**Лабораторная 4**

**Создание Windows – приложения**

**с обработкой строк**

**Задание.**

Создать программу из четырех диалоговых окон.

·        В первом окне выводятся различные элементы управления (RadioButton, CheckBox, MaskEdit – в соответствии с заданием);

·        Во втором окне происходит рисование с анимацией;

       В третьем окне происходит обработка строковых данных; в нем размещается поле ввода для ввода строки, ListBox для вывода элементов числового массива, поле для вывода результата обработки строки

Вводится строка, состоящая из слов, записанных через запятые, и заканчивающаяся точкой. ***Словом*** считать последовательность символов, не содержащую пробел. Выдать слова текста и их длины.

·        В четвертом окне происходит создание и наполнение таблицы из некоторых столбцов с подсчетом значения в одном из столбцов.

Наполнение этих окон - согласно варианту индивидуального задания.

**Индивидуальные варианты**.

**Варианты индивидуальных заданий**

Номер варианта имеет вид:

**U.V P L A T**

Назначение символов номера варианта:

**U** - **Вид элементов управления (для первого окна):**

**1. RadioButton**

**V** - **Что обозначают RadioButton:**

Пол ребенка

**P** - **Вид рисунка (для второго окна):**

Сова, машущая крыльями.

**L** - **Чем наполняется ListBox (в третьем окне):**

Получить массив кубов цифровых слов.

**A-обработка строки (в третьем окне)**

Сформировать новую строку, в которой слова будут следовать в обратном порядке.

**T – структура таблицы (четвертое окно):**

Информация о лекарствах (название лекарства, номер аптеки, цена упаковки, количество упаковок, стоимость данного лекарства – вычисляется)

**Отчет по лабораторной работе**

Отчет представляет собой таблицу вида:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя окна | Имя элемента контроля | Имя события | Словесное описание действия | Программный код |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Отчет нужно защищать, то есть объяснить программный код

**ОКОННЫЕ ПРИЛОЖЕНИЯ В C#**

# Отличия программирования, управляемого данными,

# от программирования, управляемого событиями

До сих пор рассматривались консольные приложения, которые управляются данными. В отличие от них Windows – приложение управляется событиями. На рис. показано работа приложения, управляемого данными, и приложения, управляемого событиями.



Рис. Схема консольного и Windows-приложения

То есть приложение, управляемое событиями, работает не последовательно, а как совокупность обработчиков событий. Событиями является, например, выбор пункта меню, нажатие кнопки, перемещение мыши и т.д.

# Формы и программные модули C#.

# Элементы управления в C#, их свойства и события

При создании проекта типа Windows Forms Application появляется форма, на которой можно размещать элементы управления, которые выбираются в окне Toolbox в левой части экрана.

У всех элементов управления или компонент есть свойства и события. Значения свойств можно менять как при проектировании приложения, так и при выполнении событий. Изменение свойства – это присвоение ему какого-то значения в виде оператора

Имя компоненты.имя свойства=значение.

Надо помнить, что всем компонентам при размещении на форму присваивается имя по умолчанию (имя компоненты с порядковым номером; это имя можно изменить при проектировании приложения).

Если значение свойства компоненты меняется в коде события этой же компоненты, то вместо имени компоненты пишется this.

Все визуальные компоненты имеют свойства Left,Top, Height,Width, определяющие положение компоненты на форме, Text – текст, расположенный на компоненте. Расположение компоненты в окне можно менять как визуально, так и изменением свойств в окне Properties. В этом же окне можно менять и другие свойства: цвет, текст, расположенный на кнопке и т.д.

Элемент управления **Label** предназначен для отображения информации.

Элемент управления **Button** (кнопка) имеет событие Click (Нажатие кнопки). При щелчке по кнопке появится заготовка фрагмента кода для обработки этого события. Вставка собственного кода (выделенного жирным шрифтом) приведет при исполнении приложения и нажатии на кнопку к выводу окна сообщения:

*private void buttonl\_Click(object sender, EventArgs e)*

*{****MessageBox. Show ("Первое Windows приложение!") ;*** *}*

Элемент управления **Text Box** позволяет вводить текст**.**

Элемент управления **MaskedTextBox** дополнительно к TextBox содержит свойство **Mask**, которое позволяет контролировать допустимость ввода символов (числа, даты и т.д.) .

Элементы управления **RadioButton** и **CheckButton** позволяют информировать о текущем состоянии приложения и изменять это состояние.

Традиционно элементы управления RadioButton (переключатели) следует применять, когда нужно предоставить возможность выбора между взаимоисключающими опциями — например, пола пользователя.

Чтобы сгруппировать переключатели для образования единого логического блока, следует использовать элемент управления **GroupBox**.

При первоначальном помещении элемента GroupBox на форму и последующем размещении необходимых элементов управления RadioButton внутри его границ, состояние элементов RadioButton будет автоматически изменено для отражения того, что только одна опция внутри групповой рамки может быть выбрана.

Элемент управления **CheckBox** (флажок) следует использовать, когда нужно предоставить возможность выбора одного или более режимов — например, для заполнения вопросника об использовавшихся ранее операционных системах (Windows ХР, Windows Vista, Linux и т.п.).

