Базовые понятия. Знакомство с C#

1) Алгоритм, код, программа
2) Парадигмы программирования
3) Знакомство с **MS Visual Studio**
4) Первая программа на языке **C#**
5) Простые типы данных
6) Преобразования простых типов
7) Базовые операции, выражения

Процесс

Например:

- **В папке хранятся файлы. Необходимо автоматически удалить все файлы, размер которых меньше 2 кб.**
- **В библиотеке хранится информация о книгах. Необходимо отсортировать книги по авторам и названию в алфавитном порядке**
- **…**

Например:

1. **Ввести полный путь к папке**
2. **Если папка пуста, выдать сообщение «Папка пуста!» и завершить работу**
3. **Перейти к очередному файлу в папке и узнать его размер**
4. **Если размер меньше 2 кб, то удалить его и нарастить счетчик удаленных файлов на 1**
5. **Повторять шаги 3-4, пока не будут просмотрены все файлы**
6. **Вывести на экран значение счетчика удаленных**

```
static void Main(string[] args)
{
    Console.WriteLine( "Please, enter the path:" );
    string path = Console.ReadLine();

    // получаем массив всех файлов в указанной папке
    string[] files = Directory.GetFiles( path );

    // если массив файлов пуст, то выводим сообщение и выходим
    if ( files.Length == 0 )
    {
        Console.WriteLine( "The folder is empty!" );
        Console.ReadKey();
        return;
    }

    // ...
```

Определение алгоритма

Алгоритм (algorithm) — это формально описанная вычислительная процедура, получающая **исходные данные** (input), называемые также входом алгоритма или его аргументом, и выдающая **результат** вычислений на выход (output).

Алгоритмы строятся для решения тех или иных **вычислительных задач** (computational problems). Формулировка задачи описывает, каким требованиям должно удовлетворять решение задачи, а алгоритм, решающий эту задачу, находит объект, этим требованиям удовлетворяющий.

(c) Т.Кормен «Алгоритмы. Построение и анализ»

Алгори́тм
набор инструкций, описывающих порядок действий исполнителя для достижения результата решения задачи за конечное число действий

(c) https://ru.wikipedia.org/wiki/Алгоритм

Свойства алгоритма

- Дискретность
- Детерминированность (определённость)
- Понятность
- Завершаемость (конечность)
- Массовость (универсальность)
- Результативность

Алгоритм содержит ошибки, если приводит к получению неправильных результатов либо не даёт результатов вовсе

Алгоритм не содержит ошибок, если он даёт правильные результаты для любых допустимых исходных данных

