него дважды щелкните мышкой. Оказавшись в коде программы, но теперь в процедуре текстового поля **Edit1**, напишите следующий программный код:

```
procedure TForm1.Edit1Exit(Sender: TObject);
begin
 Edit1.SelectAll;
 Edit1.CopyToClipboard;
 Edit1.Clear;

end;
```

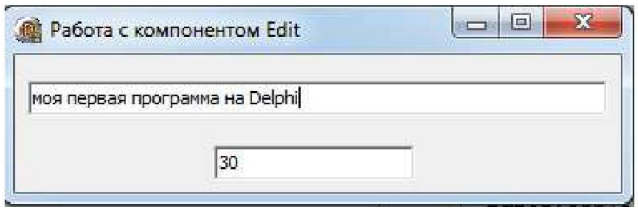


Рисунок 21.8. Код процедуры

Сохраните изменения и запустите проект. Протестируйте его работу.

3. Разработать программу, которая запрещает ввод в компонент **Edit1** подряд двух одинаковых символов.

Создайте в папке своей группы новую папку и назовите её **Edit2**. Сохраните проект в созданную ранее папку, выполнив команду **File - Save Project as...**

Разместите на форме компонент **Edit** вкладки палитры компонентов **Standard**.

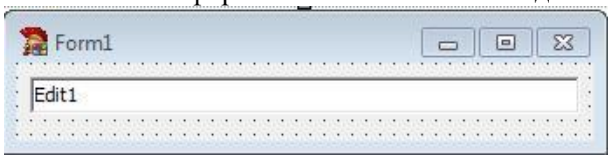


Рисунок 21.9. Окно формы

Перейдите в код программы (на клавиатуре нажмите клавишу **F12**). Введите в раздел **VAR** глобальную переменную **ch** типа **char**, в которой будет храниться последний нажатый символ.

```
var
  Form1: TForm1;
  ch: char;
```

Рисунок 21.10. Объявление пременной

Задайте для формы заголовок **«Работа с компонентом Edit»**.

Выделите текстовое поле **Edit1**, найдите в **Object Inspector** свойство **Text** и оставьте его пустым.

Создайте процедуру обработки события **keypress** текстового поля **Edit1**, параметр **Key** данной процедуры содержит символ нажатой клавиши. Если вновь введенный символ совпадает с только что нажатым символом, то он игнорируется. В противном случае, новый символ запоминается в переменной **ch**.

```
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
if ch=key then key:=#0
 else ch:=key;
end;
```

Рисунок 21.11. Код процедуры

Сохраните изменения и запустите проект. Протестируйте его работу.

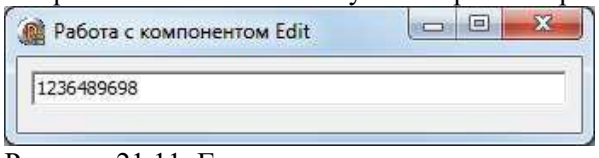


Рисунок 21.11. Готовая программа

4. Разработать программу, которая считает количество нажатий на кнопку и выдает это значение в компоненте **Edit**.

Создайте в папке своей группы новую папку и назовите её **Edit3**. Сохраните проект в созданную ранее папку, выполнив команду **File - Save Project as...**

Разместите на форме компонент **Edit** и кнопку **Button** вкладки палитры компонентов **Standard**.

Используя **Object Inspector,** задайте значения свойств компонентов формы в соответствии с рисунком.

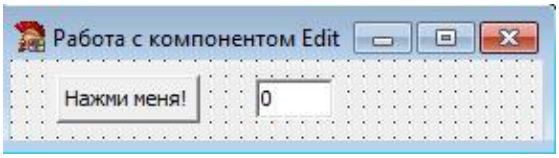


Рисунок 21.12. Окно формы

Если в целочисленной переменной **i** будем считать количество нажатий, то процедура обработки события **onclick** кнопки может быть записана в виде:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
i:=i+1;
 Edit1.Text:=IntToStr(i);
end;
```

Рисунок 21.13. Код процедуры

Однако остается вопрос, где описывать данную переменную **i**. Если сделать это внутри данной процедуры, то также необходимо осуществлять обнуление переменной, а это приведет к получению одного и того результата, равного единице. Следовательно, переменная **i** должна быть глобальной переменной в модуле, а ее начальная инициализация должна происходить в процедуре, которая выполняется всего один раз, и всего один раз происходит это событие. Таким событием является создание формы **oncreat**, данное событие произойдет один раз и процедура **formcreate(Sender:tobject)** будет вызвана всего один раз.

Перейдите в код программы (на клавиатуре нажмите клавишу **F12**). Введите в раздел **VAR** глобальную переменную **i** типа **integer**, в которой будет храниться последний нажатый символ.

Выделите форму, в **Object Inspector** перейдите на вкладку **Events** и найдите событие **oncreate** и справа от него дважды щелкните мышкой. Оказавшись в коде программы, но теперь в процедуре формы **Form1**, напишите следующий программный код:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
i:=0;
end;
```

Рисунок 21.14. Код процедуры

Сохраните изменения и запустите проект. Протестируйте его работу.

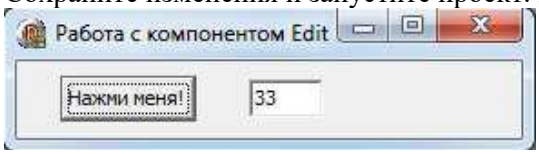


Рисунок 21.15. Готовая программа

При возникновении необходимости сделать данную переменную **i** общедоступной, можно поместить описание переменной в интерфейсной части модуля после служебного слова **public**. Именно так, как правило, и поступают. В модуле необходима всего одна переменная — форма, а все остальные описываются в виде полей. В этом случае описание формы будет иметь вид:

```
type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    i: integer;
  end;

var
  Form1: TForm1;
```

Рисунок 21.16. Описание переменных

Программе для обращения к переменной **i** необходимо писать ее полное имя **Form1.i**. Однако код процедур обработки событий можно и не переписывать, поскольку процедуры обработки описаны непосредственно в формы, следовательно, данное числовое поле доступно непосредственно.

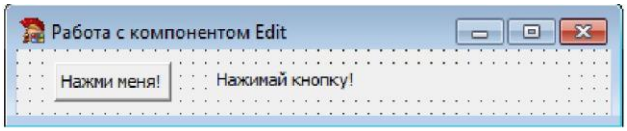
Внесите изменения в программный код в соответствии с рисунком. Сохраните проект и протестируйте работу приложения.

Данную программу можно легко модифицировать так, чтобы после определенного количества нажатий появлялось некоторое сообщение или кнопка блокировалась, или приложение автоматически закрывалось. Результат можно отображать не только посредством компонента **Edit**, но и через не редактируемый текст, т.е. Компонент **Label**, что в данном случае является более естественным. Свойству **Visible** компонента **Label** присваиваем **False**, т. Е. При открытии формы надпись отражаться не

Будет. Затем, как и ранее, при нажатии на кнопку переменная **i** увеличивается на **1**. Когда значение переменной **i** будет равно **10, 20, 30** или **40** компонент **Label** становится видимым, а свойству **Caption** надписи присваиваем значение «**Вы нажали i раз**». При следующем нажатии надпись становится невидима. Когда **i** станет равной **50**, кнопку необходимо сделать неактивной, для чего необходимо изменить значение свойства **Enabled** с **True** — включено на **False** — выключено.

Разместите на форме компонент **Label**. Задайте свойство **Caption** равным «**Нажимай кнопку!**». Удалите с формы компонент **Edit1**.

Перейдите в окно редактирования кода и внесите изменения в код процедуры обработки события **Onclick** кнопки.




```
procedure TForm1.Button1Click(Sender: TObject);
begin
Label1.Visible:=False;
 i:=i+1;
if (i=10)or(i=20)or(i=30)or(i=40) then
 Begin
 Label1.Visible:=True;
 Label1.Caption:='Вы нажали '+intToStr(i)+'раз';
 end;
if i=50 then
 Begin
 Label1.Visible:=True;
 Label1.Caption:='Вы нажали УЖЕ'+intToStr(i)+'раз';
 Button1.Enabled:=False;
 end;
end;
```

Рисунок 21.17. Код процедуры

Сохраните проект и протестируйте работу приложения.

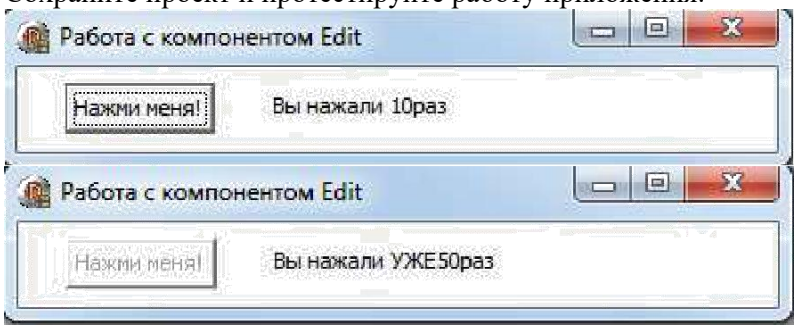


Рисунок 21.18. Готовая программа

5. Разработать программу, которая считывает строку под определенным номером и помещает её в текстовое поле.

Создайте в папке своей группы новую папку и назовите её **Memo1**. Сохраните проект в созданную ранее папку, выполнив команду **File - Save Project as...**

Для ввода или вывода нескольких строк могут использоваться компоненты **Memo** со страницы **Standard** и **richedit** со страницы **Win32** (полный текстовый редактор для **RTF**-файлов). Многие свойства у данных компонентов аналогичны свойствам компонента **Edit**, однако для возможности доступа к строкам вместо свойства **Text** имеется свойство **Lines**, при выборе которого во время проектирования задается начальное значение строк.

Для доступа к строкам во время выполнения программы также используется свойство **Lines** класса **tstring**. Подробнее остановимся на классе **tstring**, с которым в последствии мы будем еще встречаться. А именно, этот класс обладает свойствами: **Count** — целочисленное свойство, определяющее количество элементов в списке (в данном случае это будет количество строк в компоненте **Memo**), **Text** — свойство, содержащее все строки списка, **String[Index:Integer]** — свойство, определяющее строку с номером **Index**. Учитывая, что нумерация строк начинается с 0 и свойство **String** является свойством по умолчанию, можно утверждать, что свойства **Memo1.Lines.String[3] и Memo1.Lines[3]** эквивалентны и указывают на четвертую строку в компоненте **Memo1**.

Класс **tstring** обладает также рядом методов, среди которых отметим следующие: **Add(St:String):integer** добавляет строку **St** и возвращает номер этой строки; **Delete(Index:Integer)** удаляет строку с номером **Index; Insert (Index:Integer, St:String)** вставляет строку с номером **Index, Clear** полностью уничтожает все содержимое компонента.

Все содержимое компонента можно записать в файл с помощью метода **savetofile** или прочитать из файла посредством методом **loadfromfile**. Аналогичным образом можно поступить и с потоком, направив в него весь файл, или прочитать файл из потока.

Важным свойством компонентов **Memo** и **richedit** является **scrollbar**, которое определяет, будет ли окно содержать горизонтальные или вертикальные линейки прокрутки.

Компонент **richedit** обладает всеми характеристиками, присущими компоненту **Memo**, однако имеет богатые возможности для работы с текстовым форматом **RTF**. Данный формат предполагает возможность разбивать текст на параграфы. Для этого существуют специальные свойства: **selattributes** определяет атрибуты выделенного фрагмента и **Paragraph** — атрибуты абзаца.

Разместите на форме компоненты **Edit,** компонент **Memo,** компонент **Label**, кнопку **Button** вкладки палитры компонентов **Standard**.

Используя **Object Inspector,** задайте значения свойств компонентов формы в соответствии с рисунком.

