Среда разработки приложений Visual Studio.NET

Платформа .NET включает не только среду разработки для нескольких языков программирования, называемую Visual Studio.NET, но и множество других средств, механизмы поддержки баз данных, электронной почты и др.

В состав платформы .NET для обеспечения переносимости входят компиляторы, переводящие программу не в машинные коды, а в промежуточный язык (Microsoft Intermediate Language, **MSIL, или просто IL**), который не содержит команд, зависящих от языка, операционной системы и типа компьютера. Программа на этом языке выполняется не самостоятельно, а под управлением системы, которая называется **общеязыковой средой выполнения (Common Language Runtime, CLR).**

Среда CLR может быть реализована для любой операционной системы. При выполнении программы CLR вызывает так называемый **JIT-компилятор**, переводящий код с языка IL в машинные команды конкретного процессора, которые немедленно выполняются.

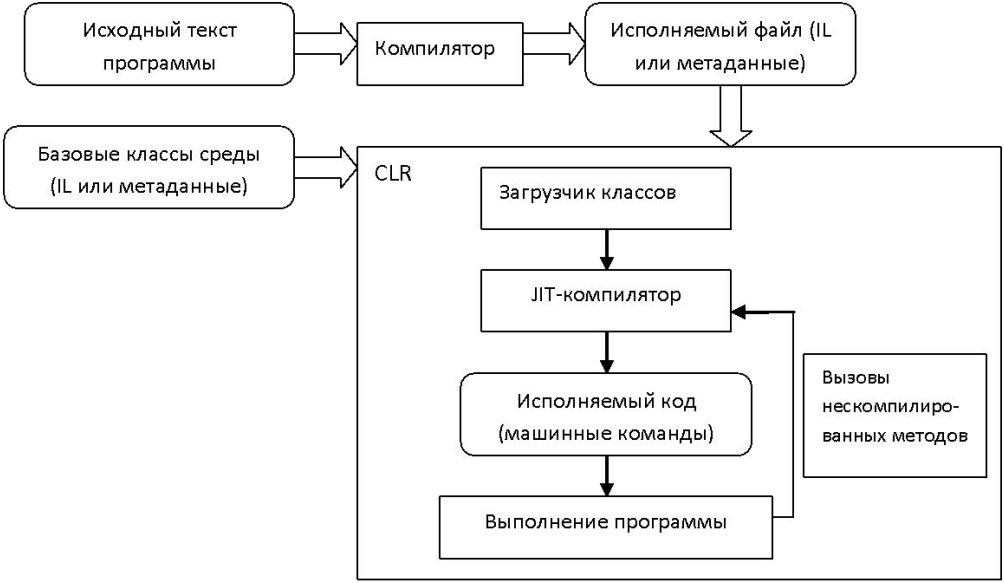
Компилятор в качестве результата своего выполнения создаёт так называемую **сборку** – файл с расширением exe или dll, который содержит код на языке IL и метаданные. Метаданные представляют собой сведения об объектах, используемых в программе, а также сведения о самой сборке. Они позволяют организовать межъязыковое взаимодействие, обеспечивают безопасность и облегчают развёртывание приложений, то есть установку программ на компьютеры пользователей.

Платформа .NET содержит огромную **библиотеку классов**, которые можно использовать при программировании на любом языке .NET. На самом нижнем находятся базовые классы среды, которые используются при создании любой программы: классы ввода-вывода, обработки строк, управления безопасностью, графического интерфейса, хранения данных и пр.

Над этим слоем находится набор классов, позволяющий работать с базами данных и XML. Классы самого верхнего уровня поддерживают разработку распределенных приложений, а также веб- и Windows-приложений. Программа может использовать классы любого уровня.

Библиотека классов вместе с CLR образуют *каркас* (framework), то есть основу платформы.