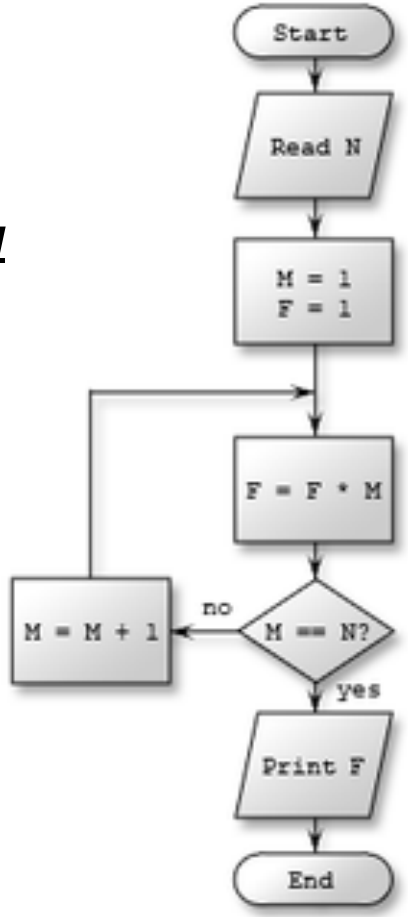
Виды представления алгоритма

- ***Словесное, языковое, формульно-словесное***

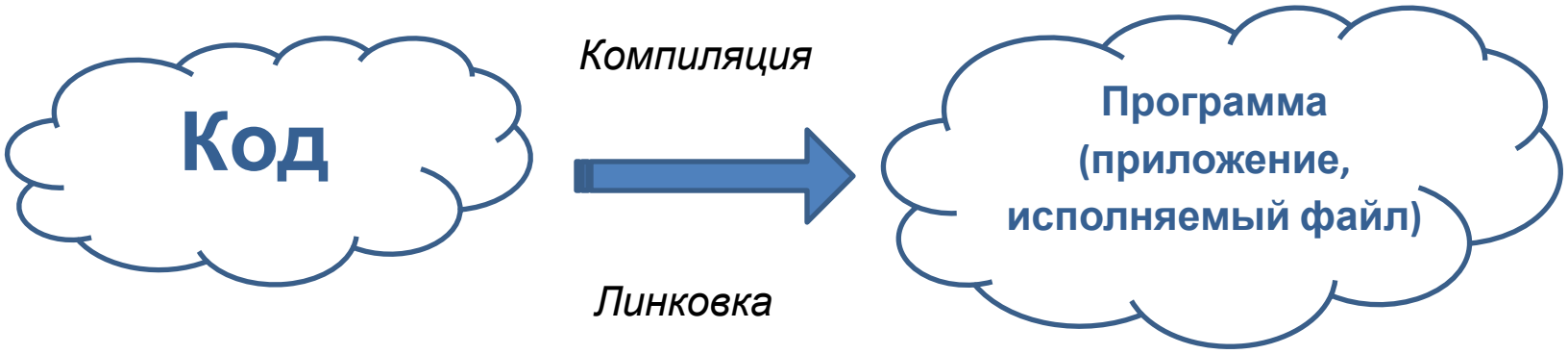
- ***Псевдокод***

```
MERGE-SORT(A, p, r)
1  if p < r
2     then q ← ⌊(p + r)/2⌋
3          MERGE-SORT(A, p, q)
4          MERGE-SORT(A, q + 1, r)
5          MERGE(A, p, q, r)
```

- ***Блок-схемы***



Процесс (продолжение)




```
static void Main(string[] args)
{
    Console.WriteLine( "Please, enter the path:" );
    string path = Console.ReadLine();

    // получаем массив всех файлов в указанной папке
    string[] files = Directory.GetFiles( path );

    // если массив файлов пуст, то выводим сообщение и выходим
    if ( files.Length == 0 )
    {
        Console.WriteLine( "The folder is empty!" );
        Console.ReadKey();
        return;
    }

    // ...
```


```
4D 5A 90 00 03 00 00 00|
B8 00 00 00 00 00 00 00|
00 00 00 00 00 00 00 00|
00 00 00 00 00 00 00 00|
0E 1F BA 0E 00 B4 09 CD|
69 73 20 70 72 6F 67 72|
74 20 62 65 20 72 75 6E|
6D 6F 64 65 2E 0D 0D 0A|
50 45 00 00 4C 01 03 00|
00 00 00 00 E0 00 02 01|
00 08 00 00 00 00 00 00|
00 40 00 00 00 00 40 00|
04 00 00 00 00 00 00 00|
00 80 00 00 00 02 00 00|
00 00 10 00 00 10 00 00|
00 00 00 00 10 00 00 00|
0C 2D 00 00 4F 00 00 00|
```

Компиляция

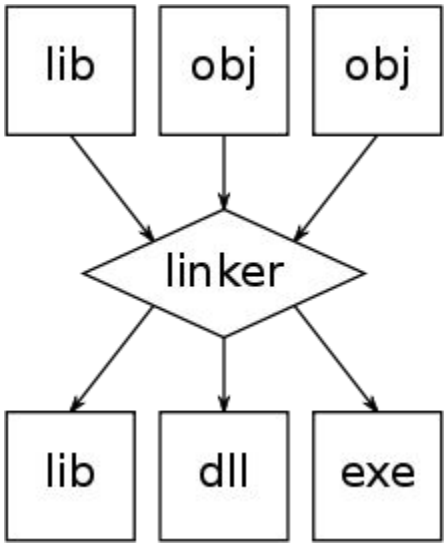
Процесс компиляции состоит из следующих этапов:

1. Лексический анализ текстового кода.
2. Синтаксический (грамматический) анализ кода.
3. Семантический анализ кода.
4. Оптимизация как текстового, так и машинного кода.
5. Генерация машинного кода.

Отдельный класс компиляторов - *интерпретаторы*

Линковка (компоновка)

Для большинства компиляторов, один объектный файл является результатом компиляции одного файла с исходным кодом. Если программа собирается из нескольких объектных файлов, компоновщик собирает эти файлы в единый исполняемый модуль (exe файл), вычисляя и подставляя адреса вместо символов, в течение *времени компоновки* (статическая компоновка) или во *время исполнения* (динамическая компоновка).



