ЛАБОРАТОРНАЯ РАБОТА № 6.
ОСНОВЫ РАБОТЫ В СРЕДЕ MS VISUAL STUDIO 2005

Цель: познакомить с порядком запуска MS Visual studio 2005, правилами открытия, сохранения и выполнения проекта, основами управления свойствами компонентов, составлением простейших алгоритмов.

Открытие и сохранение файлов

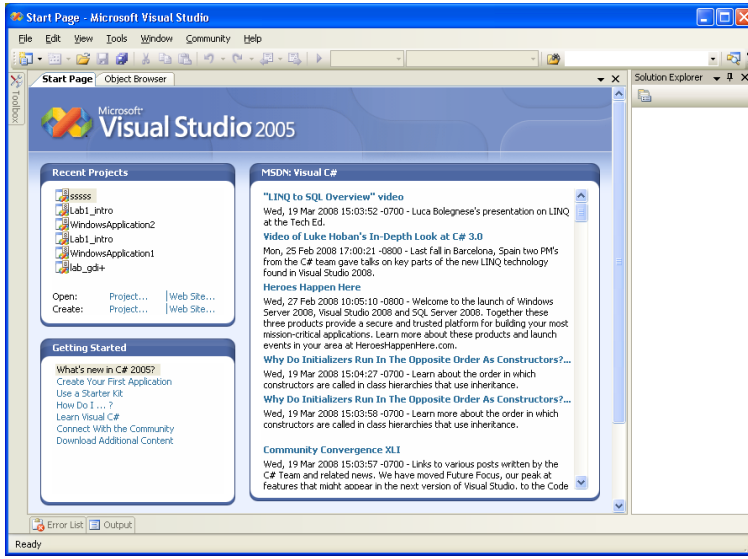
MS Visual studio 2005 является проектно-ориентированной средой для разработки приложений. Это означает, что каждое приложение оформляется как проект, содержащий один или несколько файлов дополнительно к файлу проекта. Несколько разных типов файлов могут быть частью проекта, содержащего исходные тексты программ, формы, скомпилированные модули, конфигурацию, дополнения, пакеты и резервные файлы. В этом разделе рассматриваются различные файлы проекта и их использование.

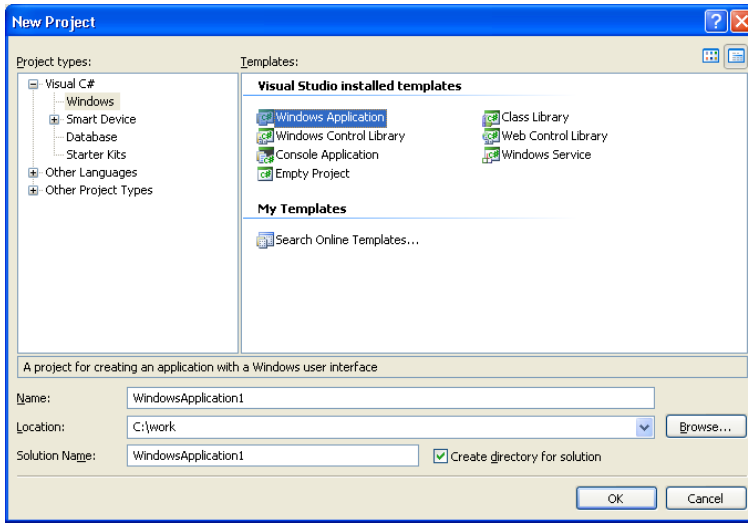
В MS Visual studio 2005 можно создавать приложения на различных языках программирования (C++, C#, VB.NET). В наших лабораторных работах будет использоваться язык C#. Программы для Windows должны иметь графический интерфейс пользователя. Графическая часть проекта Delphi, которая отображает вид программы, сохраняется в файлах ресурсов (с расширением dfm), а коды (тексты программ) находятся в других файлах (с расширением pas, причем главным файлом проекта является файл с расширением dpr). Поэтому каждый проект для Windows состоит из нескольких файлов, на которые ссылаются как на проект. Создание каждой новой программы начинается с создания проекта.

Интегрированная среда разработки (Integrated Development Environment — IDE) — это первое, что вы видите при запуске Delphi.