Схема выполнения программы в .NET



Среда Visual Studio.NET

Среда разработки Visual Studio.NET предоставляет мощные и удобные средства написания, корректировки, компиляции, отладки и запуска приложений, использующих .NET-совместимые языки (C#, VB.NET, C++ и J#

Платформа .NET является открытой средой. Это значит, что компиляторы для неё не могут поставляться и сторонними разработчиками. К настоящему времени разработаны десятки компиляторов для .NET, например, Ada, COBOL, Delphi, Eiffel, Lisp, Oberon, Perl, Python.

Все .NET –совместимые языки должны отвечать требованиям общеязыковой спецификации (Common Language Specification, CLS), в которой описывается набор общих для всех языков характеристик. Это позволяет использовать для разработки приложения несколько языков программирования и вести полноценную межъязыковую отладку. Все программы независимо от языка используют те же базовые классы библиотеки .NET.

Приложение в процессе разработки называется *проектом*. Проект объединяет всё необходимое для создания приложения: файлы, папки, ссылки и прочие ресурсы. Среда Visual Studio.NET позволяет создавать проекты различных типов, например:

- *Windows-приложение* использует элементы интерфейса Windows, включая формы, кнопки, флажки и пр.;

- *консольное приложение* выполняет вывод на экран «на консоль», то есть в окно командного процессора;

- *библиотека классов* объединяет классы, которые предназначены для использования в других приложениях;

- *веб-приложение* – это приложение, доступ к которому выполняется через браузер (например, Internet Explorer) и которое по запросу формирует веб-страницу и отправляет её клиенту по сети;

- *веб-сервис* – компонент, методы которого могут вызываться через Интернет.

Консольные приложения

Среда Visual Studio.NET работает на платформе Windows и ориентирована на создание Windows- и веб-приложений, однако разработчики предусмотрели работу и с консольными приложениями. При запуске консольного приложения операционная система создаёт так называемое консольное окно, через которое идёт весь ввод-вывод программ. Внешне это напоминает работу в операционной системе в режиме командной строки, когда ввод-вывод представляет собой поток символов.

Заготовка консольной программы

```
using System;
namespace ConsoleApplication1
{
class Class1
{
//[STAThread]
static void ]Main(string[ ] args)
{
//основное тело программы
Console.WriteLine(“Любая строка текста”);   }
}
}
```

Алфавит и лексемы

Алфавит C# включает:

- *буквы (латинские и национальных алфавитов) и символ подчеркивания (_) который употребляется наряду с буквами;*
- *цифры;*
- *специальные символы, например +, *, { и &;*
- *пробельные символы (пробел и символы табуляции);*
- *символы перевода строки.*
- Из символов составляются более крупные строительные блоки: лексемы, директивы препроцессора и комментарии.

Лексема (token) - это минимальная единица языка, имеющая самостоятельный смысл. Существуют следующие виды лексем:

- *имена (идентификаторы);*
- *ключевые слова;*
- *знаки операций;*
- *разделители;*
- *литералы (константы).*

Директивы препроцессора пришли в C# из его предшественника - языка C++. Препроцессором называется предварительная стадия компиляции, на которой формируется окончательный вид исходного текста программы.

Оператор задаёт законченное описание некоторого действия, данных или элемента программы. Например: *int a*; // это – оператор описания целочисленной переменной *a*.

Ключевые слова C#

break
checked
default
ehum
finally
goto
interface
namespace
out
public
sealed
string
true
unchecked
void.bool
char
decimal

Операции C#

Операции делятся на ***унарные, бинарные*** и ***тернарную*** по количеству участвующих в них операндов. Один и тот же знак может интерпретироваться по-разному в зависимости от контекста. Все знаки операций, за исключением [ ], ( ) и ? :, представляют собой отдельные лексемы.