Задача, грубо говоря:

связать воедино машинные коды нескольких модулей из нескольких файлов с кодом

Парадигмы программирования

- Императивное
  - **Объектно-ориентированное**
  - Аспектно-ориентированное
  - **Процедурное**
  - Модульное
  - Обобщенное
- Декларативное
  - **Функциональное**
  - Логическое
  - Потоки данных
- Параллельное

Язык программирования

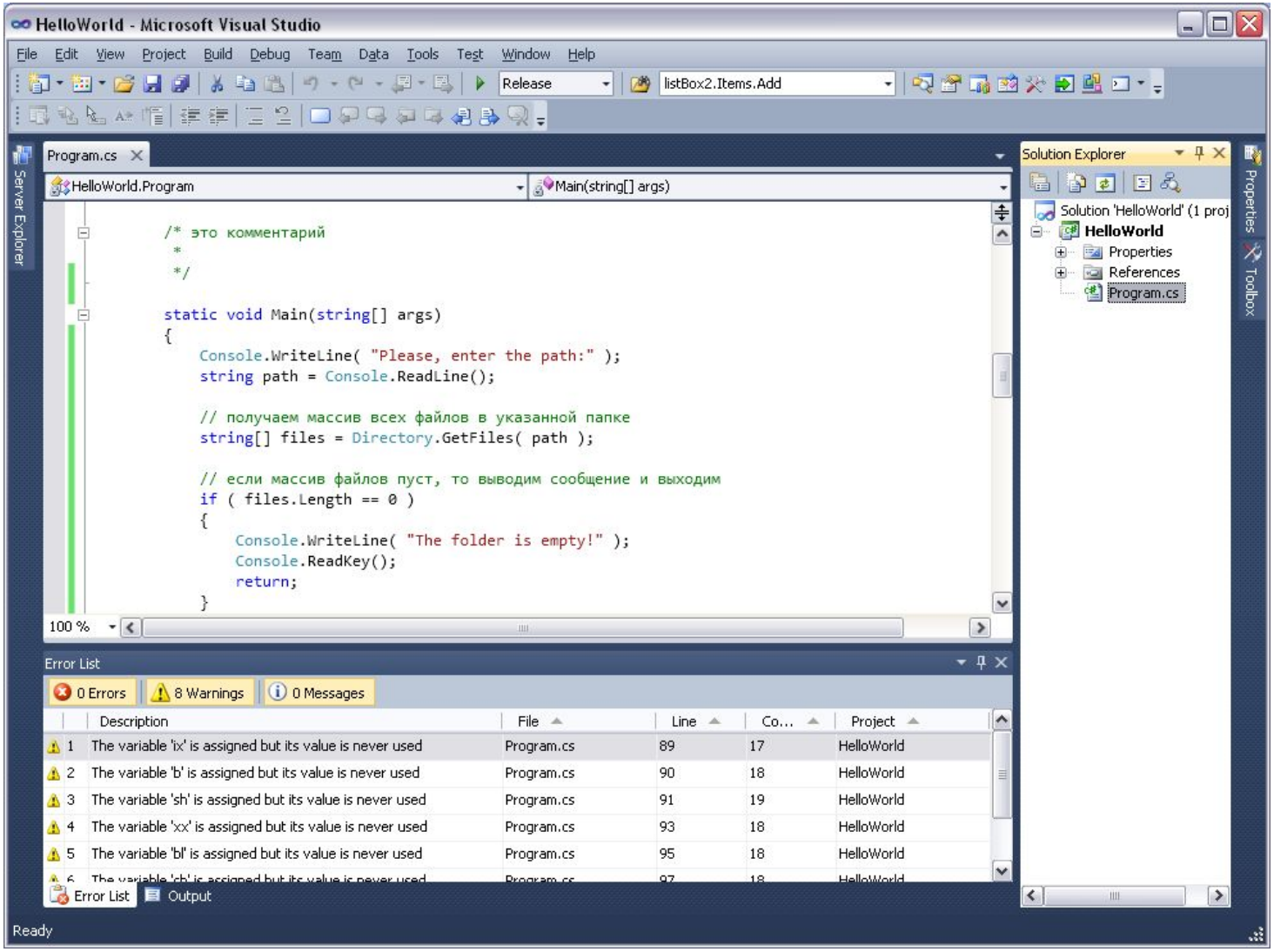
Язы́к программи́рования

формальная знаковая система, предназначенная для записи компьютерных программ. Язык программирования определяет набор лексических, синтаксических и семантических правил, задающих внешний вид программы и действия, которые выполнит исполнитель (компьютер) под её управлением