У того и другого элемента есть булевское свойство **Checked**, которое равно true, если переключатель или флажок включен, и false в противном случае.

Элемент управления **ListBox** позволяет отображать информацию в виде списков.Списки используются для отображения списков строк, из которых одновременно можно выбирать одну или более строк. Подобно флажкам и переключателям, списки предоставляют способ затребовать от пользователя выбор одного или более элементов. Список следует использовать в тех случаях, когда во время разработки неизвестно точное количество значений, из которых может осуществляться выбор (например, в списке сотрудников).

Элемент имеет свойства:

**SelectedIndex -** начинающийся с 0 номер выбранного элемента списка;

**Sorted -**  возможность упорядочения элементов списка по алфавиту;

**Items –** перечень строк, включенных в список. Его можно создавать как в момент проектирования, так и в процессе выполнения программы**.**

Для добавления в процессе выполнения нужно выполнить метод **Add.** Для удаления элемента нужно выполнить метод **RemoveAt** с параметром номер элемента в списке.

**SelectedItems –** список выбранных строк;

**SelectionMode –** режим выбора. Его возможные значения:

One — только один элемент может быть выбран в каждый конкретный момент времени;

MultiSimple — возможен выбор нескольких элементов. При использовании этого стиля при щелчке на элементе он становится выбранным и остается таким даже в случае щелчка на другом элементе до повторного щелчка на нем.

**ToStringO** Метод возвращает элемент, выбранный в текущий момент времени.

Еще один вид доступного списка — **CheckedListBox**. Он предоставляет список, но кроме текстовых строк имеется также флажок для каждого элемента в списке.

Свойство только этого вида списка – **Checkedltems.** Это свойство — перечень всех элементов в CheckedListBox, которые находятся в состоянии Checked.

У этого списка есть метод **GetItemChecked** с параметром номер элемента в списке, который возвращает true, если элемент выбран.

Элемент управления **TabControl**, как и **GroupBox,** позволяет группировать другие элементы управления. Этот элемент предоставляет простой способ организации диалоговых окон в логические части, доступные посредством вкладок в верхней части элемента управления.

TabControl содержит элементы **TabPages**, которые по существу работают подобно элементу управления GroupBox, поскольку они группируют элементы управления.

Элемент управления **dataGridView** предназначен для работы с таблицами. При копировании этого элемента на форму создается таблица без столбцов, к которой можно добавлять столбцы как в режиме проектирования, так и в режиме выполнения. При проектировании создается столбец с именем и заголовком. Для добавления столбца в режиме выполнения нужно вызвать свойство **Columns** и выполнить его метод **Add** с параметрами имя столбца и имя заголовка столбца.

Для добавления строки таблицы нужно вызвать свойство **Rows** и выполнить его метод **Add**. Таким образом создается двумерный массив, первый индекс которого - номер столбца, а второй – номер строки. Заполнение элемента таблицы происходит с помощью оператора присваивания вида *this.dataGridView1[0,nr].Value=…*

Элемент управления для создания меню это **MenuStrip**. Пункты меню создаются интерактивно; каждый пункт является элементом управления **toolStripMenuItem**, событие **Click** которого выполняется при выборе этого пункта.

Для вызова из одной формы другой нужно создать экземпляр вызываемой формы с помощью оператора вида *Form2 fr2 = new Form2();*и выполнить его метод **ShowDialog.** Закрытие формы промсходит с помощью ее метода **Close.**

Если необходимо передать из вызывающей формы в вызываемую некоторую величину, то в вызываемой форме нужно создать статический класс со статическими элементами, например,

*public static class Global*

*{*

*public static string name;*

*public static int age;*

*public static bool sex;*

*}*

В этом случае все переменные с именами вида Global.имя доступны в любой форме.

# Пример использования различных элементов управления

Создадим новое приложение Windows Form Application. На его форму поместим следующие компоненты:

* menuStrip1- меню. У этой компоненты есть свойство Items (набор пунктов). Это свойство редактируется визуально (Рис. ):

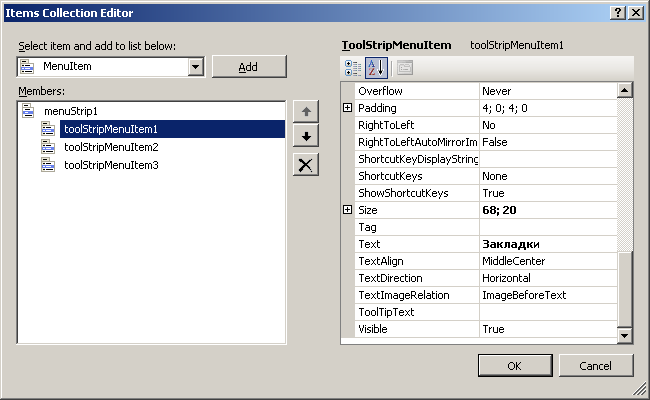


Рис. Настройка пунктов меню

Каждый пункт меню (toolStripMenuItem) имеет свойство Text – его значение отображается в приложении при его выполнении. При щелчке по пункту меню в период проектирования появится заготовка метода

*private void toolStripMenuItem1\_Click(object sender, EventArgs e)*

*{ }*

обработки события выбора данного пункта меню. Таким образом создадим меню из трех пунктов:



При выборе первого и второго пунктов меню выполняется вызов новых форм приложения:

*private void toolStripMenuItem1\_Click(object sender, EventArgs e)*

*{ Form2 fr2 = new Form2();*

*fr2.TextForm=textBox1.Text;*

*fr2.ShowDialog(); }*

*private void toolStripMenuItem2\_Click(object sender, EventArgs e)*

*{ Form3 fr3 = new Form3();*

*fr3.ShowDialog(); }*

Выбор третьего пункта меню приводит к закрытию первой формы и всего приложения:

*private void toolStripMenuItem3\_Click(object sender, EventArgs e)*

*{ this.Close(); }*

* две компоненты label (статический текст), свойство Text которых задает выводимый текст (Имя и Возраст соответственно);
* компоненту textBox для ввода текста в свойство Text;
* компоненту maskedTextBox для ввода текста в свойство Text с контролем. Вид контроля задается свойством Mask, которое должно быть строкой, состоящей из символов, каждый из которых контролирует очередной символ вводимого поля. Некоторые символы масок собраны в табл.

Таблица

|  |  |
| --- | --- |
| Символ маски | Назначение |
| 0 | Требуется цифра |
| 9 | Может быть цифра или пробел |
| # | Может быть цифра или пробел, а также знак числа + или - |
| L | Требуется латинская буква (малая или большая) |
| ? | Может быть латинская буква |
| / | Разделитель символов даты |
| < | Преобразует последующие символы в нижний регистр |
| > | Преобразует последующие символы в верхний регистр |

Вводимые символы во время выполнения программы сопоставляются с символами маски. По полю ввода можно перемещаться с помощью стрелок и курсора мыши.

* компоненту groupBox, в свойство Text которой записывается значение. Выберите любимый цвет, которое при выполнении появится в верхней части компоненты;
* внутри groupBox три компоненты radioButton со значениями поля Text соответственно Красный, Синий, Зеленый для выбора одной из них;
* компоненту checkeditListBox для размещения элементов, которые могут помечаться для помещения в список lostBox. Элементы, помещаемые в новый список, заполняются при разработке программы в свойстве Items (Рис. ):

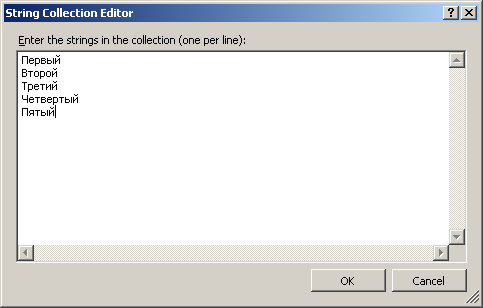


Рис. Свойство Items

* компоненту listBox для размещения выбранных элементов первого списка;
* компоненту button с текстом Вывод для определения возрасту по году рождения. Событие Click этой кнопки содержит следующий код:

*private void button1\_Click(object sender, EventArgs e)*

*{int year, age; ;*

*age = Convert.ToInt32(maskedTextBox1.Text);*

*year = 2011 - age;*

*MessageBox.Show("Первое приложение Windows!");*

*MessageBox.Show("Ваше имя: " + textBox1.Text +*

*" Ваш возраст " + maskedTextBox1.Text + " Год рождения: " + Convert.ToString(year)); }*

Функция Convert.ToString() преобразует целое число в текст.

Функция Convert.ToInt32() преобразует текст в целое число.

Функция MessageBox.Show() выводит текст на экран.

* компоненту button с текстом Любимый цвет для вывода любимого цвета. Метод Click этой кнопки содержит код:

*private void button2\_Click(object sender, EventArgs e)*

*{ string Col = "";*

*if (this.radioButton1.Checked)*

*Col = this.radioButton1.Text;*

*if (this.radioButton2.Checked)*

*Col = this.radioButton2.Text;*

*if (this.radioButton3.Checked)*

*Col = this.radioButton3.Text;*

*MessageBox.Show(Col); }*

* компоненту button с текстом Добавить в список и с кодом метода Click:

*private void button3\_Click(object sender, EventArgs e)*

*{ int i, k = 0;*

*string Elem;*

*// Проверка наличия каких-либо помеченных флажками*

*// элементов в списке CheckedListBox.*

*if (this.checkedListBox1.CheckedItems.Count > 0)*

*{ // Очистка списка ListBox, в который будет выполняться перемещение*

*this.listBox2.Items.Clear();*

*// Циклический просмотр коллекции Checkedltems списка CheckedListBox*

*//и добавление элементов в список выбранных элементов.*

*for (i = 0; i < this.checkedListBox1.Items.Count ; i++)*

*{ if (checkedListBox1.GetItemChecked(i))*

*{ Elem = this.checkedListBox1.Items[i].ToString();*

*this.listBox2.Items.Add(Elem);*

*//this.checkedListBox1.Items.RemoveAt(i-k); k++;*

*} }}}*

При выполнении форма имеет вид (Рис. ):



Рис. Форма 1 при выполнении

Форма 2 имеет вид (Рис. ):