Создание проекта

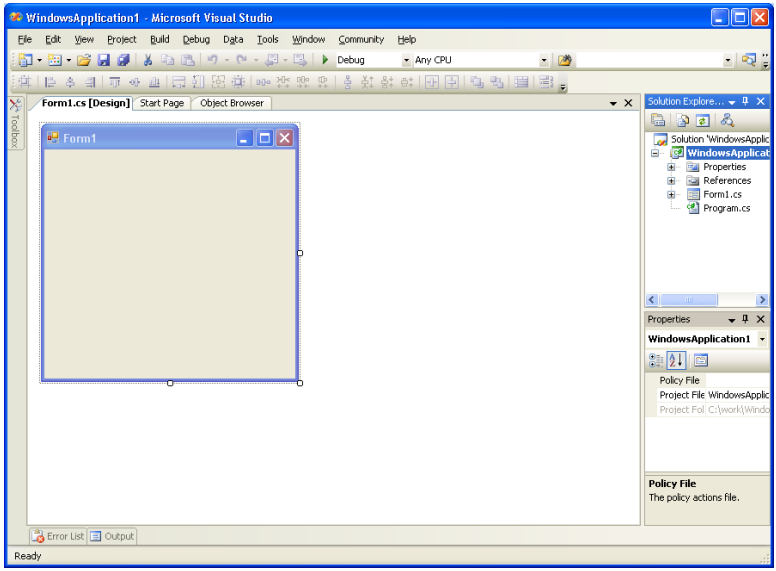
Когда вы открываете MS Visual studio 2005, то необходимо обязательно создать новый проект WindowsApplication на языке C#.



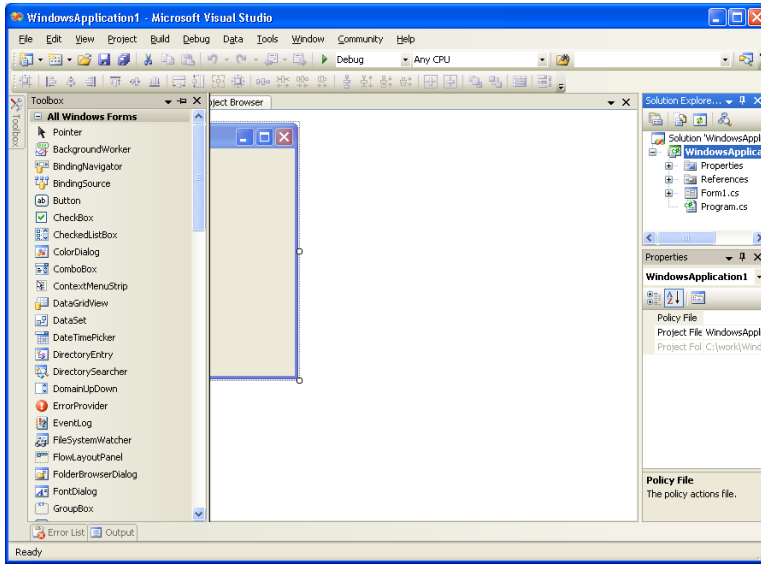


После выполнения в папке C:\work система создаст папку для проекта WindowsApplication1. В процессе создания проекта и сохранения промежуточных результатов в данной папке будут появляться отдельные файлы и новые папки. Файл проекта будет

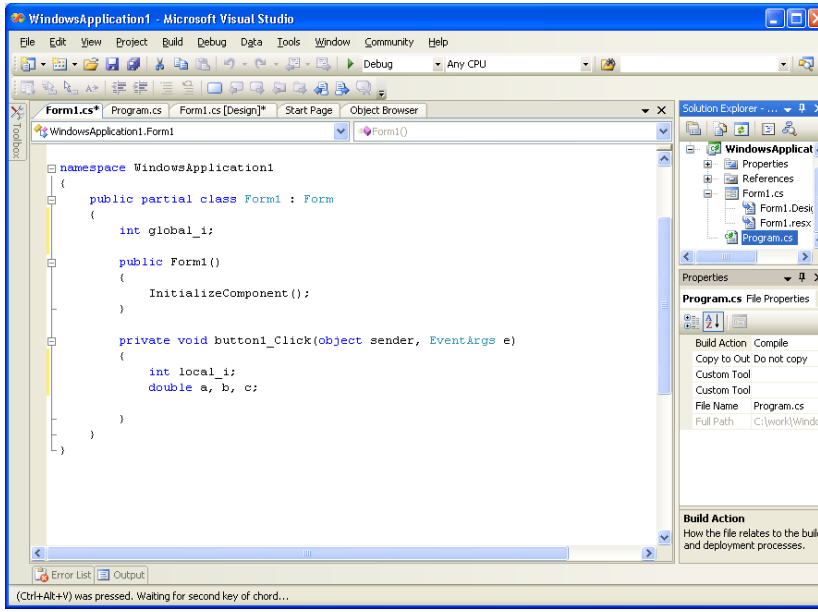
сохранен с именем <Имя_проекта>.sln. При повторном редактировании проекта открывать следует именно этот файл.