- Синтаксис
- Семантика
- Парадигма
- Структуры данных
- Типы данных

Язык программирования

Знакомство с MS Visual Studio



http://www.microsoft.com/ru-ru/softmicrosoft/VisualStudioExpress.aspx

Знакомство с MS Visual Studio

В рамках данного курса можно ограничиться следующими пунктами меню:

File
Работа с проектами и файлами (создание, открытие, сохранение, печать)

Edit
Функции редактирования текста и навигации по проекту

View
Просмотр различных окон среды

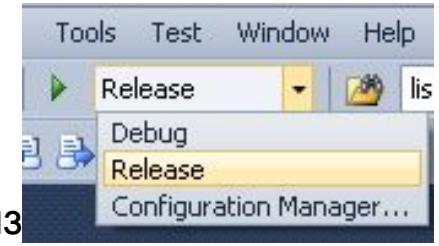
Project
Управление проектом (добавление компонент, редактирование настроек проекта)

Build
Компиляция и линковка (сборка и пересборка) проекта

Debug
Все действия, связанные с отладкой приложения (установить/снять брейкпойнт, начать пошаговую отладку, запустить приложение без отладки)

Конфигурации Debug и Release

Главное различие состоит в назначении:
конфигурация **Debug** предназначена для компиляции на этапе разработки и отладки программы, а **Release** - для сборки программы и последующего её использования пользователями программы.



Поэтому:

В конфигурации Release удаляется отладочная информация из исполняемого exe-файла. Это приводит к уменьшению размера исполняемого exe-файла (обычно в несколько раз).

Исключаются дополнительные проверки. Например, инициализированы переменные или нет. В конфигурации Release программа может работать значительно быстрее, но и могут возникнуть новые ошибки, если код недостаточно хорошо написан и протестирован.

Производится оптимизация по уменьшению времени выполнения.