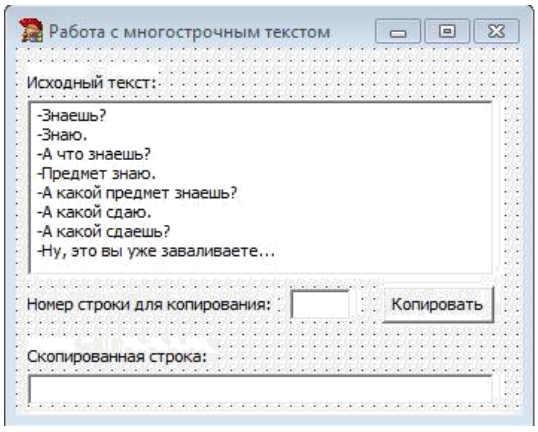


Рисунок 21.19. Окно формы

Для реализации решения задачи процедура обработки события **onclick** кнопки может быть записана в виде:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
Edit2.Text:=memo1.Lines[strToInt(edit1.Text)-1];
Memo1.Lines.Delete(strToInt(edit1.Text)-1);
end;
```

Рисунок 21.20. Код процедуры

Сохраните проект и протестируйте работу приложения.

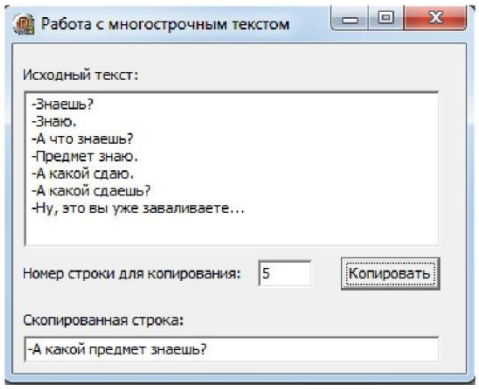


Рисунок 21.21. Готовая программа

6. Разработать программу, которая сохраняет текст, набранный в поле **Memo** в файл, имя которого задано в текстовом поле **Edit**.

Создайте в папке своей группы новую папку и назовите её **Memo2**. Сохраните проект в созданную ранее папку, выполнив команду **File - Save Project as...**

Разместите на форме компонент **Edit,** компонент **Memo,** компонент **Label**, кнопку **Button** вкладки палитры компонентов **Standard**.

Используя **Object Inspector,** задайте значения свойств компонентов формы в соответствии с рисунком.

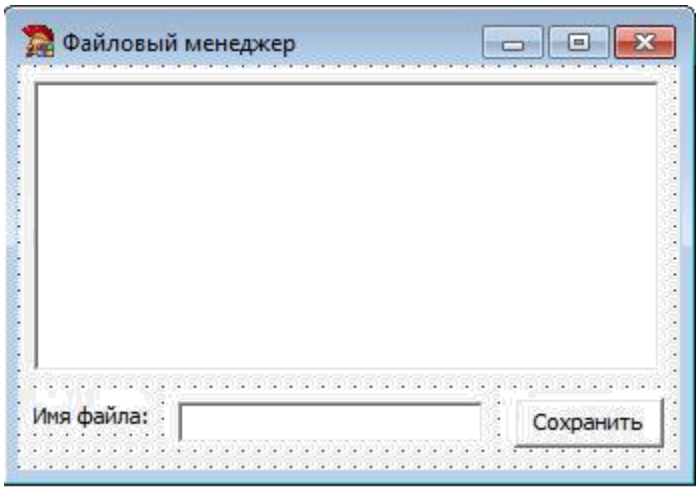


Рисунок 21.22. Окно формы

Процедура обработки события **onclick** кнопки **Button** будет состоять из одной строки, и соответственно, листинг будет иметь следующий вид:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
Memo1.Lines.SaveToFile(Edit1.Text);
end;
```

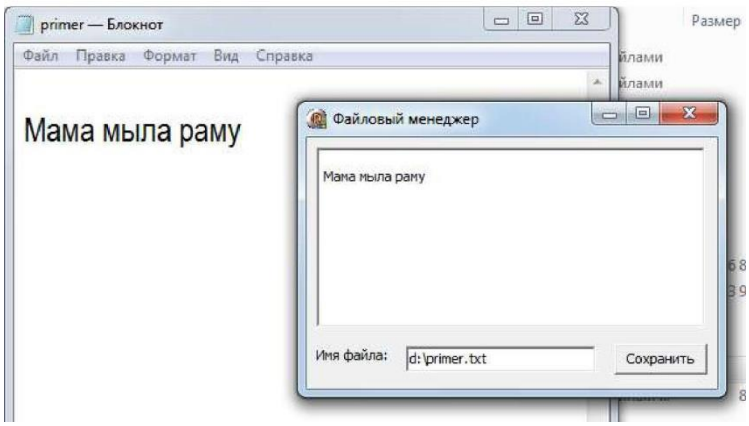


Рисунок 21.23. Код процедуры

Сохраните проект и протестируйте работу приложения.

Практическая работа №22 «Создание процедур на основе событий»

Цель работы: Закрепление на конкретных примерах полученных теоретических знаний при изучении свойств, методов и событий стандартных компонентов: Timer – невизуального компонента и Panel – панель. Создание программного кода системного события от внутренних часов. Применение основных команд получения и обработки информации о текущем времени и текущей даты.

Компонент Таймер (Timer)

Компонент Timer относится к невизуальным компонентам, т.е. При выполнении приложения на экране он не отображается. Находится данный компонент на вкладке System. Основное назначение компонента — это управление приложением в реальном режиме времени. Компонент отслеживает изменения в системном таймере и через нужный интервал времени выполняет запрограммированные для него действия. Чтобы включить таймер надо свойство Enabled задать равным true, а свойство Interval задать положительным значением. Отключить таймер можно либо свойству Enabled присвоить false, или в свойство Interval записать значение 0.

Ontimer — единственное событие компонента. Является примером системного события, возникающего без участия пользователя. Событие срабатывает для включенного таймера через каждые количество миллисекунд, указанное в свойстве Interval.

Компонент Панель (Panel)

Компонент используется как декоративный элемент формы — рамка, в которой можно как в контейнере размещать (группировать) другие компоненты интерфейса. Компонент имеет рамку и бордюр. Все компоненты, расположенные на панели, имеют одинаковые свойства и перемещаются вместе с панелью. Компонент расположен на вкладке Standart. К основным свойствам компонента относят:

Align — выравнивание панели относительно формы.

Caption — строка текста внутри панели.

Alignment — выравнивание текста строки внутри панели.

Bevelwidth — ширина рамки в пикселях. По умолчанию равно 1.

Borderwidth — ширина бордюра — линии, разделяющей внутреннюю и внешнюю рамки. По умолчанию свойство равно 0 (рамки нет).

Color — цвет плоскости панели и ее бордюра.

Font — установка параметров шрифта компонентов, расположенных на панели.

Style —установка стиля надписей компонентов, расположенных на панели.

Пример программы секундомера

```
Tform1 *Form1;                                  //Глобальные переменные
Int s, m;
//Функция формирования строки времени из целых составляющих
String    fun (int min, int sec)
{         String s, s1;
S1=inttostr (min);                      //Получение из минут строки
S="";                                   //Результирующая - сначала пустая
If (min<10)      s="0"+s1;              //Если минуты однозначные, то добавить 0
Else                    s=s1;                   //иначе – копирует исходную строку
S=s+":";                                        //Добавляем к строке разделитель
S1=inttostr (sec);                              //Получение из секунд строки
If (sec<10)             s=s+"0"+s1;             //Если секунды однозначные, то добавить 0
Else                    s=s+s1;         //иначе добавить без изменений
Return (s);                                     //Возвратить полученную строку
}
//Начальные действия при запуске приложения
Void __fastcall tform1::formcreate (tobject *Sender)
{
Form1->Button1->Caption="Включить";
Form1->Timer1->Enabled=false;
Form1->Timer1->Interval=1000;
S=m=0;
Form1->Panel1->Caption=fun (m, s);
}
```

```
//Щелчок по кнопке для включения секундомера
Void __fastcall tform1::Button1Click(tobject *Sender)
{       s=m=0;
If (Form1->Button1->Caption=="Включить")
{
Form1->Button1->Caption="Выключить";
Form1->Timer1->Enabled=true;
}
Else
{
Form1->Button1->Caption="Включить";
Form1->Timer1->Enabled=false;
Form1->Panel1->Caption=fun(m,s);
}
}
//Событие таймера для изменения счетчиков
Void __fastcall tform1::Timer1Timer(tobject *Sender)
{       s++;                            //Счетчик секунд
If (s==60)                              //Накопилась минута
{
M++;                            //Увеличить счетчик минут
S=0;                            //Сбросить секунды
}
Form1->Panel1->Caption=fun (m, s);
}
```

Задание на практическую работу

1. Описать две глобальные целочисленные переменные для счетчика секунд и минут секундомера.

2. Расположить на форме четыре панели для отображения текущего времени, даты, название дня недели и панель с секундомером. Подобрать цвета, стиль и расположение панелей на форме.

3. В любом месте формы расположить два компонента Timer: один для получения текущего времени, другой для отображения работы секундомера. Задать для обоих компонентов свойства Interval в значение 1000.

4. Добавить на форму две кнопки: одну для выхода из приложения, другую для включения/выключения секундомера.

5. Написать программный код события oncreate для формы, в котором:

- включить таймер для текущего времени (свойство Enabled);
- отключить таймер для секундомера;
- обнулить значения счетчиков для секундомера;
- кнопке задать заголовок «Включить»;
- в панель для секундомера вывести значения счетчиков с учетом впереди стоящего нуля, т.е. В свойство Caption строку «00:00»;
- отобразить на панелях текущую дату, день недели и временя.

6. Написать программный код для события onclick командной кнопки:

Проверить значение свойства Caption этой кнопки: если там значение «Включить», то поменять его на «Выключить» и наоборот;

7. Написать программный код для события ontimer компонента Timer1, в котором обновлять показания в панелях текущей даты, дня недели и времени.

8. Написать программный код для события ontimer компонента Timer2, в котором по названию на кнопке выводить или обнулять счетчики таймера..

9. Добавить на форму кнопку «Pause». Написать для нее программный код события onclick, в котором останавливать или продолжать работу таймера Timer2 (менять свойство Enabled компонента Timer2 на противоположное).

10. Добавить на форму две графические кнопки со стрелками. Написать программных код событий onclick для них, в которых увеличивать (стрелка вверх) или уменьшать (стрелка вниз) на

100 значение свойства Interval компонента Timer2, что повлияет на частоту срабатывания события ontimer. Проверять на допустимые значения свойства Interval.

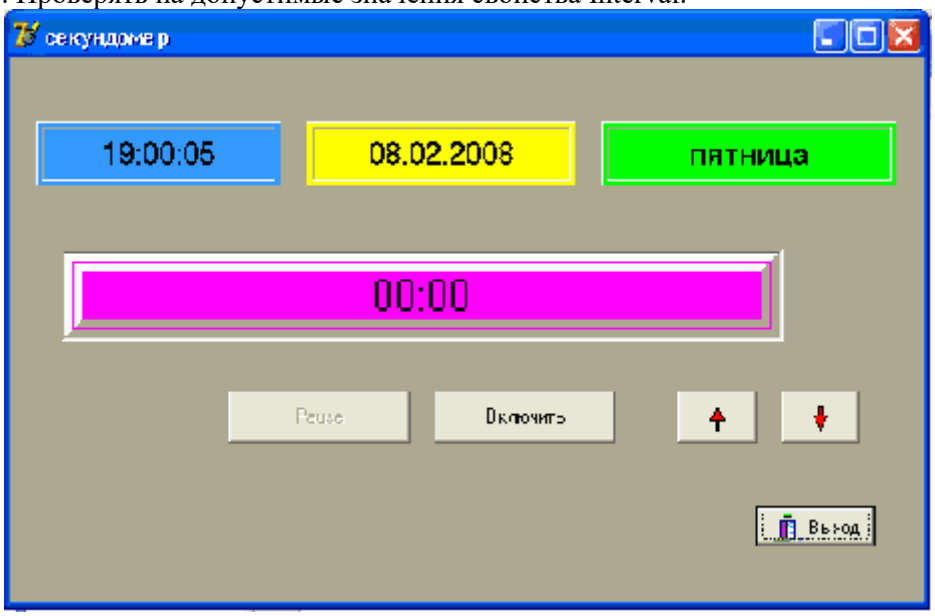


Рисунок 22.1. Приложение «Секундомер»

Практическая работа №23 «Создание проекта с использованием кнопочных компонентов»

Цель работы: Закрепление на конкретных примерах полученных теоретических знаний при изучении свойств, методов и событий стандартного компонента Edit. Освоение основных приемов контроля над вводом данных.

Компонент Строка ввода (Edit)

Однострочный редактор Edit - это поле, в котором можно вводить с клавиатуры и редактировать строку. В компоненте действуют клавиши управления курсором, Delete и Backspace. Можно выделять части текста и работать с буфером обмена стандартными клавишами. Компонент на странице Standart.

Основные свойства компонента

Text - содержит введенную пользователем строку. Свойство можно задавать на этапе проектирования или программно. Выравнивание и перенос текста невозможны.

Maxlength - определяет максимальное количество символов, которое пользователь может ввести в поле редактирования. По умолчанию стоит 0, что означает отсутствие ограничения.

Autosize — определяет, будет ли высота компонента автоматически подстраиваться под размер шрифта вводимой строки.

Borderstyle – задает стиль бордюра.

Passwordchar - символ, который появляется в строке ввода вместо реально введенных символов (используется для ввода пароля). Действительно введенный текст будет содержаться в свойстве Text. Если значение свойства равно #0 (по умолчанию), то режим пароля не установлен и отображаются реально введенные символы.

Readonly — может ли пользователь изменять текст в компоненте Edit.