Разделители используются для разделения или, наоборот, группирования элементов. Примеры разделителей: скобки, точка, запятая. Ниже перечислены знаки операций и разделители, использующиеся в C#:

- первичные { } [ ] ( ) . , : ; ->
- унарные + - ++ --
- бинарные + - * / %
- битовые сдвига << >> и поразрядные логические ~ | & **^**
- отношения == != < > <= >=
- логические ! && ||
- тернарная ? :
- присваивания = += -= *= /= %= битовые <<= >>= &= **^= |=**

Пример применения поразрядных логических операций

```
using System;
namespace ConsoleApplicationl
{  class Classl
{  static void Main() {
Console.WriteLine( 6 & 5 );    // Результат 4
Console.WriteLine( 6 | 5 );    // Результат 7
Console.WriteLineC 6 ^ 5 );    / Результат 3 } } }
```

6&5=110&101=100=4 (истина - только обе единицы)

6|5=110|101=111=7 (ложь- только оба нули)

6^5=110^101=011=3 (разные – единица, одинаковые-нули)

Операции СДВИГА

a=3=011 a<<1=110=6 (освободившиеся позиции заполняются нулями)

a=3=011 a<<2= 1100 =12

b=9==0.1001 b>>1=4=0.0100 (число положительное, освобождающиеся разряды знаковым разрядом - нулем)

Операция ДЕЛЕНИЯ

11/4(целые)=целочисленное деление (остаток не учитывается)=2

11/4 (вещественные)=вещественное деление (с остатком)=2.75

11%4 = 3

Операция ОТРИЦАНИЯ

Пусть a=3=0.011 Тогда ~a=1.100=-4 (0.100 прямой, 1.011 обратный, 1.100 дополнительный)

Классификация типов данных в C#

Классификация типов данных в C#

Встроенные типы языка C#

Название	Ключевое слово	Тип .NET	Диапазон	Описание	Размер, битов
Логический тип	bool	Boolean	true, false		
Целые типы	sbyte	SByte	-128..127	Со знаком	8
	byte	Byte	0..255	Без знака	8
	short	Int16	-32768..32767	Со знаком	16
	ushort	UInt16	0..65535	Без знака	16
	int	Int32	-2*10^9..2*10^9	Со знаком	32
	uint	UInt32	0..4*10^9	Без знака	32
	long	Int64	-9*10^18..9*10^18	Со знаком	64
	ulong	UInt64	0..18*10^18	Без знака	64
Символьный тип	char	Char	U+0000..U+ffff	Unicode-символ	16
Вещественные типы	float	Single	1.5*10^-45..3.4*10^38	7 цифр	32
	double	Double	5.0*10^-324..1.7*10^308	15-16 цифр	64
Финансовый тип	decimal	Decimal	1.0*10^-28..7.9*10^28	28-29 цифр	128
Строковый тип	string	String	Длина ограничена объемом доступной памяти	Строка из Unicode-символов	
Тип object	object	Object	Можно хранить все что угодно	Всеобщий порядок	

Пример описания переменных

```
int b = 1, a = 100;
int x = b * a + 25:
        class X                    // начало описания класса X
        {
        int A;      // поле A класса X
        int B;      //  поле B класса X
        void Y()     // ------------------------------------- метод Y класса X
        {
        int C;              // локальная переменная C, область действия - метод Y
        int A;              // локальная переменная A (НЕ конфликтует с полем A)
        {                   //  ============ вложенный блок 1 ============
        int  D;         //  локальная переменная D, область действия - этот блок
        int  A; // недопустимо! Ошибка компиляции, конфликт с локальной переменной A
        C =  B;         //  присваивание переменной C поля B класса X
        C =  this.A;  //  присваивание переменной C поля A класса X
        }                   //  ============ конец блока 1 ===============
        {                   // ============ вложенный блок 2 ============
        int D;          // локальная переменная D, область действия - этот блок
        }                   // ============ конец блока 2 ===============
        }                       //--------------------------------- конец описания метода Y класса X
        }                           // конец описания класса X
```