http://kaktusenok.blogspot.com/2013/06/visual-studio-debug-release.html

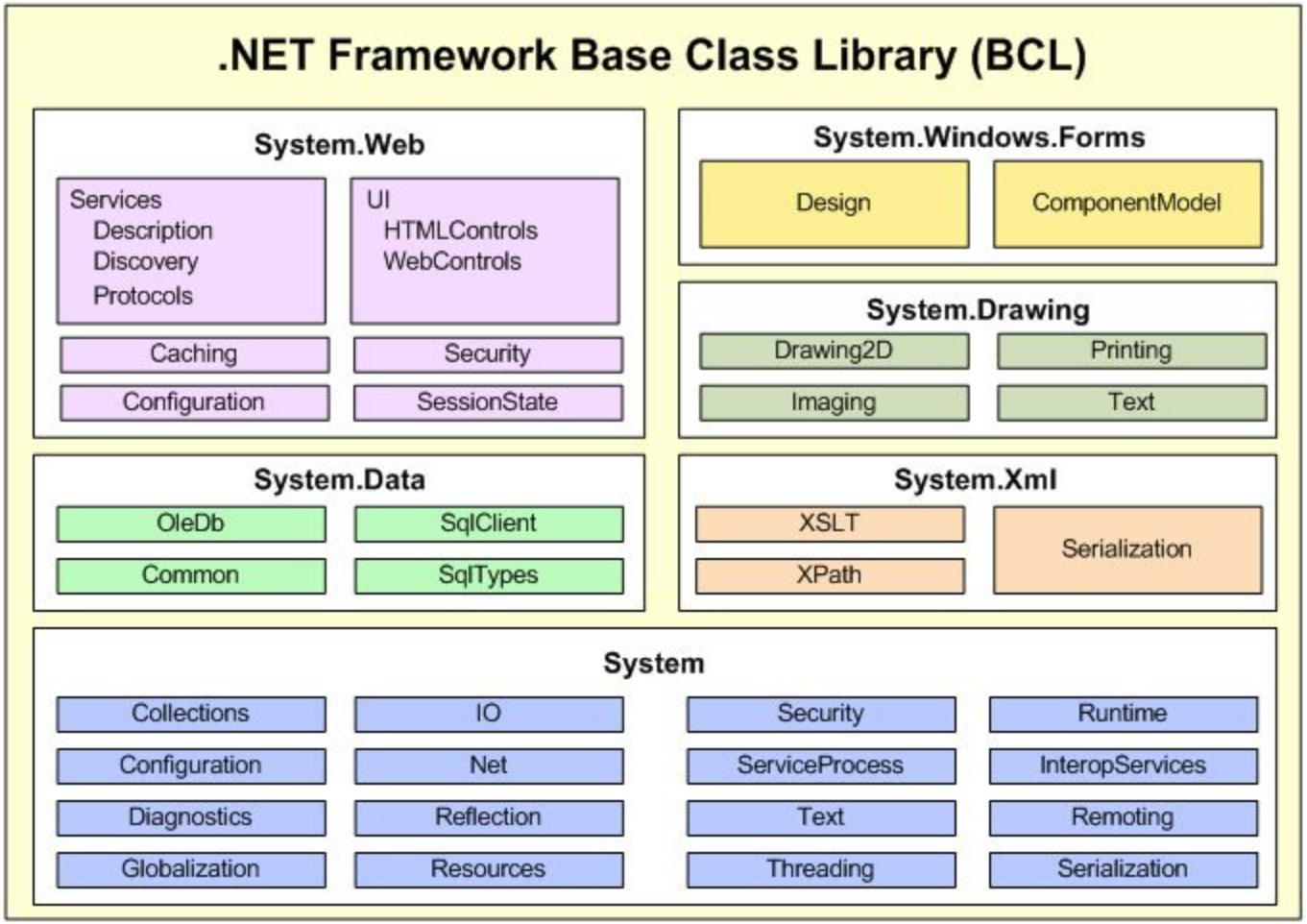
Знакомство с .NET

Назначение .NET Framework — служить средой для поддержки разработки и выполнения сильно распределенных компонентных приложений. Она обеспечивает совместное использование разных языков программирования, а также безопасность, переносимость программ и общую модель программирования для платформы Windows. Что же касается взаимосвязи с C#, то среда .NET Framework определяет два очень важных элемента. Первым из них является *общеязыковая среда выполнения* (Common Language Runtime — CLR). Это система, управляющая выполнением программ. Среди прочих преимуществ — CLR как составная часть среды .NET Framework поддерживает многоязыковое программирование, а также обеспечивает переносимость и безопасное выполнение программ.

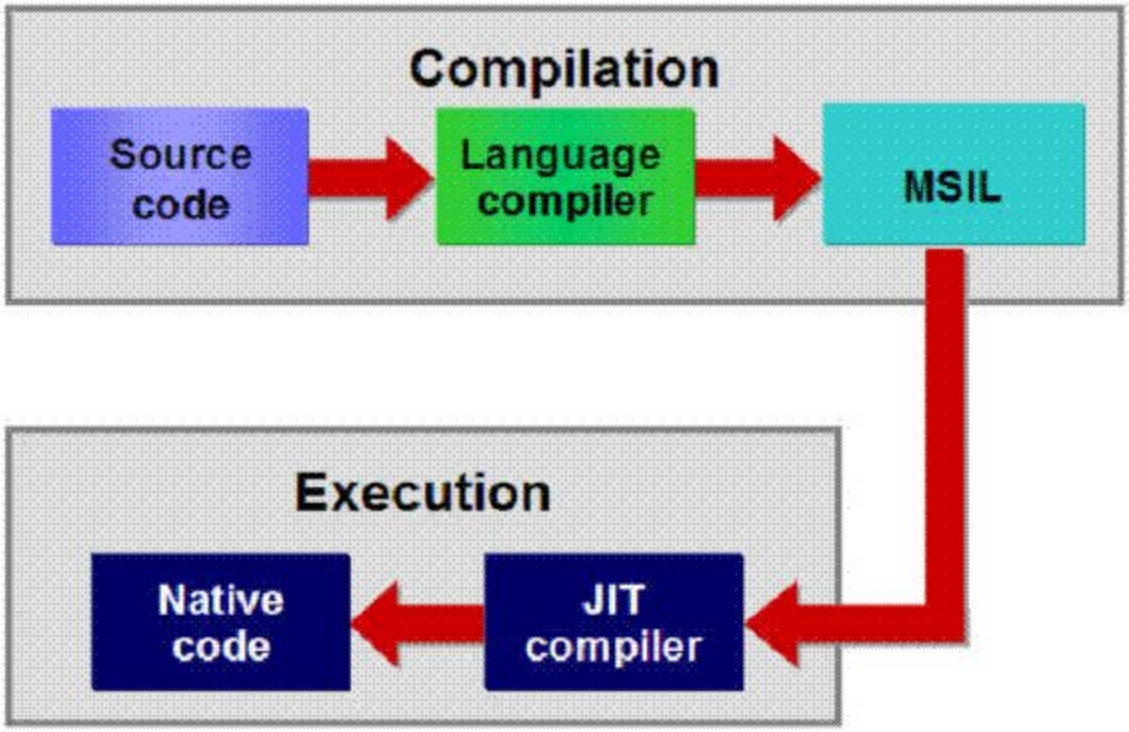
Вторым элементом среды .NET Framework является *библиотека классов*. Эта библиотека предоставляет программе доступ к среде выполнения. Так, если требуется выполнить операцию ввода-вывода, например вывести что-нибудь на экран, то для этой цели используется библиотека классов .NET.