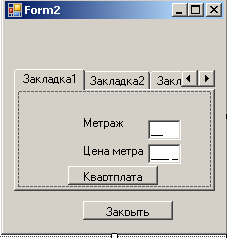


Рис. Закладки

Это компонента tabControl, которая управляет связанным набором страниц закладок. Набор закладок определяет свойство TabPages, которое настраивается визуально (Рис. )

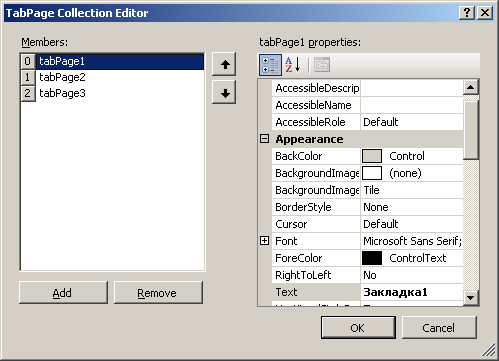


Рис. Набор закладок

Заголовок закладки определяется значением свойства Text.

На закладке можно располагать любые компоненты. Вид закладки 1 показан на рис. 9. Нажатие кнопки Квартплата вызывает выполнение кода:

*private void button1\_Click(object sender, EventArgs e)*

*{ int metr;*

*float cena;*

*string kvart;*

*metr = Convert.ToInt32(maskedTextBox1. Text);*

*cena = Convert.ToSingle(maskedTextBox2.Text);*

*kvart=Convert.ToString(metr\*cena);*

*MessageBox.Show(kvart); }*

Форма 3 имеет вид (Рис. )

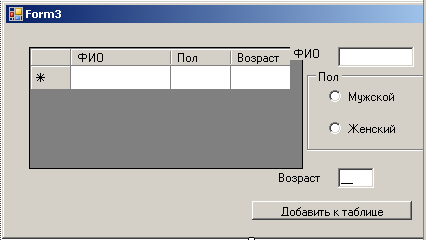


Рис. Форма с таблицей

На этой форме размещена компонента dataGridView, у которой есть свойство Columns, которое настраивается визуально (Рис. ):

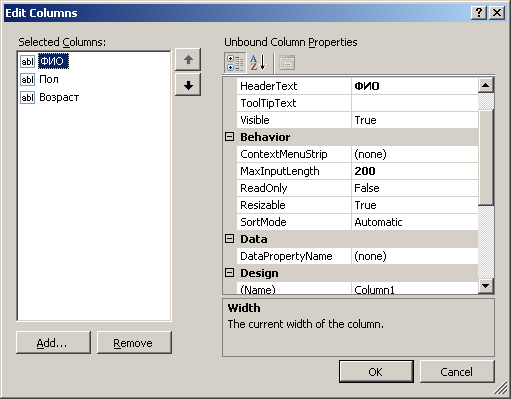


Рис. Настройка столбцов таблицы

Свойство Header Text определяет заколовок столбца таблицы, а свойство Width определяет ширину столбца.

Остальные компоненты предназначены для добавления строк таблицы. Кнопка button с текстом Добавить к тоблице имеет следующий код метода Click:

*private void button1\_Click(object sender, EventArgs e)*

*{string Col = "";*

*if (this.radioButton1.Checked)*

*Col = this.radioButton1.Text;*

*if (this.radioButton2.Checked)*

*Col = this.radioButton2.Text;*

*this.dataGridView1.Rows.Add(); //Добавление строки таблицы*

*this.dataGridView1[0,nr].Value = textBox1.Text;*

*this.dataGridView1[1,nr].Value = Col;*

*this.dataGridView1[2,nr].Value = maskedTextBox1.Text;*

*nr++; }*

Первый индекс – номер столбца, второй – номер строки, которая задается как глобальное свойство:

*public int nr = 0;*

В качестве значения второго столбца (его индекс равен 1) сохраняется текст соответствующей компоненты radioButton.

Вид формы 3 в процессе выполнения показан на рис.

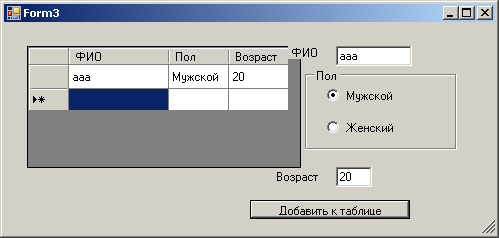
**

Рис. Форма 3 в процессе выполнения

# Обработка строковых данных

# Описание строк в С#

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.

Пример.

*string a = "hello";*

*string b = "h";*

*b += "ello";*

*Console.WriteLine(a == b);*

Переменные string объявляются как все прочие переменные простых типов – с явной или отложенной инициализацией. Чаще всего, при объявлении строковой переменной инициализация задается строковой константой. Но у strng достаточно много конструкторов. Они позволяют сконструировать строку из:

* символа, повторенного заданное число раз;
* массива символов char[];
* части массива символов.

Примеры объявления строк с вызовом разных конструкторов:

*public void TestDeclStrings()*

*{//конструкторы*

*string world = “Мир”;*

*string sssss = new string(’s’,5);*

*char[] yes = “Yes”.ToCharArray();*

*string stryes = new string(yes);*

*string strye = new string(yes,0,2);*

*Console.WriteLine(“world = {0}; sssss={1}; stryes={2}; strye= {3}”, world, sssss, stryes, strye);}*

Объект world создан без явного вызова конструктора, а объекты sssss, stryes, strye созданы разными конструкторами sring.