Математические функции – класс Math

Имя метода	**Описание**	**Пояснения**
Abs	Модуль	|x|, записывается как Abs(x)
Acos	Арккосинус	Acos(double x), угол задается в радианах
Asin	Арксинус	Asin(double x)
Atan	Арктангенс	Atan(double x)
Atan2	Арктангенс	Atan2(double x,double y) –угол, тангенс которого есть результат деления y на x
BigMul	Произведение	BigMul(int x,int y)
Celling	Округление до большего целого	Celling(double x)
Cos	Косинус	Cos(double x)
Cosh	Гиперболический косинус	Cosh(double x)
DivRem	Деление и остаток	DivRem(x,y,rem)
E	Число e	2,71828182845905
Exp	Экспонента	Exp(x)
Floor	Округление до меньшего целого	Floor(double x)
IEEERemainder	Остаток от деления	IEEERemainder(double x, double y)
Log	Натуральный логарифм	Log(x)
Log10	Десятичный логарифм	Log10(x)
Max	Максимум из 2-х чисел	Max(x,y)

Математические функции – класс Math

Имя метода	Описание	Пояснения
Min	Минимум из 2-х чисел	Min(x,y)
PI	Значение числа пи	3,14159265358979
Pow	Возведение в степень	Pow(x,y) – x в степени y
Round	Округление	Round(3.1)=3 Round(3.8)=4
Sign	Знак числа	
Sin	Синус	Sin(double x)
Sinh	Гиперболический синус	Sinh(double x)
Sqrt	Квадратный корень	Sqrt(x)
Tan	Тангенс	Tan(double x)
Tanh	Гиперболический тангенс	Tanh(double x)

Пример линейной программы расчета по заданной формуле

$$y = \sqrt{\pi \cdot x} - e^{0,2\sqrt{\alpha}} + 2tg2\alpha + 1,6 \cdot 10^3 \cdot \log_{10} x^2$$

```
using System;
namespace ConsoleApplicationl
{    class Classl
{    static void Main()
{
string buf;
Console.WriteLine("Введите x");
buf=Console.ReadLine();
double x=Convert.ToDouble(buf);

Console.WriteLine("Введите alfa");
buf=Console.ReadLine();
double a=double.Parse(buf);

double y=Math.Sqrt(Math.PI*x)-
                    Math.Exp(0.2*Math.Sqrt(a))+
                    2*Math.Tan(2*a)+
                   1.6e3*Math.Log10(Math.Pow(x,2));
Console.WriteLine("Для x= {0} и alfa ={1}", x,a);
Console.WriteLine("Результат =" +y);
Console.ReadKey();          }}}
```

Операторы языка C#

for (int j=1; j<5; j++) //оператор цикла {;;;}; //пустой оператор

Примеры оператора if

```
if (a<0) b=1;
if (a<b&& (a>d||a= =0) )  b=++; else {b*=a;a=0;}
if (a<b) if (a<c) m=a; else m=c; else if (b<c) m=b; else m=c;
```

switch(выражение) {

case константное_выражение_1: [операторы_1 оператор_перехода_1] ...
case константное_выражение_K: [операторы_K оператор_перехода_K]
[default: операторы_N оператор_перехода_N]}

<u>Операторов перехода, позволяющих прервать естественный порядок выполнения операторов блока, в языке C# несколько:</u>

- оператор безусловного перехода goto;
- оператор выхода из цикла break;
- оператор перехода к следующей итерации цикла continue;
- оператор возврата из функции return;
- оператор генерации исключения throw.

Операторы цикла

Цикл с предусловием while имеет формат:

while (выражение) оператор

Цикл с постусловием do имеет вид:

do оператор while выражение;

Цикл с параметром имеет следующий формат:

for (инициализация; выражение модификации;) оператор;

Цикл перебора foreach используется для просмотра всех объектов из некоторой группы данных, например *массива, списка или другого контейнера*.