Color, Font – цвет фона и шрифт текста в компоненте.

Int sellength, int selstart, String seltext - неопубликованные свойства – определяют количество выделенных символов, номер (начинал с 0) первого выделенного символа и сам выделенный текст. Если выделенного символа нет, то свойство selstart указывает позицию курсора ввода.

Основные методы компонента

Clear() - удаляет всю строку и устанавливает значение свойства Text равным пустой строке.

Clearselection() - удаляет выделенный фрагмент в строке.

Copytoclipboard() - копирует выделенный фрагмент в буфер.

Cuttoclipboard() - переносит выделенный фрагмент в буфер обмена.

Pastefromclipboard() - замещает выделенный фрагмент текста или вставляет в место, указываемое курсором ввода, содержимое буфера обмена.

Selectall() – выделяет всю строку в компоненте.

3.4. Основные события компонента

Onchange (tobject Sender) - является главным событием компонента Edit. Возникает, когда возможно изменился текст в поле компонента, т.е. При изменении свойства Text. Обычно в этом событии производят проверку или сохранение новой введенной информации.

Onkeypress (tobject Sender, char Key) – возникает при нажатии печатной клавиши. В обработчике можно распознать вводимый символ и при необходимости изменить его или запретить ввод. Обычно в обработчике события onkeypress() проводят контроль за правильностью ввода информации, и в случае неверного ввода либо сообщают об этом пользователю, либо подавляют неверно введенный символ.

Пример программы с контролем ввода

В компоненте Edit ведется контроль правильности ввода вещественного числа.

```
Void __fastcall tform1::Edit1KeyPress(tobject *Sender, char &Key)
{if (!Isdigit(Key) && Key!=8)          //Не цифра и не backspace
Switch (Key)                           //Анализ введенного символа
{
Case 13: Form1->selectnext(Edit1,true,true);     //Enter
Break;
Case 27: Form1->Edit1->Clear();                  //Esc
Break;
Case '.': case ',':                    //Точка или запятая
```

```
Key=decimalseparator;          //Заменить на разделитель
If (Form1->Edit1->Text.Pos(Key)!=0)  Key=0;//Не единственная
Break;
Case '-':                                  //Минус
If (!Form1->Edit1->Text.isempty())              Key=0;
Break;
Default:          Key=0;          //Игнорировать остальные символы
}
}
```

Задание на практическую работу

Разработать приложение «Конвертор величин измерения» с использованием компонента ввода строки Edit. Создать программный код необходимых событий.

В событии onkeypress компонента Edit выполнять контроль за вводом информации. Щелчком по кнопке осуществлять перевод величин в одном направлении. Выделять цветом и шрифтом активный компонент Edit.

Добавить кнопку для перевода информации в обратном направлении. Так же проводить контроль за правильностью ввода.

Разместить на форме кнопку «Сохранять» и после каждого выполненного конвертирования, результаты помещать в текстовый файл в форме:

Время	Операция	Результат

Варианты индивидуальных заданий

1. Перевод из рублей в доллары
2. Перевод из рублей в евро
3. Перевод из чисел между 16-ричной и 10-тичной системами счисления
4. Переводы температур между Цельсием и Фаренгейту: $t_c = \frac{5(t_f - 32)}{9}$
5. Переводы из чисел между 2-ичной и 10-тичной системами счисления
6. Переводы углов между градусами и радианами
7. Перевод из чисел между 16-ричной и 2-ичной системами счисления
8. Переводы секунд в часы, минуты и секунды
9. Переводы длин между верстами и метрами (1 верста=1066,8 метра)
10. Переводы длин между саженями и метрами (1 саженьа=2,1336 метра)
11. Переводы длин между дюймами и сантиметрами (1дюйм=2,54сантиметра)
12. Переводы длин между морскими милями и метрами (1 м.м.=1852,2 метра)
13. Переводы весов между фунтами и килограммами (1 фунт=0,40951 кг)
14. Переводы скоростей между км/ч и м/с
15. Переводы весов между пудами и килограммами (1 пуд=16,380 кг)
16. Переводы для кругов площадей в радиусы

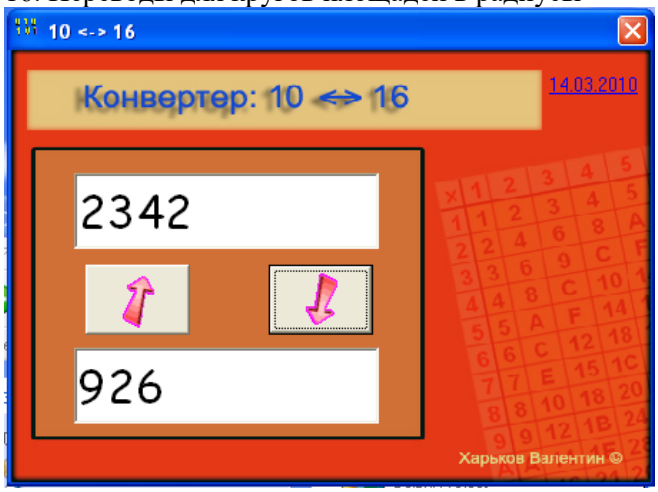


Рисунок 23.1. Приложение «Конвертер единиц измерения»

Практическая работа №24 «Создание проекта с использованием компонентов стандартных диалогов и системы меню»

Цель работы: Закрепление на конкретных примерах полученных теоретических знаний при изучении свойств, методов и событий стандартного компонента Label и команд ввода-вывода информации. Применение основных команд для ввода пароля и вывода необходимых расчетных данных.

Компонент надпись (Label)

Надпись - статический текст, который нельзя редактировать во время выполнения программы. Используется для вывода информации на форму и для заголовков управляющих элементов, у которых отсутствует свойство Caption. Компонент Label находится на вкладке Standart. Перечислим основные свойства компонента.

Caption - текст надписи.

Aligment - задает способ выравнивания надписи в Label по горизонтали.

Layout – задает способ выравнивания надписи в Label по вертикали.

Autosize - указывает, будет ли размер компонента Label подстраиваться под размер выводимой надписи.

Transparent - отвечает за прозрачность фона компонента. Если свойство равно true, то надпись прозрачна (надписи поверх рисунков, например, на географической карте). Если свойство равно false, то в свойство Color устанавливается цвет фона надписи. Все атрибуты надписи задаются в свойстве Font.

Wordwrap — разрешение на разбивку строки на несколько строк. Перенос возможен, если свойства Word Wrap и autosize установить в значение true.

Команды диалога

Команды диалога создают типовые окна интерактивного ввода-вывода. Используются для вывода пользователю необходимых сообщений и получения от него ответа. Данные подпрограммы не входят в палитру компонентов C++ Builder, они не являются объектами, следовательно, не имеют свойств, методов и событий. На этапе проектирования окна не отображаются, а появляются при выполнении приложения.

Окно сообщения showmessage

Используется для привлечения внимания пользователя, например, чтобы информировать его об ошибке, подтвердить запрос и т.д. Чтобы окно появилось на экране надо вызвать функцию showmessage (“Сообщение”) и передать как параметр выводимое в окне сообщение. Команда выводит окно в центре экрана со строкой сообщения и кнопкой OK. Заголовок окна совпадает с именем файла проекта. Окно диалога будет находиться на форме до тех пор, пока пользователь его не закроет.

Окно с сообщением и выбором пользователя messagedlg

Команда messagedlg является более универсальной, чем команда showmessage, т.к. Не только выводит сообщение, но анализирует реакцию пользователя на это сообщение. Общий вид команды:

Выбор=messagedlg (“Сообщение”, Тип_окна, Кнопки, Справка);

Параметр Тип_окна задает тип сообщения. Параметр Кнопки задает набор кнопок окна. Параметр Справка определяет раздел справочной системы, появляющийся при нажатии haf1. Обычно он равен 0. При нажатии любой из кнопок, диалоговое окно закрывается, и функция messagedlg возвращает результат — выбранную кнопку из списка кнопок, находящихся в окне.

Окна ввода inputbox и inputquery

Данные окна используются для получения от пользователя необходимых исходных данных. Общий вид команд:

Результат=inputbox (“Заголовок”, “Подсказка”, “По_умолчанию”);

Проверка=inputquery (“Заголовок”, “Подсказка”, Строка_Результат);

Появляется окно диалога с заголовком, двумя кнопками OK и Cancel, строкой ввода и подсказкой над ней. Параметр Заголовок содержит заголовок окна диалога. Параметр Подсказка содержит строку над строкой ввода, поясняющая пользователю его действия. Параметр По_умолчанию задает строку, которая загружается в строку ввода при появлении окна диалога. Функции возвращают в Результат введенную строку (тип String). При нажатии клавиши Esc функция возвращает в Результат строку По_умолчанию. Команда inputquery возвращает true, если пользователь нажал кнопку Ok (или клавишу Enter), или false, если нажал кнопку Cancel (или клавишу Esc). Результат ввода записывается в третий строковый параметр Строка_Результат.

Задание на практическую работу

1. Расположить на форме: панель, которая должна занимать большую часть формы – «Минное поле»; на ней кнопку размером 50x50 с картинкой мины; в свободном пространстве формы четыре метки с именами: labelx, labely, labeld, Labels (свойство Name) для отображения текущего положения на поле.

2. Описать две глобальные целые переменные d_old (предыдущее расстояние до кнопки) и d_new (новое расстояние до кнопки).

3. Создать функцию rast() для класса формы tform1, в которой вычисляется расстояние между серединой кнопки и текущим положением курсора мыши. Прототип функции задать в файле .h в описании класса формы раздела public. Описание и вызов член-функции разместить в файле .cpp. Как параметры функции передаются координаты мыши, возвращается вычисленное расстояние. Функции может выглядеть следующим образом

```
Double tform1::rast (int x, int y)
{       int xc, yc;
Xc=Form1->Button1->Left+Form1->Button1->Width/2;
Yc=Form1->Button1->Top+Form1->Button1->Height/2;
Return (sqrt(pow((x-xc),2)+pow((y-yc),2)));
}
```

4. В событии onshow для формы организовать работу с паролем. Считать подготовленный заранее пароль из текстового файла «Key.txt» (создается заранее в Блокноте или из среды Builder командой File\New\Other\Text). Работу с паролем оформить через команды диалога ввода-вывода.

5. В событии oncreate для формы реализовать действия:

- запустить датчик случайных чисел;
- задать случайное положение кнопки (свойства Left, Top) в диапазоне размера панели минус ширина кнопки;
- сделать кнопку невидимой (свойство visible);
- задать начальное значение переменной d_old, равное ширине панели.

6. В событии onmousemove для панели реализовать действия:

- вывести в надписи labelx и labely текущее положение курсора мыши на панели;
- в надпись labeld разместить значение переменной d_new, полученное с помощью функции вычисления расстояния;
- сравнить два расстояния d_new и d_old и вывести сообщение «Ближе» или «Дальше» в компонент надпись Labels;
- перезаписать в значение переменной d_old текущее расстояние d_new для следующих перемещений и сравнений;
- если курсор мыши, вблизи кнопки, то сделать кнопку видимой.

7. Щелчком по кнопке (событие onclick кнопки) сделать кнопку невидимой, задав ей новые случайные координаты.

8. После каждого обнаружения кнопки («Мины»), уменьшать ее размер на 10 пикселей. Осуществить данное действие в событии onclick формы.

9. По желанию пользователя сменить пароль, перезаписав его в текстовый файл. Осуществить данное действие в событии onclose формы.

10. Добавить счетчик количества попыток обнаружения кнопки. Проверять его с заданным общим количеством попыток. Выводить на метку информацию об оставшихся количествах попыток.

11. Добавить компонент Timer. Использовать его для задания времени игрового процесса.

12. В событии ontimer выводить на метку количество оставшегося игрового времени.

13. Остановить игровой процесс в случае определения победителя. Выигрывает пользователь в случае обнаружения десяти «Мин», выигрывает ПК, если время игры истекло, а все «Мины» не найдены.

Практическая работа №25 «Разработка функциональной схемы работы приложения»

Цель работы: Получение навыков работы в визуальной среде программирования. Выполнения основных этапов разработки приложений. Изучение возможностей окна Инспектора объектов для задания свойств форме на этапе проектирования приложения. Знакомство со способами написания обработчиков событий, назначением и типами параметром. Изучение возможностей окна Инспектора объектов для задания шаблонов обработчиков событий формы. Способы вызова событий.

Этапы разработки проложений

Процесс разработки приложений в C++ Builder содержит следующие этапы:

1. Построение визуального интерфейса: выбор нужных компонентов и размещение их на форме (кнопки, меню и т.д.).

2. Установка с помощью она Инспектора объектов свойств формы и ее элементов управления.

3. Присоединение кода на языке C++ к компонентам для обработки событий, возникающих от пользователя (с помощью мыши или клавиатуры) или системы (таймер и др.).