(c) Г.Шилдт «C# 4.0. Полное руководство»

Базовая библиотека .NET



Компиляция приложений .NET



Знакомство с C#

```
/* ============================================================================================
 * Подключение необходимых пространств имен
 * Здесь подключаем System, т.к. в нем объявлены все классы библиотеки .NET
 ============================================================================================*/
using System;

namespace HelloWorld
{
    // программа оформляется в виде объекта класса (т.к. C# - это объектно-ориентированный язык)
    // Объектно-ориентированное программирование будет изучаться на 2-ом курсе
    class Program
    {
        // код основной программы здесь (главная процедура)

        static void Main(string[] args)
        {
            Console.WriteLine("Hello world!");     // вывод на экран сообщения "Hello world!"
            Console.ReadKey();                     // ожидание нажатия любой клавиши пользователем
        }
    }
}
```

C#

Название «Си шарп» (от англ. sharp — диез) происходит от музыкальной нотации, где знак диез, означает повышение соответствующего ноте звука на полутон, что аналогично названию языка C++, где «++» обозначает инкремент переменной.

Название так же является игрой с цепочкой C → C++ → C++++(C#), так как символ «#» можно составить из 4х знаков «+»

Реализации C#

- Реализация C# в виде компилятора **csc.exe** включена в состав **.NET Framework** (включая .NET Micro Framework, .NET Compact Framework и его реализации под Silverlight и Windows Phone 7).
- В составе проекта **Rotor** (Shared Source Common Language Infrastructure) компании Microsoft.
- Проект **Mono** включает в себя реализацию C# с открытым исходным кодом.
- Проект **DotGNU** также включает компилятор C# с открытым кодом.
- **DotNetAnywhere** — ориентированная на встраиваемые системы реализация CLR, поддерживает практически всю спецификацию C# 2.0.

! Не путать язык, платформу и среду разработки
!

Переменные

Переменная — это именованная область памяти, для которой может быть установлено значение. Она называется переменной потому, что ее значение может быть изменено по ходу выполнения программы. Иными словами, содержимое переменной подлежит изменению и не является постоянным.

```
// объявили переменную для хранения возраста (целое число, тип данных - int)
// и инициализировали ее значением 24 (присвоили ей значение)
int age = 24;
```

4 байта памяти


```
// здесь также целое число, но тип данных - short
short value = -7;
```

2 байта памяти

Именование переменных

Идентификатор (имя переменной) должно начинаться с буквы или символа _, за которыми могут идти буквы, символы _ и цифры.

Идентификатор не может совпадать с одним из ключевых слов C# :

abstract	as	base	bool	break
byte	case	catch	char	checked
class	const	continue	decimal	default
delegate	do	double	else	enum
event	explicit	extern	false	finally
fixed	float	for	foreach	goto
if	implicit	in	int	interface
internal	is	lock	long	namespace
new	null	object	operator	out
override	params	private	protected	public
readonly	ref	return	sbyte	sealed
short	sizeof	stackalloc	static	string
struct	switch	this	throw	true
try	typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual	void
volatile	while			