В языке C# массив относится к *ссылочным типам данных*, то есть располагается только в *динамической памяти*, поэтому **создание массива начинается с выделения памяти под его элементы.** Всем элементам при создании массива присваиваются значения по умолчанию – нули для значимых типов, и null для ссылочных. Все массивы в C# имеют общий базовый **класс Array**, определенный в пространстве имен System. using System;

```
namespace ConsoleApplication1
{class Class1
{   static void Main()
{  const int n=6; int [] a= new int[n] {3,12,5,-9,8,-4};
Console.WriteLine("Исходный массив: ");
for (int i=0;i<n;++i) Console.Write("\t" +a[i]);  Console.WriteLine();
int s=0,k=0;
for (int i=0;i<n;++i)   if (a[i]<0)   {s+=a[i];  ++k;  }
Console.WriteLine("Сумма отрицательных  "+s);
Console.WriteLine("Количество отрицательных  "+k); }}}
```

Примеры описания <u>двумерных</u> массивов:

int [,] a; //элементов нет

int [,] b=new int [2,3]; //элементы равны 0

int [,] c={{1,2,3},{4,5,6}}; //new подразумевается

int [,] d=new int[,]{{1,2,3},{4,5,6}}; //размерность вычисляется

int [,] e=new int [2,3] {{1,2,3},{4,5,6}}; //избыточное описание

```
{class Class1
{static void Main()        {int [] a=  {3,12,5,18,-9,8,-4};
PrintArray ("Исходный массив:", a); //пользовательская функция (метод)
Console.WriteLine(Array.IndexOf(a,18);   Array.Sort(a);
PrintArray ("Отсортированный массив:", a);
Console.WriteLine(Array.BinarySearch(a,18));        }//конец Main()
public    static void PrintArray(string s, int[] a)
    {Conole.WriteLine(s);
    for (int i=0;i<a.Length;++i)
    Console.Write("\t"+a[i]);Console.WriteLine();} //конец функции
                }//конец класса Class1
```

Основные элементы класса Array

Свойство	Описание
Length	Количество элементов массива (по всем размерностям)
Rank	Количество размерностей массива
BinarySearch	Двоичный поиск в отсортированном массиве
`Clear`	Выполняет начальную инициализацию элементов. В зависимости от типа элементов устанавливает значение `0` для арифметического типа, `false` - для логического типа, `Null` для ссылок, "" - для строк.
Copy	Копирование части или всего массива в другой массив.
IndexOf	Поиск индекса первого вхождения элемента в одномерный массив.
LastIndexOf	Поиск индекса последнего вхождения элемента в одномерный массив.
Reverse	Изменение порядка следования элементов на обратный.
Sort	Сортировка элементов одномерного массива
`CopyTo`	Копируются все элементы одномерного массива в другой одномерный массив, начиная с заданного индекса
`GetLength`	Возвращает число элементов массива по указанному измерению.
`GetValue, SetValue`	Возвращает или устанавливает значение элемента массива с указанными индексами.

Оператор foreach для работы с массивами

```
int [] a= {3,12,5,-9,8,-4};
Console.WriteLine("Исходный массив: ");
foreach (int x in a)    Console.Write("\t" +x);
Console.WriteLine();
```

Использование оператора foreach для вывода на экран двумерного массива имеет вид:

```
foreach (int [] x in a)
{foreach (int y in x) Console.Write("\t"+y);
                Console.WriteLine();}
```

Для получения псевдослучайной последовательности чисел необходимо сначала создать экземпляр класса с помощью конструктора: Random a = new Random();

Название	Описание
Next ()	Возвращает целое положительное число во всем положительном диапазоне типа int
Next(макс)	Возвращает целое положительное число в диапазоне [0, макс]
Next (мин, макс)	Возвращает целое положительное число в диапазоне [мин, макс]
NextBytes(массив)	Возвращает массив чисел в диапазоне [0, 255]
NextDouble()	Возвращает вещественное положительное число в диапазоне [0, 1)

Пример работы с генератором псевдослучайных чисел

```
using System:
namespace ConsoleApplicationl
{    class Classl
{    static void Main()
{ Random a = new Random();
   Random b = new Random(1);
const int n = 10;
Console.WriteLine( "\n   Диапазон [0, 1]:" );
for ( int i = 0;  i < n; ++i  )   Console.Write( "{0 ,6:0.##}", a.NextDouble()
);
Console.WriteLine( "\n   Диапазон [0, 1000]:" );
for ( int i = 0; i < n; ++i  )    Console.Write( "     " + b.Next(1000) );
Console.WriteLine( "\n   Диапазон [-10, 10]:" );
for ( int i = 0;  i < n; ++i  ) Console.Write( "     " + a.Next(-10, 10) );
Console.WriteLine(  "\n   Массив [0, 255]:"  );
byte[] mas = new byte[n];
a.NextBytes( mas );
for (int i = 0; i < n; ++i) Console.Write( " " + mas[i] ); } } }
```