Над строками – объектами этого класса – определен широкий набор операций.

Оператор + служит для объединения строк.

*string a = "good " + "morning";*

В данном примере создается строковый объект, содержащий текст "good morning".

Оператор [] служит для доступа только для чтения к отдельным знакам объекта string.

*string str = "test";*

*char x = str[2];* // x = 's';

Строковая константа имеет тип string и может быть написана в двух формах: в кавычках и в кавычках с @ (точная константа). Константа в кавычках:

"good morning"

Точная константа начинается со знака @ и заключена в двойные кавычки. Пример:

*@"good morning"*

Преимущество точных строк заключается в том, что escape-последовательности не обрабатывается, благодаря чему можно удобно написать, например, полное имя и путь файла:

*@"c:\Docs\Source\a.txt" // лучше чем "c:\\Docs\\Source\\a.txt"*

Чтобы включить знак двойной кавычки в строку в кавычках с @, следует использовать знак кавычек дважды:

*@"""Ahoy!"" cried the captain."*

**Функции обработки строк**

**Сравнение строк**

# Функция Compare (string, string) сравнивает два заданных объекта [string](http://msdn.microsoft.com/ru-ru/library/system.string(v=vs.90).aspx) и возвращает целое число, равное

# 0, если строки равны,

# >0, если первая строка больше второй,

# <0, если первая строка меньше второй/

Пример сортировка по убыванию:

*public int Compare () {*

*string s1 = “ccc”;*

*string s2 = “dfgh”;*

*return - string.Compare (s1,s2); }*

**Работа с подстрокой**

# Подстрокой является последовательность символов, содержащихся в строке. Функция Substring [.NET Framework 3.5](http://msdn.microsoft.com/ru-ru/library/hxthx5h6(v=VS.90).aspx)

* [Silverlight](http://msdn.microsoft.com/ru-ru/library/hxthx5h6(v=VS.95).aspx)

возвращает подстроку исходной строки, начиная с заданной позиции строки. Она существует в двух формах:

**1.** Substring(int startIndex)

**2.** Substring(int startIndex, int length)

#### Первая форма принимает один параметр – начальная позиция (нумеруется с 0), вторая форма дополнительно имеет второй параметр – длину подстроки.

Пример 1:

*string strText = "C# substring function";*

*string out= strText.Substring(3);*

 Результат: substring function

Пример 2:

*string strText = "C# substring function";*

*string out= strText.Substring(3, 9);*

Результат:substring

**Функция Split делит строку по разделителям и размещает слова в массиве**:

Split(char[] separator);

#### Пример:

*string strData = "a,b,c,d,e,f,g,h,i,j";*

*separator = new char[] { ',' };*

*string[] strSplitArr = strData.Split(separator);*

Результат:a b c d e f g h i j

Функция **Remove** удаляет часть строки, начиная с начального символа возможно с учетом количества символов

**1.** Remove(int startIndex);

**2.** Remove(int startIndex, int count);

Начало формы

Пример 1:

*string strText = "C# Remove String Function";*

*string out= (strText.Remove(2));*

Результат:C#

Пример 2:

*string strText = "C# Remove String Function";*

*string out= strText.Remove(10, 7);*

Результат:C#

**Функция Replace заменяет часть строки другой строкой.** Она существует в двух видах:

**1.** Replace(char oldChar, char newChar);

**2.** Replace(string oldValue, string newValue);

Пример 1:

*string strText = "C# Replace $tring Function";*

*string out=strText.Replace('$', 'S');*

Результат: C# Replace String Function

Пример 2:

*string strText = "ab abc abcd abcde abcdef";*

*string out=strText.Replace("ab", "AB");*

Результат:AB ABc ABcd ABcde ABcdef

Функция **IndexOf используется для получения первой позиции строки или символа** в исходной строке. Поиск ведется или с начала строки или начиная с заданной позиции строки лмбо до конца строки, либо в пределах заданного количества символов.

Формы функции:

**1.** IndexOf(char value);

**2.** IndexOf(string value);

**3.** IndexOf(char value, int startIndex);

**4.** IndexOf(string value, int startIndex);

**5.** IndexOf(char value, int startIndex, int count);

**6.** IndexOf(string value, int startIndex, int count);

Начало формы

#### Пример

#### *string strText = "C# IndexOf String Function";*

#### *string out1=strText.IndexOf('#');*

#### *string out2=strText.IndexOf("Str", 15);*

#### *string out3=strText.IndexOf("Str", 3, 15);*

#### Результат: **1 -1 11**

#### Функция L**ength возвращает длину строки**

Пример:

*strText = "C# string length function-123";*

*string out=(strText.Remove(strText.Length);*

Результат -1

**Функция Join объединяет значения элементов текстового массива в строку, вставляя между словами разделители. Процесс объединения может начаться с первого элемента массива или с указанного с учетом указанного количества элементов**

**Функция существует в двух формах:**

**1.** Join(string separator, string[] value);

**2.** Join(string separator, string[] value, int startIndex, int count);

Пример

*int arrLength = 10;*

*string[] arr1 = new string[arrLength];*

*int i = 0;*

*for (i = 0; i < arr1.Length; i++)*

*arr1[i] = i.ToString();*

*string out,out1;*

*out=string.Join(",", arr1);*

*out1=string.Join(",", arr1, 7, arr1.Length - 7);*

Результат**:**

0,1,2,3,4,5,6,7,8,9  
  
**Функция StartsWith проверяет, начинается ли заданная строка с указанной**

**подстроки или нет. Она возвращает булевское значение.**

Ее вид:

StartsWith(string value);

Пример.