4. Компиляция исходного кода C++ и ресурсов формы в исполнимый exe -файл, который может запускаться из среды C++ Builder или из операционной системы Windows как отдельное приложение.

Форма

Одним из важнейших компонентов C++ Builder является форма. **Визуально форма** - это специальное окно, которое составляет основу приложения Buider, и является контейнером для других компонентов. Типичная форма представляет собой прямоугольное окно и может содержать следующие элементы: рамку, заголовок, главное меню под заголовком, строку состояния в нижней части, вертикальную и горизонтальную полосы прокрутки.

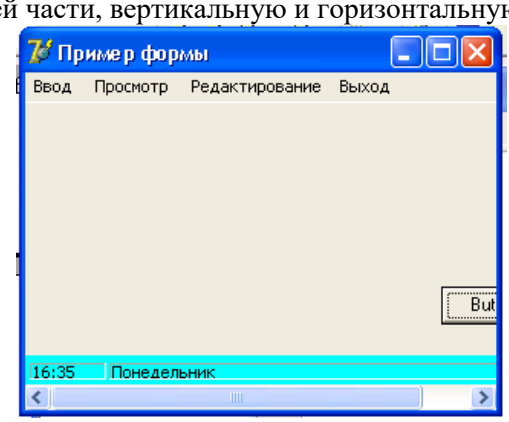


Рисунок 25.1. Вид формы с основными элементами

Все составляющие формы можно менять с помощью свойств формы. Остальная часть пространства формы называется клиентской областью. В ней размещаются различные элементы управления, выводиться текст и графика и т.д.

Программно форма - это объект стандартного класса tform. Как любой объект форма имеет свои свойства, которые можно задавать как на этапе проектирования приложения с помощью окна Инспектора объектов, так и на этапе выполнения приложения из текста программы.

Основные свойства формы

Alphablend – разрешает прозрачность формы. Если свойство равно true, то форма может быть прозрачна.

Alphablendvalue – задает степень прозрачности формы от 0 до 255. Значение 0 свойства означает полностью прозрачна; 255 – полную непрозрачность; полупрозрачная – промежуточное значение.

Autoscroll – простое логическое – определяет, будут ли автоматически появляться полосы прокрутки, если при заданном размере окна не все компоненты помещаются на нем. Если true (по умолчанию), то будут.

Bordericons – задает набор кнопок в правом углу заголовка.

Borderstyle - определяет внешний вид и поведение рамки окна формы.

Borderwidth – ширина рамки.

Caption – задает заголовок окна.

Clientheight, clientwidth – определяет высоту и ширину в пикселях клиентской области формы.

Color – цвет формы.

Icon – определяет пиктограмму, отображаемую в заголовке окна формы.

Left, Top – определяют в пикселях координаты левого угла формы относительно экрана монитора.

Windowstate - состояние формы после запуска.

События

Событие – это действие со стороны пользователя или самой системы относительно запущенного на выполнение приложения. С помощью событий происходит взаимодействие пользователя и системы с запущенным приложением. Метод создания программ на основе возникающих событий, называется **событийно-ориентированным**. **Обработчиком события** - это функция, содержащая команды на языке программирвоания C++, которые должны выполниться при возникновении события, с которым обработчик связан.

Создание обработчика главного события

У каждого компонента есть **главное событие** – событие, которое с этим компонентом происходит обязательно или наиболее часто. Например, для формы – это событие **oncreate** – создание формы, для кнопки – это onclick – щелчок по кнопке и т.д. Для создания обработчика главного события надо:

- активизировать компонент на форме, для которого создается обработчик;
- выполнить двойной щелчок по компоненту.

Среда C++ Builder подготовит заготовку будущего события.

Создадим обработчик главного события формы – oncreate.

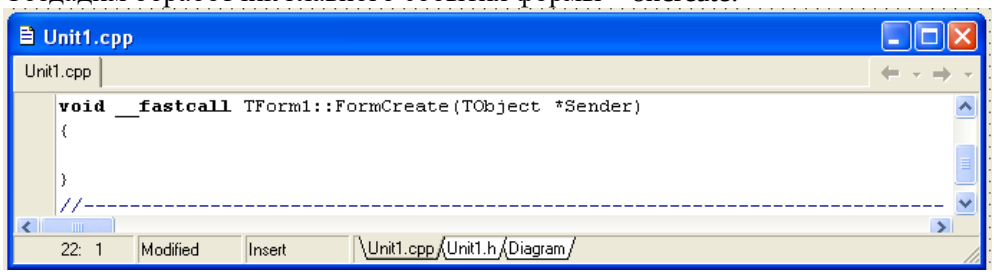


Рисунок 25.2. Окно текста программы с заготовкой главного события формы

Обработчик имеет имя formcreate, один параметр Sender типа *tobject, нет возвращаемого значения, обработчик принадлежит классу tform1. Курсор установиться между {}, где следует написать команды обработчика.

Создание обработчика произвольного события

Для создания заготовки любого события используется окно Инспектора объектов. Требуется выполнить следующие действия:

1. Активизируется компонент на форме, для которого пишется обработчик.
2. В окне Инспекторе объектов перейти на вкладку Events (События). На ней расположен список событий, распознаваемых выбранным компонентом.
3. Найти в списке имя нужного события.
4. Выполнить двойной щелчок в пустой строке справа от события.
5. Из окна Инспектора объектов система вернется в окно Редактора, в котором в позиции курсора вписываются команды события.

Для случая создания шаблона обработчика события щелчка по форме:

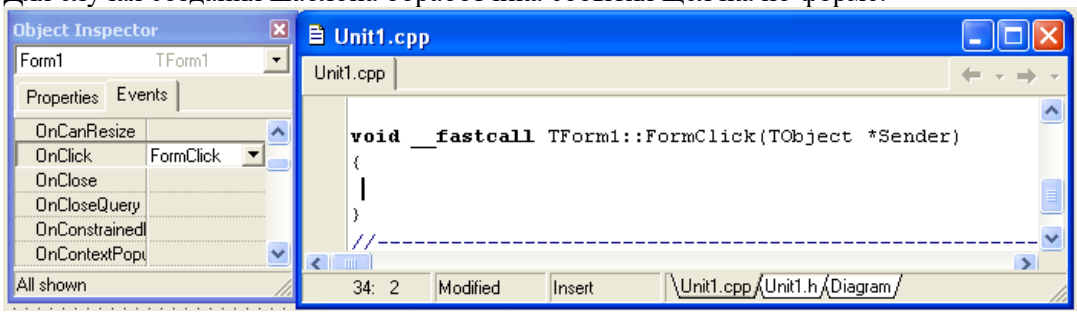


Рисунок 25.3. Создание шаблона произвольного обработчика события

Задание на практическую работу

Необходимо освоить основные свойства формы и способы задания им значений в требуемом диапазоне, а так же освоить основные приемы создания обработчиков событий формы. Для этого надо создать отдельную папку, в которую будут помещаться все файлы, относящиеся к одному приложения. Установить следующие свойства формы с использованием окна Инспектора объектов:

1. Выравнивание формы различными вариантами (свойство Align);
2. Оформить рамку формы различными вариантами (свойство borderstyte);
3. Задать заголовок окна формы (свойству Caption присвоит значение «Моя первая форма»);
4. Подобрать цвет формы (свойство Color);
5. Выбрать вид указателя формы из списка стандартных указателей (свойство Cursor);
6. Поэкспериментировать со свойством прозрачности формы (свойства alphablend и alphablendvalue);
7. Изображение всплывающей подсказки формы (в свойство Hint записать текст подсказки – «Это форма», в свойство showhint записать значение true);
8. Указать состояние формы (свойство windowstaty);
9. Создать иконку (значок) для формы. Для этого надо выполнить два этапа:

Создание иконки. Для этого требуется выполнить:

- вызвать графический редактор командой Tools\Image Iditor.
- выбрать в нем пункт меню File\New\Icon File(ico);
- задать свойства иконки (размер 32x32, цвет=16);
- рисовать иконку по пикселям;
- сохранить файл иконки в папке со всеми файлами данного проекта командой File\Save As…;

-выйти из Image Editor.

Подключение иконки к форме. Для этого надо выполнить:

- выбрать свойство Icon формы, дважды щелкаем напротив этого свойства на кнопке […]. При этом появляется окно Picture Editor;
- щелкнуть по кнопке Load…. Появиться окно Load Picture. Выбрать в нем нужную иконку;
- щелкнуть по кнопке «Открыть» в окне «Load Picture». При этом происходит возврат в окно Picture Editor, в котором высвечивается нужная иконка;
- щелкнуть по кнопке Ок. Нужная иконка появиться в заголовке.

Сохранить все файлы проекта командой меню File\Save All.

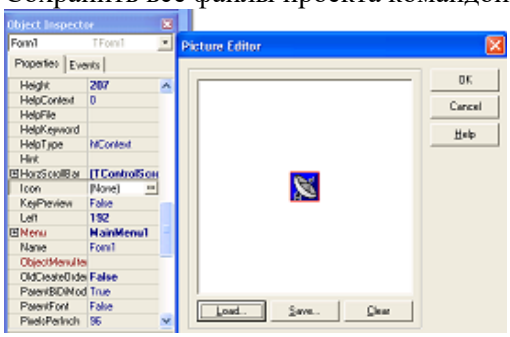


Рисунок 25.4. Задание иконки для формы

Выполните самостоятельно

1. В событии onmousedown левая кнопка мыши случайно меняет цвет формы, правая кнопка мыши сворачивает окна приложения до иконки.
2. Событие ondblclick вызывает закрытие формы методом Close, а так как форма является главной, то ее закрытие завершает работу всего приложения.
3. Событие onmousemove выводит текущие координаты мыши в заголовке формы.
4. В событии oncreate активизировать датчик случайных чисел.
5. В событии onkeydown при нажатии клавиш [Alt]+[X] осуществить выход из приложения.
6. Событие onkeydown анализирует код нажатой клавиши и выполняет необходимые действия для формы в зависимости от нажатой клавиши, а именно:

- при нажатии клавиш управления курсором форма должна перемещаться в указанном направлении (свойства Left и Top для формы);

- при одновременном нажатии клавиши [Shift] и клавиш управления курсором форма должна увеличиваться в размерах в нужном направлении (свойства Left, Top, Width, Height для формы);

- при нажатии клавиши [Ctrl] и клавиш управления курсором, происходит уменьшении размер формы в нужном направлении.

7. События onmousewheeldown и onmousewheelup меняют прозрачность формы (свойства alphablend и alphablendvalue для формы).

Практическая работа №26 «Разработка оконного приложения с несколькими формами»

Цель работы: Закрепление на конкретных примерах полученных теоретических знаний при изучении свойств, методов и событий стандартных компонентов cspinedit и cgauge. Создание на основе данных компонентов развлекательного приложения «Тренажер памяти»

Компонент счетчик - cspinedit

Компонент cspinedit используется для увеличения или уменьшения некоторой целочисленной величины – счетчика. Представляет собой пару кнопок с треугольниками и полем редактора, в котором может выводиться текущее значение счетчика. Компонент расположен на вкладке Samples. Основными свойства компонентами cspinedit являются:

Increment – шаг изменения счетчика (по умолчанию равно 1).

Maxvalue, minvalue – допустимые границы для индекса. По умолчанию равно 0, что означает отсутствие ограничений.

Value – текущее значение счетчика. По умолчанию равно 0.

Основное событие компонента – это onchange – возникает при изменении значения счетчика, т.е. При щелчках на кнопках со стрелками или программно при изменении свойства Value. В событии можно получить значение свойства Value.

Компонент индикатор хода процесса - cgauge

Компонент может использоваться для отображения хода длительного процесса. Основные свойства компонента:

Progress – значение величины, отображаемое индикатором (счетчик выполненных действий).

Maxvalue, minvalue – диапазон изменение отображаемой величины. По умолчанию свойства имеют значения 100 и 0 соответственно.

Color, backcolor, forecolor – задает цвет панели компонента, цвет фона графика, цвет указателя текущего значения величины.

Showtext – разрешает отображение текста на панели.

Kind – определяет тип индикатора.

Задание на практическую работу

Разработать приложение «Тренажер памяти» с использованием компонента счетчика cspinedit и индикатора cgauge. На форму выводится число, состоящее из заданного количества цифр, и задерживается на заданное количество секунд. Пользователь должен запомнить это число и правильно его ввести. Проводится ряд таких опросов. В конце теста делается заключение о качестве Вашей памяти.

На форме расположить:

- три компонента cspinedit для задания количества опросов, количества цифр в числе и временного интервала в секундах для запоминания числа;
- метку Label для вывода на ней сформированного датчиком случайных чисел числа с указанным количеством цифр;
- компонент Edit для ввода пользователем отгадываемого значения;
- невизуальный компонент Timer для времени отображения числа;
- снабдить форму необходимым количеством кнопок.