Закладка панели компонентов (ToolBox) находится слева сверху от окна формы. При наведении на нее курсора мыши появляются все компоненты. Окно свойств и событий находится справа.



Для создания приложения на форму размещаются необходимые компоненты. Для записи текста программы следует активировать компонент, перейти на закладку события, после двойного щелчка мыши приступить к написанию кода программы. Локальные переменные следует описывать в начале блока кода обработки события. Глобальные переменные описаются в начале блока кода класса (по умолчанию Form1).



Типы

Типы данных и арифметические операции и функции

Примеры	Обозначения
целый	
	int
вещественный	*double*
символьный	*char*
логический	*Boolean*

Тело программы заключается в фигурные скобки {} (как в Паскале begin и end). Нет специального раздела описаний. Однострочный комментарий – два слеша. Многострочный:

```
/*
Это комментарий
Это комментарий
Это комментарий
*/
```

{ } – как в Паскале соответствуют begin и end.

Оператор присваивания **=**

В C# предусмотрены специальные составные операторы присваивания, которые упрощают программирование определенных инструкций присваивания. Лучше всего начать с примера. Рассмотрим следующую инструкцию:

```
x = x + 10;
```

Используя составной оператор присваивания, ее можно переписать в таком виде:

```
x += 10;
```

Пара операторов += служит указанием компилятору присвоить переменной x сумму текущего значения переменной x и числа 10. А вот еще один пример. Инструкция

```
x = x - 100;
```

аналогична такой:

```
x -= 100;
```

Операторы отношений

Оператор	*Значение*
==	Равно
!=	Не равно
>	Больше
<	Меньше
>=	Больше или равно
<=	Меньше или равно

Логические операторы

Оператор	Значение
&	И
|	ИЛИ
^	Исключающее ИЛИ
&&	Сокращенное И
||	Сокращенное ИЛИ
!	НЕ

Что касается логических операторов, то их операнды должны иметь тип `bool`, и результат логической операции всегда будет иметь тип `bool`. Логические операторы `&`, `|`, `^` и `!` выполняют базовые логические операции И, ИЛИ, исключающее ИЛИ и НЕ в соответствии со следующей таблицей истинности.

p	*q*	*p* & *q*	*p* | *q*	*p* ^ *q*	!*p*
false	false	false	false	false	true
true	false	false	true	true	false
false	true	false	true	true	true
true	true	true	true	false	false

Как видно из этой таблицы, операция "исключающее ИЛИ" сгенерирует результат ИСТИНА лишь в случае, если истинен только один из ее операндов.

C# поддерживает специальные *сокращенные* (short-circuit) версии логических операторов И и ИЛИ, которые можно использовать для создания более эффективного кода. Вспомним, что, если в операции И один операнд имеет значение ЛОЖЬ, результат будет ложным независимо от того, какое значение имеет второй операнд. А если в операции ИЛИ один операнд имеет значение ИСТИНА, результат будет истинным независимо от того, какое значение имеет второй операнд. Таким образом, в этих двух случаях вычислять второй операнд не имеет смысла. Если не вычисляется один из операндов, тем самым экономится время и создается более эффективный код.

Сокращенный оператор И обозначается символом `&&`, а сокращенный оператор ИЛИ — символом `||` (их обычные версии обозначаются одинарными символами `&` и `|`, соответственно). Единственное различие между обычной и сокращенной версиями этих операторов состоит в том, что при использовании обычной операции всегда вычисляются оба операнда, в случае же сокращенной версии второй операнд вычисляется только при необходимости.