*string strText = "ASP.Ne t C# string.StartsWith function";*

*string out=strText.StartsWith("ASP.Net");*

*string out1=strText.StartsWith("S");*

 Результат:

True

False

Функция IndexOfAny() возвращает номер первой позиции строки, совпадающей с началом массива символов.

Функция LastIndexOfAny()возвращает номер последней позиции строки, совпадающей с концом заданного массива символов.

Пример

*string myString = "To.be.or.not.to.be."};      
    char[] myChars = {'b', 'e'};  
    int index1 = myString.IndexOfAny(myChars);*

*int index2 = myString.LastIndexOfAny(myChars);*

Результат: 3 17

**Пример задачи, использующей строковые данные**

Задан текст, состоящий из слов, записанных через запятые. Длина текста не больше 200 символов. Необходимо выдать на экран слова и их длины. Если слово состоит только из цифр, то возвести его в куб. Кроме того, составить выходную строку, содержащую нецифровые слова, в которых порядок символов изменен на обратный.

Разработаем функции:

str\_sl – разбиение строки на слова

chisl\_cub – формирование массива кубов чисел по массиву слов

obr\_sl – получение из нецифровых слов строки слов в обратном порядке

Для каждой функции составим таблицу спецификаций и словесный алгоритм:

Функция str\_sl

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Входные величины | | | Выходные величины | | |
| Имя | Назначение | Диапазон | Имя | Назначение | Диапазон |
| X | строка | Текст | n | Число слов | Целое число |
| Rsl | Разделитель слов | Текст | mslov | Массив слов | Текст |

Словесный алгоритм:

1.     Разбиение строки x на массив слов mslov

2. n=длине массива mslov

3.     Возврат n

Функция chisl\_cub

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Входные величины | | | Выходные величины | | |
| Имя | Назначение | Диапазон | Имя | Назначение | Диапазон |
| n | Число слов | Целое число | mch | Массив числовых слов | Целые числа |
| mslov | Массив слов | Текст | l | Число числовых слов | Целое число |

#### Словесный алгоритм:

1. i=0 l=0

2. ПОКА i<n ВЫПОЛНИТЬ

2.1. j1=номеру позиции первой цифры в mslovi

2.2. j2=номеру позиции последней цифры в mslovi

2.3. ЕСЛИ j1=0 И j2=длина mslovi - 1 ТО

2.3.1. b=результату преобразования mslovi в число

2.3.2. mchl=b\*b\*b

2.3.3. l=l+1

ЕСЛИ ВСЕ

2.4. i=i+1

ПОКА ВСЕ

1. Возврат l

Функция obr\_sl:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Входные величины | | | Выходные величины | | |
| Имя | Назначение | Диапазон | Имя | Назначение | Диапазон |
| n | Число слов | Целое число | rezstr | Результирующая строка из слов в обратном порядке | Текст |
| Mslov | Массив слов | Текст |  |  |  |

Словесный алгоритм:

1. i=0 rezstr=””