Для работы со строками в языке C# существует несколько возможностей:

- используя символьный тип **char(символ)**, который соответствует классу **System.Char**, для организации строки – массив символов типа char, который соответствует классу **System.Array**
- используя строки типа **String, е**му соответствует базовый класс **System.String** библиотеки .NET. , его объекты должны оставаться неизменны.
- используя строки типа **StringBuilder.** В этом случае для работы со строками применяется класс **StringBuilder**, определенный в пространстве имен System. Text и позволяющий изменять значение своих экземпляров.

Функции (методы) в языке C#

Метод, не возвращающий значение – вызывается как отдельный оператор.

Метод, возвращающий значение - вызывается в правой части оператора присваивания.

Если метод возвращает значение, оно передается в точку вызова. Если метод имеет тип void управление передается на оператор, следующий после вызова.

Параметры, описываемые в заголовке метода, определяют множество значений аргументов, которые можно передавать в метод. Список аргументов при вызове метода должен соответствовать списку параметров по количеству, порядку следования и типу. Для каждого параметра должны задаваться его тип и имя. Существует **два способа** передачи параметров: **по значению и по ссылке.**

При передаче *по значению* метод получает копии аргументов, и операторы внутри метода работают с этими копиями. Доступа к исходным значениям аргументов у метода нет, и, следовательно, нет возможности их изменять.

При передаче *по ссылке (по адресу)* метод получает адреса аргументов, осуществляет доступ к ячейкам памяти по этим адресам, следовательно может изменять значения аргументов, модифицируя параметры. В языке C# для обмена данными между вызывающей и вызываемой функциями есть 4 типа параметров:

- **параметры-значения** void func(int x)
- **параметры-ссылки** void func(**ref** int x)
- **выходные параметры** static void p(int a, out int b, out int c)
- **параметры-массивы**

Одним из основных достоинств языка C# является его схема работы с памятью: *автоматическое выделение памяти под объекты* и *автоматическая уборка мусора*. При этом невозможно обратиться по несуществующему адресу памяти или выйти за границы массива, что делает программы более надежными и безопасными.

В некоторых случаях возникает необходимость работать с адресами памяти непосредственно, например, при взаимодействии с операционной системой, написании драйверов и др. Такую возможность представляет так называемый *небезопасный* (unsafe) *код*, выполнение которого *среда CLR не контролирует*.

Указатели являются *отдельной категорией* типов данных. Они **не наследуются** от типа object, и **преобразование между типом object и типами указателей невозможно**. В частности, для них не выполняется упаковка и распаковка. Однако допускаются преобразования между разными типами указателей, а так же указателями и целыми.

Основными операциями с указателями **являются * и &.** Обе эти операции являются унарными, т. е. имеют один операнд, перед которыми они ставятся. Операция **&** соответствует операции "**взять адрес**". Операция ***** соответствует словам "**значение, расположенное по указанному адресу**".

Операции с указателями

Операция	Описание
*	**Разадресация – получение значения**, которое находится по адресу, хранящемуся в указателе.
->	Доступ к элементу структуры через указатель
[]	Доступ к элементу массива через указатель
&	**Получение адреса** переменной
++,--	Увеличение и уменьшение значения указателя **на один адресуемый элемент**
+,-	Сложение с целой величиной и вычитание указателей
= =,!=,<>,<=,>=	Сравнение адресов, хранящихся в указателях. Выполняется как сравнение беззнаковых целых величин.
stackalloc	Выделение памяти в стеке под переменную, на которую ссылается указатель