Описать глобальные переменные:

- целые: для количества опросов, цифр в числе, временной интервал, количество выполненных опросов, количество верных и неверных ответов.
- строковые: отгадываемое число и число, вводимое пользователем.

Создать программный код событий:

- oncreate для формы, в котором задать значения по умолчанию для тренажера; запустить датчик; очистить компоненты Label и Edit;
- ontimer для таймера, в котором скрывать случайное число в Label;
- onkeypress для компонента Edit, в котором при нажатии на клавишу [Enter] проводить сверку введенного пользователем числа в Edit и числа, скрытом в Label.

Использовать два компонента индикатора процесса cgauge: один для отображения количества совершенных попыток и другой – для отображения количества правильно выполненных заданий на текущий момент.

1. При запуске приложения выводится окна с приглашением ввода имени тестируемого.

2. После прохождения сеанса тестирования в тестовом файле сохраняется информация в виде: имя тестируемого и процент качества его памяти.

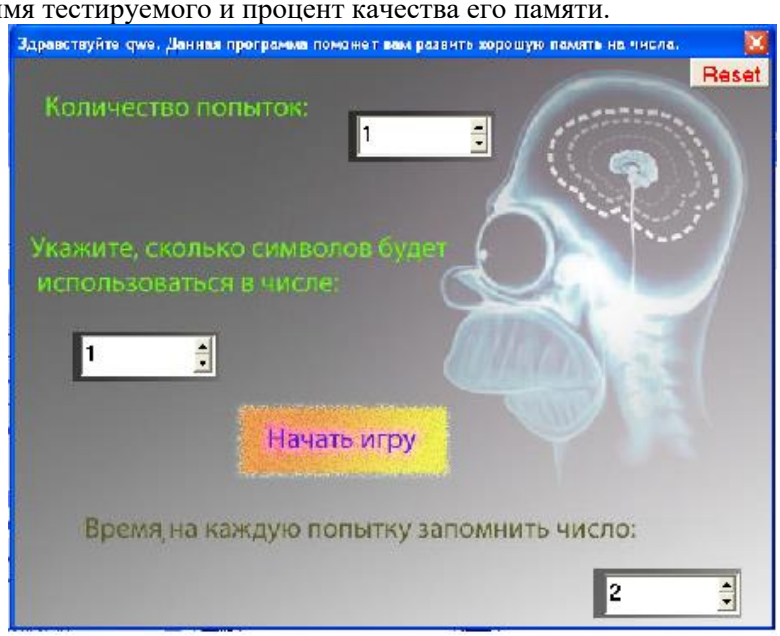


Рисунок 26.1. Приложение «Тренажер памяти»

Практическая работа №27 «Разработка игрового приложения»

Цель работы: Создать динамическое приложение на C#. Разработать игру «Тараканьи бега»

Используя уже известный объект picturebox, создадим простенькую игру "Тараканьи бега". Суть игры состоит в том, чей жучёк прибежит к финишу первым. Для игры нужны две картинки жучков, один будет красный, другой синий. Картинки можно нарисовать в редакторе Paint, или загрузить к себе в папку с проектом представленные здесь

Картинки:

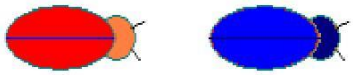


Рисунок 27.1. Изображение объектов «Тараканы»

Создав новый проект, следует расположить на форме следующие объекты:

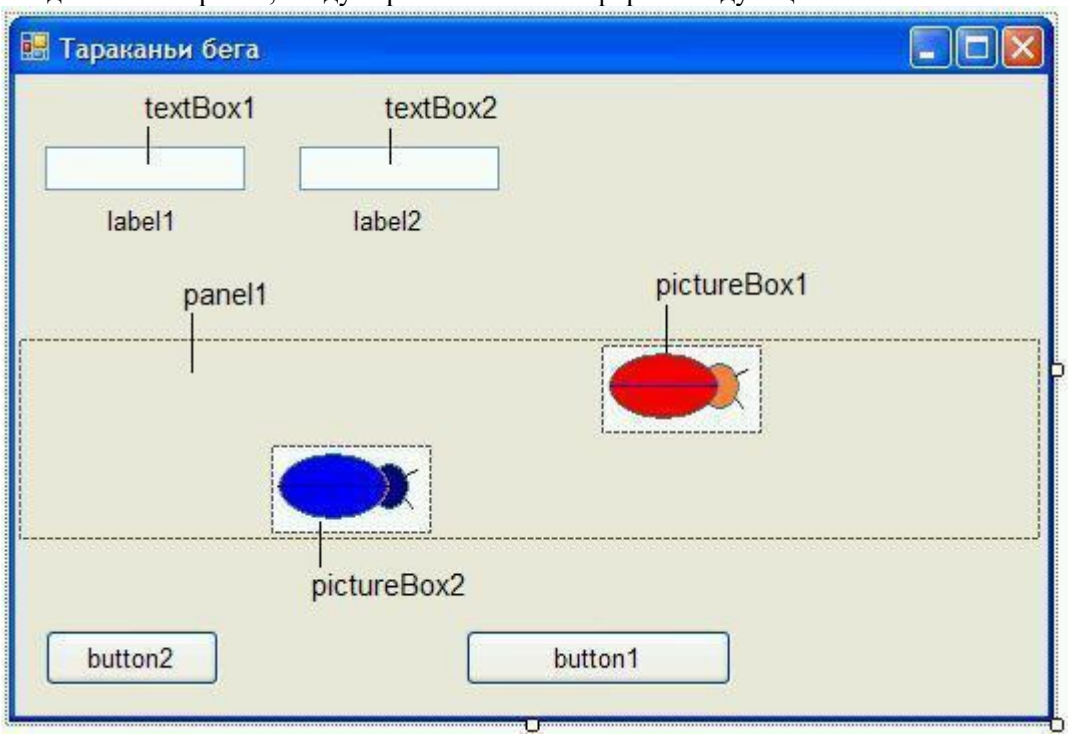


Рисунок 27.2. Форма приложения

В Объекты picturebox1 и picturebox2 следует загрузить картинки. Для этого в свойстве Image нажать маленькую кнопочку и, далее, нажать кнопку Import на панели ресурсов.

Когда картинки будут загружены, следует панели panel1 присвоить в свойстве backcolor такой же цвет, как и фон картинок жучков.

Двойным щелчком на выделенной форме следует создать событие и прописать в нём начальные установки нашей программы:

```
Private void Form1_Load(object sender, eventargs e)
{
Textbox1.Text = "Вася";
Textbox2.Text = "Петя";
Label1.Text = "0";
Label2.Text = "0";
Picturebox1.Left = 1;
Picturebox2.Left = 1;
}
```

Кнопке button1 в свойстве Text следует присвоить значение "СТАРТ", а button2 значение "Сброс".

Запустим приложение на исполнение; форма должна принять вид, показанный на рисунке:

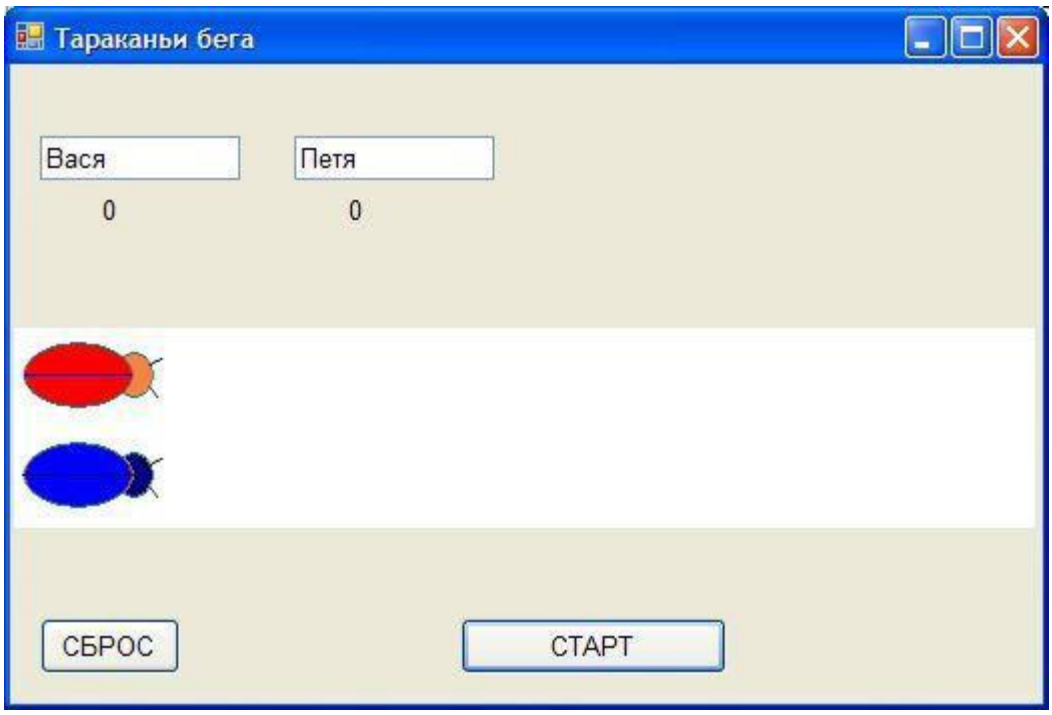


Рисунок 27.3. Игровая консоль приложения

Далее, создадим в коде программы несколько глобальных переменных:

```
Public partial class Form1 : Form
{
Int flag1;
Int x1, x2;
Public Form1()
{
...
```

Переменная flag1 - предназначена для контроля игровой ситуации, если flag1 равна 0, то жучки стоят на месте, а если flag1 не равна 0, то жучки начинают двигаться. Переменные x1 и x2 предназначены для хранения горизонтальных координат положения жучков на беговой дорожке.

Создадим событие для кнопки "Сброс", это событие очищает все переменные и устанавливает жучков в стартовую позицию.

```
Private void button2_Click(object sender, eventargs e)
{
Label1.Text = "0";
Label2.Text = "0";
X1 = 1; x2 = 1;
Picturebox1.Left = x1;
Picturebox2.Left = x2;
Flag1 = 0;
}
```

Для кнопки "СТАРТ" тоже создадим событие, оно будет очень простое, нужно присвоить переменной flag1 любое значение не равное 0.

```
Private void button1_Click(object sender, eventargs e)
{
Flag1 = 1;
}
```

Теперь осталось создать событие, которое будет перемещать наших жучков. Для этого нам понадобится компонент таймер timer1. Установим компонент на форме и произведём некоторые настройки.

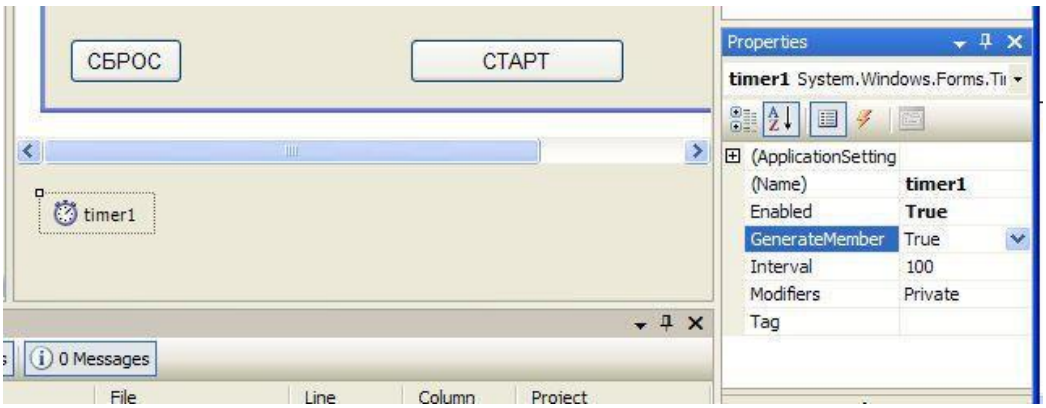


Рисунок 27.4. Таймер

Свойство generatemember и Enabled должны быть True. Свойство Interval равно 100 - это один тик таймера в секунду. Если жучки будут двигаться медленно, то это значение можно уменьшить, например до 50.

Для определения координаты финиша следует одного из жучков сдвинуть до конца вправо.