Как хранить целые числа

```
/* ================================================== ЦЕЛОЧИСЛЕННЫЕ ЗНАКОВЫЕ ТИПЫ ДАННЫХ
 ================================================================================================*/

sbyte smallNumber = -12;         // целое число, занимает 1 байт, диапазон: -128..127
short biggerNumber = 7000;       // целое число, занимает 2 байта, диапазон: -32768..32767
int bigNumber = -100000;         // целое число, занимает 4 байта, диапазон: -2147483648..2147483647
long hugeNumber = 500000000000; // целое число, занимает 8 байт, диапазон: -9223372036854775808..9223372036854775807


/* ================================================ ЦЕЛОЧИСЛЕННЫЕ БЕЗЗНАКОВЫЕ ТИПЫ ДАННЫХ
 ================================================================================================*/

byte smallNumberU = 12;             // целое число, занимает 1 байт, диапазон: 0..255
ushort biggerNumberU = 7000;        // целое число, занимает 2 байта, диапазон: 0..65535
uint bigNumberU = 100000;           // целое число, занимает 4 байта, диапазон: 0..4294967295
ulong hugeNumberU = 500000000000;   // целое число, занимает 8 байт, диапазон: 0..18446744073709551615
```

Другие простые типы данных

```
double pi = 3.1415738;          // вещественное число, занимает 8 байт
float coefficient = 0.35f;      // вещественное число, занимает 4 байта

decimal newPrice = 25.50M;      // тип данных для представления финансовых величин

char symbol = 'A';              // символ

bool indicator = false;         // логический тип (истина или ложь, true или false)

string message = "Error!";      // строка (размер занимаемой памяти зависит от количества символов)
```

```
double x = 12.555;
double y = -12.444;

double sum = x + y;

Console.WriteLine( sum );   // выведет 0.110999999999 из-за ошибок округления
```

Размерность типов данных

```
Console.WriteLine( "double: " + sizeof(double) + " bytes" );
Console.WriteLine();
      Console.WriteLine( "float: " + sizeof(float) + " bytes");
      Console.WriteLine();
      Console.WriteLine( "decimal: " + sizeof(decimal) + " bytes");
      Consol
      Consol
      Consol
      Consol
      Consol
      Consol
      Consol
      Consol
      Consol
      Consol
      Consol
```

Пример использования переменных

Задача.
Замер земельного участка показал, что ширина равна 105,36 м, а высота – 87.32 м.
Написать программу, которая посчитает площадь и периметр этого участка и выведет результаты на экран.

```
class Program
{
    static void Main( string[] args )
    {
        double width = 105.36;          // в переменной width храним ширину
        double height = 87.32;          // в переменной height храним высоту

        double area = width * height;               // рассчитываем площадь прямоугольника

        double perimeter = (width + height) * 2;    // рассчитываем периметр прямоугольника

        Console.WriteLine( "The area of the rectangle equals to " + area );
        Console.WriteLine( "The perimeter of the rectangle equals to " + perimeter );

        Console.ReadKey();
    }
```

Еще об инициализации

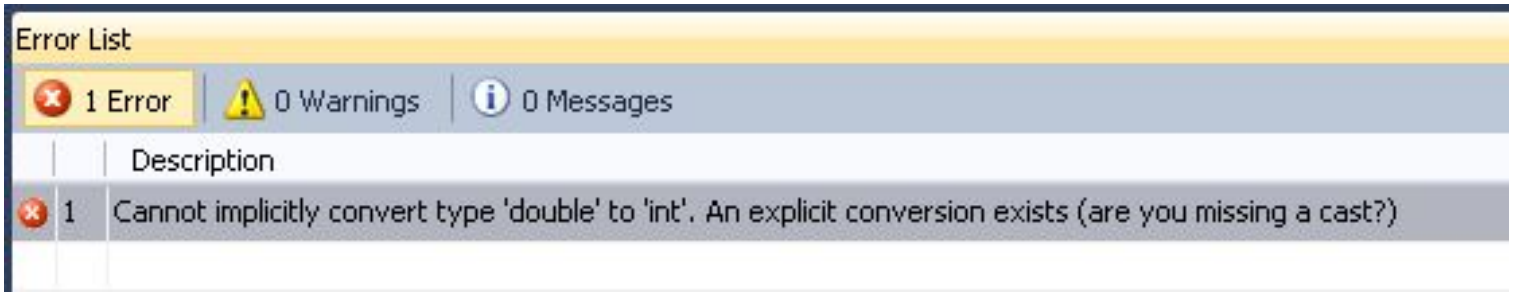
```
// объявление нескольких переменных и инициализация первых двух
int x1 = 20, x2 = 50, y1, y2;

// динамическая инициализация - переменной diffX присваиваем результат операции x2 - x1
int diffX = x2 - x1;

// а так нельзя - компилятор требует, чтобы переменные были инициализированы
int diffY = y2 - y1;
```

Неявная типизация

Как пояснялось выше, все переменные в C# должны быть объявлены. Как правило, при объявлении переменной сначала указывается тип, например `int` или `bool`, а затем имя переменной. Но начиная с версии C# 3.0, компилятору предоставляется возможность самому определить тип локальной переменной, исходя из значения, которым она инициализируется. Такая переменная называется *неявно типизированной*.