2. ПОКА i<n ВЫПОЛНИТЬ

2.1. j=числу цифр в mslovi

2.2. ЕСЛИ j≠длине mslov, ТО

2.2.1. a=обращение слова mslovi

2.2.2. Присоединение a к rezstr

2.2.3. Присоединение “,” к rezstr

ЕСЛИ ВСЕ

2.3. i=i+1

ПОКА ВСЕ

3. Возврат rezstr

Тексты функций

*static int str\_sl(string x, char[] rsl, char[] kstr, out string[] mslov)*

*{char[]razd =new char[2];*

*razd[0]=rsl[0];*

*razd[1]=kstr[0];*

*int n = 0, beg;*

*mslov = new string[20];*

*string y;*

*mslov = x.Split(razd);*

*n = mslov.Length;*

*return n;*

*}*

*static string obr\_sl(int n,string[] mslov)*

*{int i,j,j1,j2;*

*string rezstr,b;*

*string a;*

*char[] cif = {'1','2','3','4','5','6','7','8','9','0'};*

*rezstr=new string(' ',0) ;*

*for (i = 0; i < n; i++)*

*{ b = string.Copy(mslov[i]);*

*int l = b.Length;*

*j1 = b.IndexOfAny(cif);*

*j2 = b.LastIndexOfAny(cif);*

*if (j1 != 0 && j2 != 0)*

*{a = new string(' ', 0);*

*for (j = 0; j<l; j++)*

*a = b[j]+a;*

*rezstr = rezstr + a;*

*rezstr = rezstr + ",";*

*}*

*} return rezstr;*

*}*

*static int chisl\_cub(int n, string[] mslov, out int[] mch)*

*{*

*int l = 0, i, j1, j2, b;*

*mch = new int[20];*

*char[] cif = { '1', '2', '3', '4', '5', '6', '7', '8', '9', '0' };*

*for (i = 0; i < n; i++)*

*{*

*j1 = mslov[i].IndexOfAny(cif);*

*j2 = mslov[i].LastIndexOfAny(cif);*

*if (j1 == 0 && j2 == mslov[i].Length - 1)*

*{*

*b = Convert.ToInt32(mslov[i]);*

*mch[l] = b \* b \* b; l++;*

*}*

*}*

*return l;*

*}* **Компонента Graphics**

Для рисования нужно помещать на форму поверхность рисования, которую обеcпечивает этот класс. Основные действия по рисованию описаны в событии Paint. Для рисования какой-то кривой (окружности, прямой, треугольника и т.д.) необходимо выполнить какой-то метод Graphics.

Для указания цвета линии задается некоторый объект карандаш (Pen), а для указания цвета заливки задается некоторый объект кисть (Brush). Для вывода текста используется некоторый шрифт – объект Font.

Объект Pen создается с помощью конструктора вида

Pen p1 = new Pen(цвет линии, толщина линии);

Цвет линии задается элементом структуры Color

В этой структуре определены имена для 128 цветов. 16 из них (основные) определены в табл.

Таблица

Имена основных цветов

|  |  |
| --- | --- |
| Название цвета | Имя поля структуры |
| Черный | *Black* |
| Серебристый | *Silver* |
| Серый | *Gray* |
| Белый | *White* |
| Темно-бордовый | *Maroon* |
| Красный | *Red* |
| Фиолетовый | *Purple* |
| Фуксия | *Fuchsia* |
| Зеленый | *Green* |
| Светло-зеленый | *Lime* |
| Оливковый | *Olive* |
| Желтый | *Yellow* |
| Темно-синий | *Navy* |
| Синий | *Blue* |
| Морской волны | *Teal* |
| Голубой | *Aqua* |

Например: Pen p1 = new Pen(Color.Blue, 2);

Объект Brush создается с помощью функции

SolidBrush b1 = new SolidBrush(цвет);

Создание поверхности рисования происходит с помощью вызова

Graphics g = e.Graphics;

После этого можно применять различные методы Graphics.

При рисовании используется традиционная для машинной графики система координат: начало координат - верхний левый угол, ось X возрастает слева направо, а ось Y сверху вниз.

Во многих методах используется структура Rectangle, которая содержит набор из четырех чисел с плавающей запятой, определяющих расположение и размер прямоугольника. Создание этой области происходит с помощью конструктора, который инициализирует новый экземпляр класса [RectangleF](http://msdn.microsoft.com/ru-ru/library/system.drawing.rectanglef) с указанным расположением и размером.

RectangleF(float абсцисса верхнего угла прямоугольнике,

float ордината верхнего левого угла прямоугольника,float ширина,float высота)

Также в некоторых методах используется структура PointF, которая представляет упорядоченную пару координат Х и Y с плавающей запятой, определяющих точку на двумерной плоскости. Ее создание производится с помощью конструктора PointF(float абсцисса точки,float ордината точки)

Некоторые методы перечислены в табл. 11.

Таблица 11

Некоторые методы класса Graphics

|  |  |
| --- | --- |
| Метод | Описание |
| DrawArc(Pen карандаш,RectangleF прямоугольник,float Угол (в градусах), который измеряется по часовой стрелке, начиная от оси X и заканчивая начальной точкой дуги,float Угол (в градусах), который измеряется по часовой стрелке, начиная от значения параметра startAngle и заканчивая конечной точкой дуги) | Рисует дугу, которая является частью эллипса, заданного структурой [Rectangle](http://msdn.microsoft.com/ru-ru/library/system.drawing.rectangle)F |
| DrawEllipse(Pen карандаш,  RectangleF прямоугольник) | Рисует эллипс, определяемый ограничивающей структурой [RectangleF](http://msdn.microsoft.com/ru-ru/library/system.drawing.rectanglef) |
| DrawLine(Pen карандаш,PointF точка1,  PointF точка2) | Проводит линию, соединяющую две структуры [PointF](http://msdn.microsoft.com/ru-ru/library/system.drawing.pointf) |
| DrawPolygon(Pen карандаш,PointF[] массив точек) | Рисует многоугольник, определяемый массивом структур [PointF](http://msdn.microsoft.com/ru-ru/library/system.drawing.pointf) |
| DrawRectangle(Pen карандаш,int абсцисса,int ордината,int ширина,int высота) | Рисует прямоугольник, который определен парой координат, шириной и высотой |

В качестве примера выведем на форму фигурку барана. Для этого определим в классе формы поля:

float x0 = 230;//Абсцисса начальной точки

float y0 = 180;//Ордината начальглй точки

float t = 0;//Для последующей анимации

Определим событие Paint:

*public void OnPaint(object sender, PaintEventArgs e)*

*{ Graphics g = e.Graphics; // Описать поверхность рисования*

*DrawSheep(g, x0, y0, t); // Нарисовать барана }*

*и метод DrawSheep:*

*void DrawSheep(Graphics g, float x0, float y0, float t)*

*{Pen p1 = new Pen(Color.Blue, 2); // заготавливаем карандаши и кистьи*

*Pen p2 = new Pen(Color.Red, 3); // для рисования барана*

*Pen p3 = new Pen(Color.Gray, 4);*

*Pen p4 = new Pen(Color.Black, 3);*

*SolidBrush b1 = new SolidBrush(Color.Green);*

*g.FillEllipse(b1, x0, y0, 100, 50); // туловище*

*float xhead = x0 + 60; // вычисляем координаты головы*

*float yhead = y0 - 20;*

*g.DrawPolygon(p1, new PointF[] { new PointF(xhead, yhead), new PointF(xhead + 30, yhead + 30),new PointF(xhead + 60, yhead) });*

*// рисуем голову*

*g.DrawEllipse(p2, x0 + 80, y0 - 12, 5, 5); // рисуем глаза*

*g.DrawEllipse(p2, x0 + 95, y0 - 12, 5, 5);*

*g.DrawArc(p3, x0 + 20, y0 - 45, 50, 50, 0, -90); // рисуем рога*

*g.DrawArc(p3, x0 + 60, y0 - 45, 50, 50, 0, -90);*

*float xlegs = x0 + 10; // вычисляем координаты ног*

*float ylegs = y0 + 42; // и рисуем четыре ноги*

*g.DrawLines(p4, new PointF[] { new PointF(xlegs, ylegs), new PointF(xlegs + 20 \* (1 - t), ylegs + 20), new PointF(xlegs, ylegs + 40) });*

*xlegs = xlegs + 20;*

*g.DrawLines(p4, new PointF[] { new PointF(xlegs, ylegs), new PointF(xlegs + 20 \* (1 - t), ylegs + 20), new PointF(xlegs, ylegs + 40) });*

*xlegs = xlegs + 40;*

*g.DrawLines(p4, new PointF[] { new PointF(xlegs, ylegs), new PointF(xlegs + 20 \* (1 - t), ylegs + 20), new PointF(xlegs, ylegs + 40) });*

*xlegs = xlegs + 20;*

*g.DrawLines(p4, new PointF[] { new PointF(xlegs, ylegs), new PointF(xlegs + 20 \* (1 - t), ylegs + 20), new PointF(xlegs, ylegs + 40) });*

*g.DrawArc(p4, x0 - 25, y0 + 12, 25, 25, 0, 245);*

*// рисуем хвост*

*}*

Зададим размеры формы 600, 480

При запуске приложения получим (рис. )

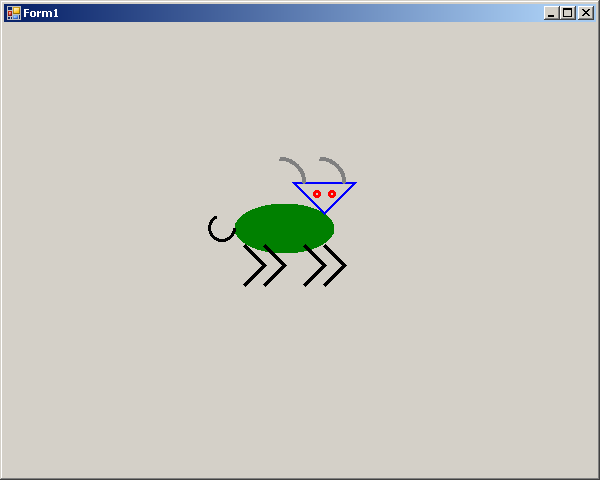


Рис. Рисунок барана

Анимация - это почти всегда меняющаяся картинка, в примере будет меняться с одной стороны положение барашка, а с другой изгиб лап. Изгиб лап у барана определяется параметром t из отрезка [-0.3,0.7], а положение барана будет также вычисляться в зависимости от параметра t

Здесь будет использован стандартный класс Math, содержащий математические функции, но после вычисления необходимо привести результат вычисления к типу float. Для регулярного изменения координат барана имеется невизуальная компонента Timer. Перетаскивание эту еомпоненту на форму приведет к ее размещению в невидимой части экрана, так как она невизуальная. В свойствах устанавливается значение Interval = 60. Это свойство определяет, через сколько миллисекунд должен быть "тик" часов. Значение равное 1000 соответствует одной секунде. Следует учитывать, что интервал "тиканья" является относительно приблизительным, и может варьироваться в зависимости от мощности компьютера и его текущей загрузки. В списке событий таймера имеется одно событие Tick. Метод реакции на его наступление будет в примере называться Tick и содержать код:

*private void timer1\_Tick(object sender, EventArgs e)*

*{float dt = 0.01f;*

*t = t + dt;*

*if ((t < - 0.3f)||(t > 0.7f))*

*{ dt = -dt; }*

*x0 = 230 + (float)(100 \* Math.Sin(10 \* t));*

*y0 = 180 - (float)(20 \* Math.Sin(30 \* t));*

*Invalidate(); //Перерисовка формы, то есть вызов метода Paint*

*}*

Для запуска таймера с событием DoubleClick формы связывается метод, который «запускает» таймер:

*private void DoubleClick(object sender, EventArgs e)*

*{ timer1.Enabled = true; }*

При этом рисунок будет перерисовываться, и барашек будет двигаться