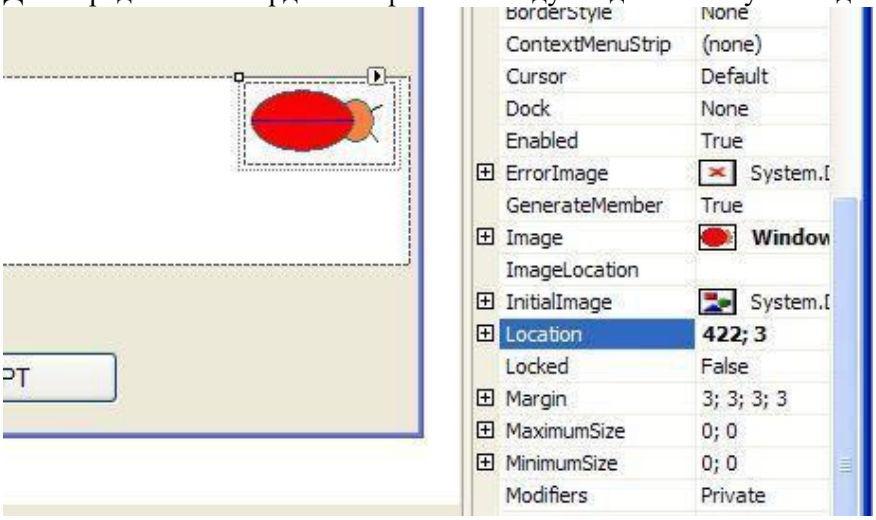


Рисунок 27.5. Настройна координат

Первая цифра в свойстве Location означает финишную координату (у вас это значение может быть другим, его нужно запомнить для вставки в код программы).

Двойным щелчком по иконке timer1 создадим событие движения жучков:

```
Private void timer1_Tick(object sender, eventargs e)
{
If (flag1 != 0) // Если дан старт :
{
Random a = new Random(); // Включаем генератор случайных чисел
//    В переменную count записываем случайное число в диапазоне от 0 до 8
Int count = a.Next(8);
//    Наращиваем значение координаты x1 на случайное число count
X1 += count;
// Выводим значение пройденного пути для первого игрока label1.Text = Convert.tostring(x1);
//    Смещаем первого жучка на случайную величину picturebox1.Left = x1;

//    Создаём случайное число для второго игрока и повторяем всё то же самое
Count = a.Next(8); x2 += count;
Label2.Text = Convert.tostring(x2); picturebox2.Left = x2;
}
//    Проверяем какой из игроков дошёл до финиша и останавливаем процесс
If ((x1 >= 422) || (x2 >= 422))

{
```

```
Flag1 = 0;
}
}
```

Запустим программу на исполнения и, нажав кнопку "СТАРТ" сыграем несколько партий, что бы убедиться, что игроки приходят к финишу с переменным успехом.

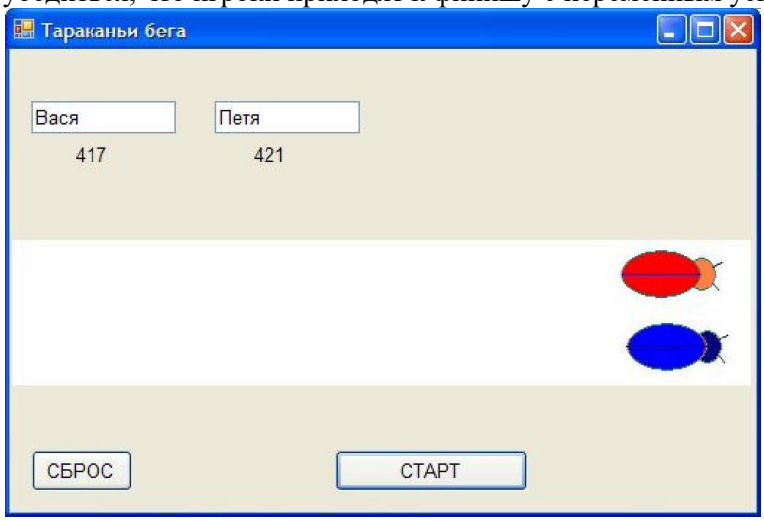


Рисунок 27.6. Готовое приложение

В Заключение следует сказать, что исполнимый файл (exe) находится в папке с вашим проектом в папке bin \ debug \ имя_проекта.exe. Этот файл можно копировать на любой носитель и выполнять независимо от среды программирования Visual Studio.

Практическая работа №28 «Создание процедур обработки событий. Компиляция и запуск приложения»

Цель работы: Закрепление на конкретных примерах полученных теоретических знаний при изучении возможностей компонента Shape. Освоение технологии работы с инструментом «Кисть» и «Карандаш». Заполнение замкнутых областей масками заполнения.

Спецификация компонента фигура (Shape)

Компонент Shape используется для отображения на форме различных геометрических фигур. Размер и положение фигуры на форме указывается с помощью свойств Height, Width, Left, Top. Пользователь не может сам изменять размеры и положение фигуры. Однако, используя события onmouseup и onmousedown для фигуры, можно запрограммировать необходимые действия. Находится на вкладке Additional.

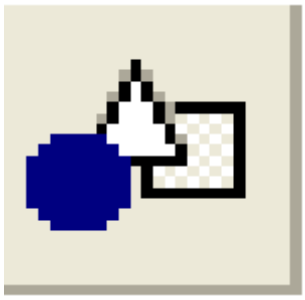


Рисунок 28.1. Компонент Shape

Перечислим основные свойства компонента Shape.

Shape – задает нужную фигуру для рисования. Принимает значения:

Stcircle - круг;

Stellipse - эллипс;

Strectangle - прямоугольник;

Stroundrect - прямоугольник со скругленными углами;

Stroundsquare - квадрат со скругленными углами;

Stsquare - квадрат.

Pen – задает атрибуты карандаша для контура фигуры. Является объектным свойством и имеет ряд настраиваемых атрибутов:

Tcolor Color - цвет линии, вычерчиваемой карандашом.

Style - вид (стиль) линии; может принимать значения:

Pssolid - сплошная линия;

Psdash - пунктирная линия, длинные штрихи;

Psdot - пунктирная линия, короткие штрихи;

Psdashdot – чередование длинного и короткого штрихов;

Psdashdotdot – чередование одного длинного и двух коротких штрихов;

Psclear - линия не отображается.

Int Width - толщина линии (в пикселях) только для сплошного стиля.

Brush – задает атрибуты кисти для заливки замкнутых областей. Свойство является объектым типом.

Tcolor Color - цвет закрашивания замкнутой области;

Style – стиль заполнения области. Может принимать:

Bssolid - сплошная заливка;

Bsclear - область не закрашивается;

Bshorizontal - горизонтальная штриховка;

Bsvertical - вертикальная штриховка;

Bsfdiagonal - диагональная штриховка с наклоном линий вперед;

Bsbdiagonal - диагональная штриховка с наклоном линий назад;

Bscross - горизонтально-вертикальная штриховка, в клетку;

Bsdiagcross - диагональная штриховка, в клетку.

События от мыши для перетаскивания компонентов

Onmousedown – нажатие кнопки мыши (начало перетаскивания);

Onmousemove – перемещение мыши (само перетаскивание)

Onmouseup – отпускание кнопки мыши (завершение перетаскивания).

Обозначим (x0, y0) – старое положение компонента, (X, Y) – текущее положение курсора мыши (параметры события). Пересчет нового положения объекта происходит по формулам:

Объект->Left = объект->Left + (X - x0);

Объект->Top = объект->Top + (Y - y0);

Потребуются следующие глобальные переменные:

```
Int     x0, y0;         //Координаты начала перетаскивания
Bool            drag;           //Подтверждение перетаскивания:
//true – перемещение с нажатой кнопкой мыши
//false – перемещение без нажатой кнопки
```

Для перетаскиваемого компонента требуется написать обработчик события onmousedown, в котором требуется выполнить следующие действия:

```
X0=X;
Y0=Y            //координаты мыши в момент нажатия
Drag=true;              //сигнализация о перетаскивании
```

При перетаскивании возникает событие onmousemove для перетаскиваемого компонента, в котором вычисляется новое положение объекта по предложенным выше формулам при условии, что перемещение происходит с нажатой кнопкой.

```
If  (drag==true)        //Нажатая кнопка мыши
{       //Формулы пересчета  }
```

Перетаскивание завершается при отпускании кнопки мыши возникает событие onmouseup для перетаскиваемого объекта, в котором надо выключить флажок перетаскивания:

```
drag=false;
```

Пример программы с применением механизма перетаскивания

Перетаскивание фигуры осуществляется левой кнопкой мыши, изменение размера фигуры – правой кнопкой мыши.

```
Int x0, y0, kn;
Bool drag;
Void __fastcall tform1::Shape1MouseDown (tobject *Sender,
                        Tmousebutton Button, tshiftstate Shift, int X, int Y)
{
X0=X;
Y0=Y;                                           //Координаты начала перетаскивания
Drag=true;                                              //Фиксация перетаскивания
If (Button==mbleft)     kn=1;           //Левая или правая
Else    kn=2;                           //Правая
}
//---------------------------------------------------------------------------
Void __fastcall tform1::Shape1MouseMove
                (tobject *Sender, tshiftstate Shift, int X, int Y)
{
If (drag==true)                         //Нажата кнопка мыши
        If (kn==1)                      //Для левой кнопки - перетаскивание
                {
                Form1->Shape1->Left=Form1->Shape1->Left+(X-x0);
                Form1->Shape1->Top=Form1->Shape1->Top+(Y-y0);
                }
        Else                            //Для правой кнопки – изменение размера
                {
                Form1->Shape1->Width=Form1->Shape1->Width+(X-x0);
                Form1->Shape1->Height=Form1->Shape1->Height+(Y-y0);
                X0=X;
                Y0=Y;
                }
}
//---------------------------------------------------------------------------
Void __fastcall tform1::Shape1MouseUp (tobject *Sender,
                        Tmousebutton Button,  tshiftstate Shift, int X, int Y)
```

```
{
Drag=false;                    //Отключить перемещение
}
```

Задание на практическую работу

Разработать приложение с использованием компонентов: компонента Shape и других необходимых элементов интерфейса для создания приложения, выполняющего операции с графической фигурой. Приложение должно настраивать отдельные графической фигуры, задавать свойства кисти и карандаша. Осуществить перетаскивание фигуры и изменение ее размера с помощью событий от мыши.

На форме расположить компоненты: Shap – основной элемент интерфейса, необходимое количество дополнительных элементов интерфейса, а так же восемь маленьких фигур (Shap) для задания маски плоскости фигуры.

1. Тип фигуры (свойство Shape) выбирать из выпадающего списка.
2. Левой кнопкой мыши выполнять перемещение фигуры.
3. Правой кнопкой мыши выполнять изменение размера фигуры.

Настроить свойства карандаша для контура фигуры (цвет, толщину, стиль), используя для этого необходимые элементы интерфейса.

Настроить свойства кисти (цвет, маску, цвет маски). Маску выбирать из восьми маленьких фигур, заполненных стандартными масками. Для этого создать событие onmousedown, в котором заполняется плоскость основной фигуры выбранной маской.

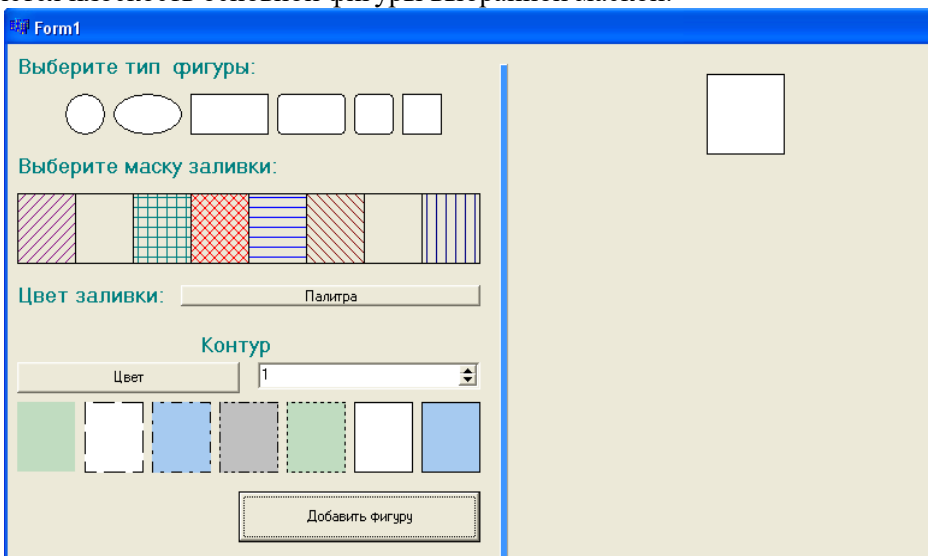


Рисунок 28.2. Фрагмент работы приложения «Редактор фигур»

Практическая работа №29 «Разработка интерфейса приложения»

Цель работы: Закрепить теоретические сведения о спецификации класса tcanvas, используемого для построения графических изображений по пикселам. Результатом практической работы должно стать прикладное приложение «Графические примитивы», которое выполняет построение графических фигур по заданным в программе условиям.