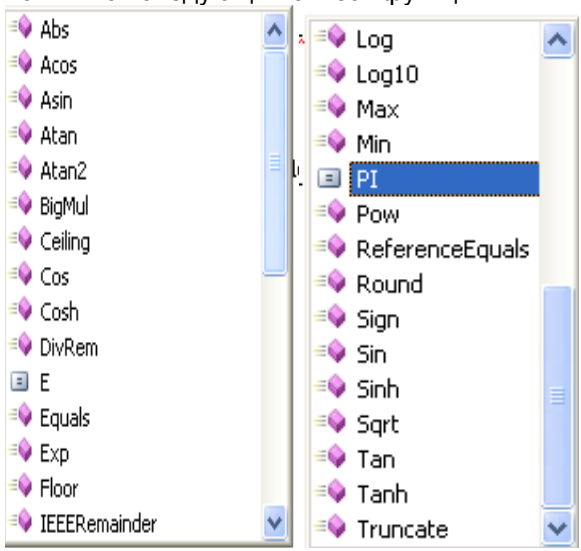
Арифметические операторы:

Оператор	***Действие***
+	Сложение
–	Вычитание, унарный минус
*	Умножение
/	Деление
%	Деление по модулю
--	Декремент
++	Инкремент

Встроенные функции
Используются в следующем виде. Пример
У=Math.Abs(x);
Y= Math.Min(z,t);

Y=Math.Pi;

Список встроенных функций можно посмотреть, если на записать фразу Math и за ней поставить точку и подождать, появиться следующий список функций.



Значения строковых величин заключается в кавычки.

Вывод данных может осуществляется следующим образом:

```
Double c,x;
X:=-35;
C=Math.Abs(x);
label1.Text =c.ToString();
```

Команда ветвления

```
if(условие) инструкция;
else инструкция;
```

Здесь под элементом *инструкция* понимается одна инструкция языка C#. Часть `else` необязательна. Вместо элемента *инструкция* может быть использован блок инструкций. В этом случае формат записи `if`-инструкции принимает такой вид:

```
if(условие)
{
    последовательность инструкций
}
else
{
    последовательность инструкций
}
```

Найти max(a,b,c). Данные вводятся и выводятся в textBox *(как Edit в Delphi)*.

```
private void button1_Click(object sender, EventArgs e)
    {
        int a, b, c,max;
          a = Convert.ToInt32(textBox1.Text); - перевод в число
        b = Convert.ToInt32(textBox2.Text);
        c = Convert.ToInt32(textBox3.Text);
        if (a > b)            {
           max = a;
        } else {
           max = b;
        }
        if (c > max)
        {
           max = c;
        }
        textBox4.Clear();
```

```
            textBox4.Text += "Max=" + max.ToString() +
(char)13 + (char)10; вывод результата
}
```

Вычислить сумму и произведение чисел от 1 до 10.

```
 int i;
double sum, f;
sum = 0;
f = 1;
for (i = 1; i <= 10; i++)
{
   sum = sum + i;
   f = f * i;
}
   label1.Text=sum.ToString();
   label2.Text=f.ToString();
```

Вывести на экран занчания функции y=x2, x=1(0.1)2

Вариант 1

```
          double x, y;
         string s1,s2;
         x = 1;
      while  (x<=2.2001)
      {y=x*x;
       s1 = x.ToString();
       s2 = y.ToString();
 listBox1.Items.Add( s1 +"      "+ s2);
       x=x+0.1;
      }
```

Вариант 2

```
double x, y;
        string s1,s2;
        x = 1;
        do
        {
           y = x * x;
           s1 = x.ToString();
           s2 = y.ToString();
           listBox2.Items.Add(s1 + "    " + s2);
           x = x + 0.1;
        }
        while (x <=2.2001);
```

Массив

```
int[] a = new int [11];
            int i,c;
            label1.Text = "";
            for (i=1; i<=10; i++)
            {
                c=i*i;
                a[i]=c;
 label1.Text = label1.Text + a[i].ToString()+"  ";
}
```