Назначение класса tcanvas

Ряд объектов (Form, Image, paintbox и др.) Имеют неопубликованное свойство Canvas (холст, поверхность), с помощью которого можно рисовать на поверхности этих компонентов. Свойство Canvas доступно только при выполнении приложения, следовательно, рисунки, полученные с помощью этого свойства, являются динамическими и существуют только в процессе выполнения приложения.

Холст состоит из отдельных пикселей. Левый верхний имеет координаты (0,0), а координаты правого нижнего пикселя зависят от размера холста, который можно получить, обратившись к свойствам Height и Width того объекта, на котором рисуется изображение. Для выполнения операций с холстом используется заголовочный файл Graphics.hpp.

Методы класса tcanvas позволяют выводить графические примитивы (точки, линии, окружности, прямоугольники и т.д.). Свойства класса tcanvas задают характеристики выводимых графических примитивов: цвет, толщину и стиль линий, цвет и маску заполнения областей, характеристики шрифта при выводе текстовой информации.

Свойства класса tcanvas

Cliprect – определяет доступную область для рисования и перерисовке при событии onpaint.

Pixels[int X][int Y] - двумерный массив цветов пикселей изображения. С помощью этого свойства можно получить доступ к любому пикселю картинки, чтобы прочитать его или изменить.

Penpos - определяет текущее положение пера.

Pen (карандаш) – определяет атрибуты пера для рисования линий, границ и геометрических фигур. Аналог свойства компонента Shape.

Brush (кисть) - используется для заливки (закрашивания) замкнутых областей, например геометрических фигур. Аналог свойства компонента Shape.

Font – устанавливает параметры шрифта для текста на поверхности рисования. Можно использовать стандартное диалоговое окно fontdialog.

Методы класса tcanvas

Void moveto(int x, int y) - перемещает графическое перо в новую пиксельную позицию (x, y). При перемещении пера рисование не происходит.

Void lineto(int x, int y) - вычерчивает прямую линию от текущей позиции пера до точки с координатами (x, y). После вычерчивания линии ее конец является текущей позицией пера.

Void Polyline(tpoint *m, int kol) - вычерчивает ломаную линию, последовательно соединяя kol точек, координаты которых находятся в массиве m.

Void Polygon (tpoint *m, int kol) – аналогично методу polyline, но вычерчивает закрашенный замкнутый многоугольник.

Void Rectangle(int x1, int y1, int x2, int y2) или void Rectangle(trect r) – вычерчивается прямоугольник с контуром и закрашенной областью.

Void fillrect(trect r) - вычерчивает закрашенный прямоугольник без контура. Используется объект Brush.

Void framerect(trect r) — только контур прямоугольника толщиной в один пиксель. Используется объект Brush для задания свойств линии.

Void roundrect(int x1, int y1, int x2, int y2, int x3, int y3) – прямоугольник со скругленными углами; параметры x3, y3 задают размер эллипса скругления.

Void Ellipse(int x1, int y1, int x2, int y2) или void Ellipse(trect r) - вычерчивает закрашенный эллипс или окружность. Параметры задают координаты прямоугольника, в который будет вписан эллипс.

Void Arc(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4) - вычерчивает дугу окружности или эллипса. Параметры x1, y1, x2, y2 задает координаты прямоугольника, в который вписывается дуга. Параметры x3, y3, x4, y4 задают начальную и конечную точку пересечения дуги эллипса с прямой, исходящей из центра эллипса.

Void Chord(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4) – рисует хорду, т.е. Часть эллипса, образованного дугой и прямой линией, соединяющей концы дуги. Параметры аналогичны методу Arc.

Void Pie(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4) - рисует сектор эллипса или круга. Параметры аналогичны методу Arc.

Void floodfill(x, y, col, f) – закрашивает текущей кистью замкнутую фигуру с контуром цвета col. Параметры x, y задают координату точки начала окрашивания; четвертый параметр – способ заливки. Может принимать значения:

Fsborder – залить всю область до границы с цветом col;

Fssurface – залить всю область до границы с любым цветом, кроме col.

Void textout(int x, int y, String s) – вывод строки s в позиции (x, y). Параметры шрифта задаются свойством Font объекта Canvas. Фон текста закрашивается текущим цветом кисти.

Int textheight(String s) и int textwidth(String s) – возвращают пиксельную высоту и ширину строки s с учетом установленного шрифта.

События графики

Событие **onpaint** имеют все компоненты, у которых есть свойство Canvas, кроме компонента Image. Именно в этом событии выполняют основную работу по выводу графики на поверхность компонента (Canvas). Событие происходит каждый раз, когда нужно перерисовать рисунок на холсте компонента. Перерисовка возникает, когда изменяется содержимое холста компонента (например, при анимации) или когда необходимо восстановить часть или весь компонент, если до этого он был перекрыт другими компонентами. Операционная система сообщает приложению о необходимости перерисовки путем вызова события onpaint. Вызвать событие onpaint можно и программным способом методом Repaint() или Invalidate() этого компонента. Для компонента Image данного события не требуется, т.к. В этом классе предусмотрены все действия по перерисовки изображения в случае его порчи.

Задание на практическую работу

Разработать приложение с использованием компонентов: paintbox, Timer и других необходимых элементов интерфейса для создания приложения, выполняющего построение графических примитивов с заданными условиями отображения. Для построения графических примитивов использовать спецификацию класса tcanvas компонента paintbox.

1. На форме разместить компонент paintbox, который должен занимать большую часть формы и являться областью отображения графических примитивов.

2. В выпадающем списке выбрать тип примитива: точки, линии, прямоугольники, круговые фигуры, дуги, хорды, сектора, шрифты. Выбрать нужный примитив из выпадающего списка.

3. Атрибуты для кажого примитива задаются случайными значениями с учетом возможного диапазона для данного свойства.

4. В контейнере radiogroup выбрать режим отображения примитивов: по одному, количество (указать нужное количество), бесконечно.

5. Обеспечить скорость смены рисования примитива, выбрав необходимое значение для скорости таймера.

6. Обеспечить работу кнопок: «Пуск», «Пауза», «Очистить».

7. Использовать четыре компонента paintbox одного размера.

8. Для каждого компонента выбрать изображаемые в нем графические примитивы с помощью компонента checklistbox.

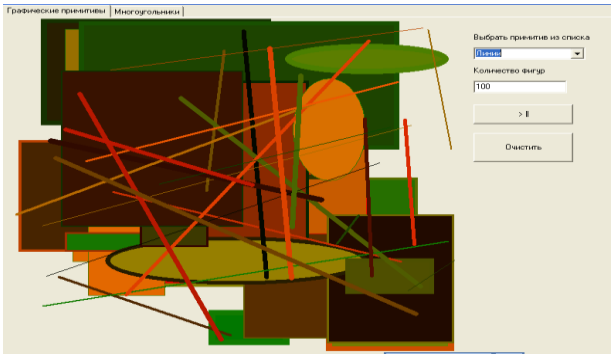


Рисунок 29.1. Фрагмент работы приложения «Графические примитивы»

Практическая работа №30 «Тестирование, отладка приложения»

Цель работы: Закрепить теоретические сведения о спецификации класса tcanvas, используемого для построения графических изображений по пикселам. Результатом практической работы должно стать прикладное приложение, выполняющее построения графическихе фигур на базе математических расчетныйх формул.

Назначение класса tcanvas

Ряд объектов (Form, Image, paintbox и др.) Имеют неопубликованное свойство Canvas (холст, поверхность), с помощью которого можно рисовать на поверхности этих компонентов. Свойство Canvas доступно только при выполнении приложения, следовательно, рисунки, полученные с помощью этого свойства, являются динамическими и существуют только в процессе выполнения приложения.

Холст состоит из отдельных пикселей. Левый верхний имеет координаты (0,0), а координаты правого нижнего пикселя зависят от размера холста, который можно получить, обратившись к свойствам Height и Width того объекта, на котором рисуется изображение. Для выполнения операций с холстом используется заголовочный файл Graphics.hpp.

Методы класса tcanvas позволяют выводить графические примитивы (точки, линии, окружности, прямоугольники и т.д.). Свойства класса tcanvas задают характеристики выводимых графических примитивов: цвет, толщину и стиль линий, цвет и маску заполнения областей, характеристики шрифта при выводе текстовой информации.

Свойства класса tcanvas

Trect cliprect – определяет доступную область для рисования и перерисовке при событии onpaint.

Tcolor Pixels[int X][int Y] - двумерный массив цветов пикселей изображения. С помощью этого свойства можно получить доступ к любому пикселю картинки, чтобы прочитать его или изменить.

Tpoint penpos - определяет текущее положение пера.

Pen (карандаш) – определяет атрибуты пера для рисования линий, границ и геометрических фигур. Аналог свойства компонента Shape.

Brush (кисть) - используется для заливки (закрашивания) замкнутых областей, например геометрических фигур. Аналог свойства компонента Shape.

Font – устанавливает параметры шрифта для текста на поверхности рисования. Можно использовать стандартное диалоговое окно fontdialog.

Методы класса tcanvas

Void moveto(int x, int y) - перемещает графическое перо в новую пиксельную позицию (x, y). При перемещении пера рисование не происходит.

Void lineto(int x, int y) - вычерчивает прямую линию от текущей позиции пера до точки с координатами (x, y). После вычерчивания линии ее конец является текущей позицией пера.

Void Polyline(tpoint *m, int kol) - вычерчивает ломаную линию, последовательно соединяя kol точек, координаты которых находятся в массиве m.

Void Polygon (tpoint *m, int kol) – аналогично методу polyline, но вычерчивает закрашенный замкнутый многоугольник.

Void Chord(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4) – рисует хорду, т.е. Часть эллипса, образованного дугой и прямой линией, соединяющей концы дуги. Параметры аналогичны методу Arc.

Void Pie(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4) - рисует сектор эллипса или круга. Параметры аналогичны методу Arc.

Формулы вращения

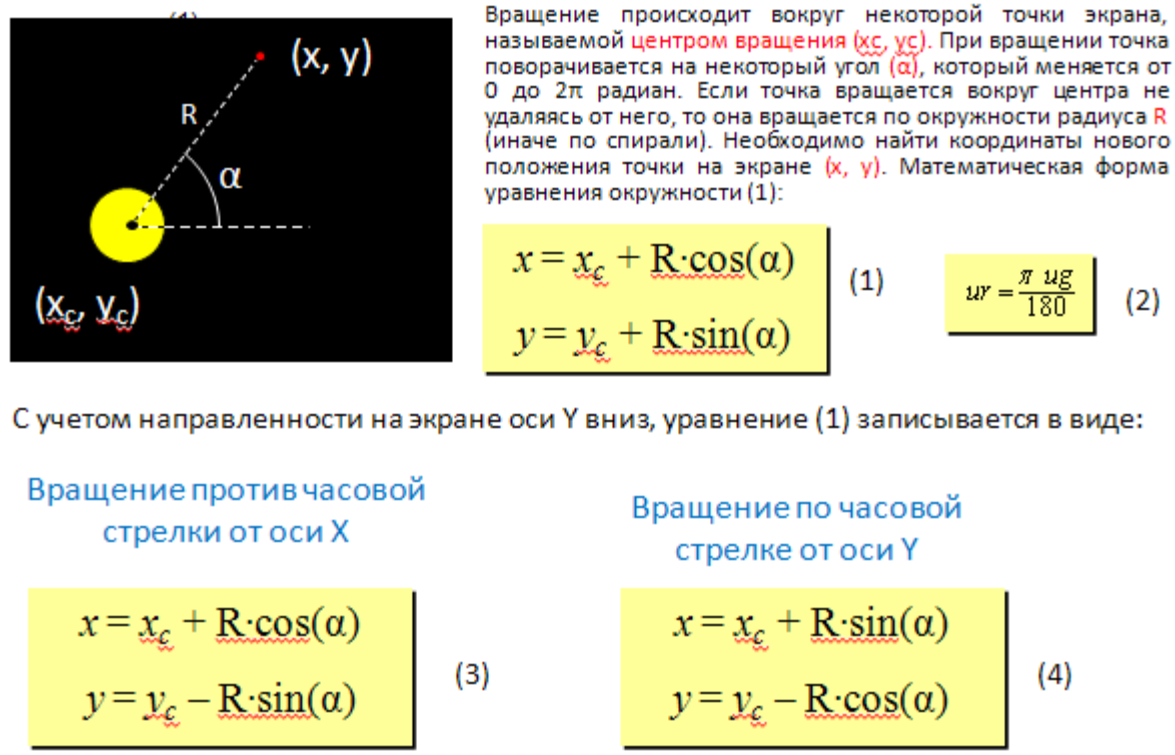


Рисунок 30.1. Формулы вращения

Пример программы с расчетными формулами для графических фигур

Программа строит правильный многоугольник путем рассчета координат вершин многоугольника по формулам вращения. Сединяет между собой построенные вершины линиями.

```
Void __fastcall tform1::Button3Click (tobject *Sender)
{
Paintbox2->Canvas->Pen->Color = paintbox2->Color;
Paintbox2->Canvas->Brush->Color = paintbox2->Color;
Paintbox2->Canvas->Rectangle (0,0,paintbox2->Width,paintbox2->Height);
Paintbox2->Canvas->Pen->Color = clblack;
N = strtoint(Edit2->Text);                  //Число вершин в многоугольнике
Int R = strtoint(Edit3->Text);    //Радиус многоугольника
P = new tpoint[N];                          //Массив вершие многоугольника
Int centerx = paintbox1->Width/2;           //Центр
Int centery = paintbox1->Height/2;
Float angle = M_PI*2/N;                     //Размер угла вершины  многоугольника
For(int i=0; i<N; i++)                      //Заполнение вершин значениями по формулам
{
P[i].x = centerx + R * cos (angle*i);
P[i].y = centery + R * sin (angle*i);
}
Paintbox2->Canvas->Polygon (p,N-1);  //Изобразить многоугольник
If(checkbox1->Checked)
{
For(int i=0; i<N; i++)                      //Соединить все вершины между собой
{
For(int j=0; j<N; j++)
{
```

```
Paintbox2->Canvas->Pen->Color = random (1000000000);
Paintbox2->Canvas->moveto (p[i].x,p[i].y);
Paintbox2->Canvas->lineto (p[j].x,p[j].y);
}
}
}
}
```

Практическая часть

Разработать приложение с использованием компонентов: paintbox и других необходимых элементов интерфейса для создания приложения, выполняющего построение графических фигур правильной формы с заданными условиями отображения и расчетными координатами вершин. Для построения графических примитивов использовать спецификацию класса tcanvas для компонента paintbox.

1. На форме разместить компонент paintbox, который должен занимать большую часть формы и являться областью отображения графических примитивов.
2. Задать количество вершин многоугольника.
3. Задать величину радиуса многоугольника.
4. Расчитать координаты вершин многоугольника по формулам вращения.
5. Построить многоугольник.
6. Установить флажок, с помощью которого задается режим соединения вершин правильного многоугольника.
7. Вводя коэффициент смещения, выполнить сдвиг вложенного многоугольника на заданное значение коэффициента.

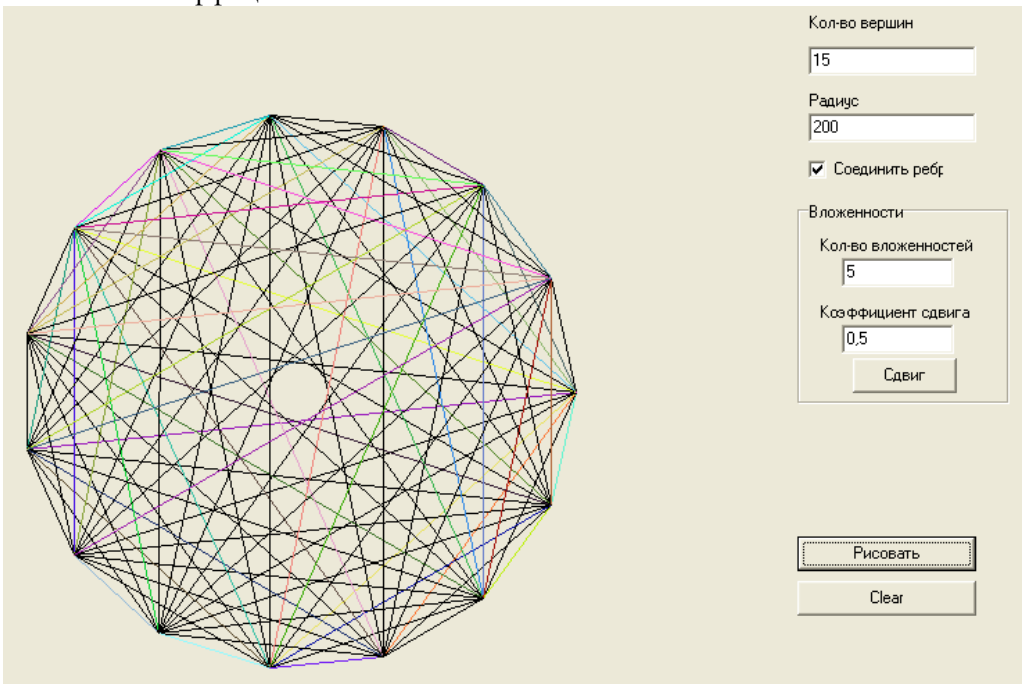


Рисунок 30.2. Фрагмент работы приложения «Построение геометрических фигур»

Практическая работа №31 «Программирование приложений»

Цель работы: Закрепить теоретические сведения о спецификации класса tcanvas, используемого для построения графических изображений по пикселам. Результатом практической работы должно стать прикладное приложение «Вращение планет», построенные на базе алгоритма анимации путем стерания движущегося объекта.

Назначение класса tcanvas

Ряд объектов (Form, Image, paintbox и др.) Имеют неопубликованное свойство Canvas (холст, поверхность), с помощью которого можно рисовать на поверхности этих компонентов. Свойство Canvas доступно только при выполнении приложения, следовательно, рисунки, полученные с помощью этого свойства, являются динамическими и существуют только в процессе выполнения приложения.

Холст состоит из отдельных пикселей. Левый верхний имеет координаты (0,0), а координаты правого нижнего пикселя зависят от размера холста, который можно получить, обратившись к свойствам Height и Width того объекта, на котором рисуется изображение. Для выполнения операций с холстом используется заголовочный файл Graphics.hpp.

Методы класса tcanvas позволяют выводить графические примитивы (точки, линии, окружности, прямоугольники и т.д.). Свойства класса tcanvas задают характеристики выводимых графических примитивов: цвет, толщину и стиль линий, цвет и маску заполнения областей, характеристики шрифта при выводе текстовой информации.

Свойства класса tcanvas

Trect cliprect – определяет доступную область для рисования и перерисовке при событии onpaint.

Tcolor Pixels[int X][int Y] - двумерный массив цветов пикселей изображения. С помощью этого свойства можно получить доступ к любому пикселю картинки, чтобы прочитать его или изменить.

Tpoint penpos - определяет текущее положение пера.

Pen (карандаш) – определяет атрибуты пера для рисования линий, границ и геометрических фигур. Аналог свойства компонента Shape.

Brush (кисть) - используется для заливки (закрашивания) замкнутых областей, например геометрических фигур. Аналог свойства компонента Shape.

Font – устанавливает параметры шрифта для текста на поверхности рисования. Можно использовать стандартное диалоговое окно fontdialog.

Методы класса tcanvas

Void moveto(int x, int y) - перемещает графическое перо в новую пиксельную позицию (x, y). При перемещении пера рисование не происходит.

Void lineto(int x, int y) - вычерчивает прямую линию от текущей позиции пера до точки с координатами (x, y). После вычерчивания линии ее конец является текущей позицией пера.

Void Polyline(tpoint *m, int kol) - вычерчивает ломаную линию, последовательно соединяя kol точек, координаты которых находятся в массиве m.

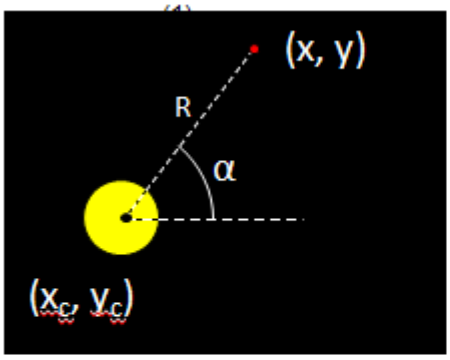
Void Polygon (tpoint *m, int kol) – аналогично методу polyline, но вычерчивает закрашенный замкнутый многоугольник.

Void Chord(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4) – рисует хорду, т.е. Часть эллипса, образованного дугой и прямой линией, соединяющей концы дуги. Параметры аналогичны методу Arc.

Void Pie(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4) - рисует сектор эллипса или круга. Параметры аналогичны методу Arc.

Формулы вращения

Формулы вращения



Вращение происходит вокруг некоторой точки экрана, называемой центром вращения (xc, yc). При вращении точка поворачивается на некоторый угол (α), который меняется от 0 до 2π радиан. Если точка вращается вокруг центра не удаляясь от него, то она вращается по окружности радиуса R (иначе по спирали). Необходимо найти координаты нового положения точки на экране (x, y). Математическая форма уравнения окружности (1):

$$x = x_c + R \cdot \cos(\alpha)$$
$$y = y_c + R \cdot \sin(\alpha) \quad (1)$$

$$ur = \frac{\pi \cdot ug}{180} \quad (2)$$

С учетом направленности на экране оси Y вниз, уравнение (1) записывается в виде:

Вращение против часовой стрелки от оси X

$$x = x_c + R \cdot \cos(\alpha)$$
$$y = y_c - R \cdot \sin(\alpha) \quad (3)$$

Вращение по часовой стрелке от оси Y

$$x = x_c + R \cdot \sin(\alpha)$$
$$y = y_c - R \cdot \cos(\alpha) \quad (4)$$

Рисунок 31.2. Формулы вращения

Пример программы по формулам вращения против часовой стрелке

Программа выполняет вращение «Луны» вокруг «Земли»

```
#define RZ 50          //Радиус Земли
#define RL 30          //Радиус Луны
#define RO 200         //Радиус орбиты вращения
Tform1 *Form1;
Int xc, yc, xl, yl;            //Центры Земли и луны
Float ug=0;                    //Угол вращения
//Событие прорисовки формы - начальные установки
Void __fastcall tform1::formpaint(tobject *Sender)
{
//Прорисовка Земли
Xc=Form1->Width/2                                   //Центр Земли
Yc=Form1->Height/2;
Form1->Canvas->Brush->Color=clyellow;          //Цвет Земли
Form1->Canvas->Ellipse(xc-RZ, yc-RZ, xc+RZ, yc+RZ);    //Земля
//Прорисовка Луны
Xl=xc+RO;                          /                /Центр Луны на оси X
Yl=yc;
Form1->Canvas->Pen->Color=Form1->Color;        //Контур Луны
Form1->Canvas->Brush->Color=clblue;                 //Цвет Луны
Form1->Canvas->Ellipse(xl-RL, yl-RL, xl+RL, yl+RL);    //Луна видна
Form1->Timer1->Enabled=true;                        //включить таймер
}
//Событие таймера - движение луны
Void __fastcall tform1::Timer1Timer(tobject *Sender)
{
//Стереть Луну в старой позиции
Form1->Canvas->Brush->Color=Form1->Color;
```

```
Form1->Canvas->Ellipse(xl-RL, yl-RL, xl+RL, yl+RL);
//Расчитать новое положение Луны
Ug=ug+M_PI/100;                          //Новый угол потороота
If (ug>=2*M_PI) ug=0;                    //Пройден весь круг
Xl=xc+RO*cos(ug);                        //Новая позиция Луны
Yl=yc-RO*sin(ug);
//Изобразить Луну на новом месте
Form1->Canvas->Brush->Color=clblue;      //Луна видна
Form1->Canvas->Ellipse(xl-RL, yl-RL, xl+RL, yl+RL);
}
```

Задание на практическую работу

Рисунок 31.3. Слайд с заданием на практическую часть

Список литературы

Основные источники

1. Голицына, О. Л. Основы алгоритмизации и программирования: учебное пособие / О.Л. Голицына, И.И. Попов. — 4-е изд., испр. и доп. — Москва: ФОРУМ: ИНФРА-М, 2021. — 431 с. — (Среднее профессиональное образование). - ISBN 978-5-00091-570-7. - Текст: электронный. - URL: https://znanium.com/catalog/product/1150328 (дата обращения: 01.06.2021). – Режим доступа: по подписке.
2. Колдаев, В. Д. Основы алгоритмизации и программирования: учебное пособие / В. Д. Колдаев; под ред. проф. Л. Г. Гагариной. — Москва: ФОРУМ: ИНФРА-М, 2021. — 414 с. — (Среднее профессиональное образование). - ISBN 978-5-8199-0733-7. - Текст: электронный. - URL: https://znanium.com/catalog/product/1151517 (дата обращения: 01.06.2021). – Режим доступа: по подписке.

Дополнительные источники

1. Трофимов, В. В. Основы алгоритмизации и программирования: учебник для среднего профессионального образования / В. В. Трофимов, Т. А. Павловская; под редакцией В. В. Трофимова. — Москва: Издательство Юрайт, 2021. — 137 с. — (Профессиональное образование). — ISBN 978-5-534-07321-8. — Текст: электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/473347 (дата обращения: 01.06.2021).

Интернет-источники

1. Электронная библиотечная система Znanium: сайт.- URL: https://znanium.com/ – Текст: электронный.
2. Электронная библиотечная система Юрайт: сайт. - URL: https://urait.ru/ -Текс: электронный.