Неявно типизированная переменная объявляется с помощью ключевого слова `var` и должна быть непременно инициализирована. Для определения типа этой переменной компилятору служит тип ее инициализатора, т.е. значения, которым она инициализируется.

```
var symbol = 'X';    // компилятор автоматически выведет тип char для переменной symbol

var N = 500;         // компилятор автоматически выведет тип int для переменной N

N = 23.56;           // нельзя так! компилятор уэе назначил переменной N тип int
```



Форматирование вывода в консоль

Для управления форматированием числовых данных служит другая форма метода `WriteLine()`, позволяющая встраивать информацию форматирования, как показано ниже.

```
WriteLine("форматирующая строка", arg0, arg1, ... , argN);
```

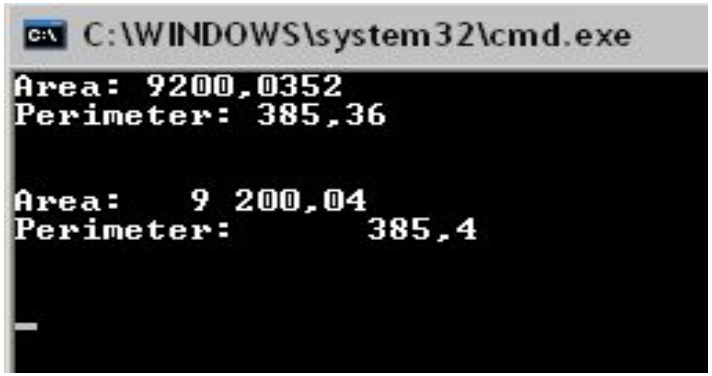
В этой форме аргументы метода `WriteLine()` разделяются запятой, а не знаком +. А *форматирующая строка* состоит из двух элементов: обычных печатаемых символов, предназначенных для вывода в исходном виде, а также спецификаторов формата. Последние указываются в следующей общей форме:

```
{argnum, width: fmt}
```

где *argnum* — номер выводимого аргумента, начиная с нуля; *width* — минимальная ширина поля; *fmt* — формат. Параметры *width* и *fmt* являются необязательными.

Форматирование вывода (примеры)

```
class Program
{
    static void Main( string[] args )
    {
        double width = 105.36;          // в переменной width храним ширину
        double height = 87.32;          // в переменной height храним высоту

        double area = width * height;               // рассчитываем площадь прямоугольника

        double perimeter = (width + height) * 2;    // рассчитываем периметр прямоугольника


        Console.WriteLine( "Area: {0} \nPerimeter: {1}\n\n", area, perimeter );

        Console.WriteLine( "Area: {0, 10: #,###.##} \nPerimeter: {1, 10: #,###.#}\n\n", area, perimeter );


        Console.ReadKey();
    }
```



Управляющие символы

Управляющая последовательность	Описание
\a	Звуковой сигнал (звонок)
\b	Возврат на одну позицию
\f	Перевод страницы (переход на новую страницу)
\n	Новая строка (перевод строки)
\r	Возврат каретки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\0	Пустой символ
\'	Одинарная кавычка
\"	Двойная кавычка
\\	Обратная косая черта

Игнорирование управляющих символов

```
Console.WriteLine( @"Area: {0} \nPerimeter: {1}\n\n" , area, perimeter);
```

```
Area: 9200,0352 \nPerimeter: 385,36\n\n
```

Константы

```
            const double g = 9.80665;       // ускорение свободного падения

            const double G = 6.67384e-11;   // гравитационная постоянная

            Console.WriteLine(
@"g = {0,15}
G = {1,15}",  g, G );
```

Преобразования типов

Автоматическое преобразование типов

Когда данные одного типа присваиваются переменной другого типа, *неявное* преобразование типов происходит автоматически при следующих условиях:

- оба типа совместимы;
- диапазон представления чисел целевого типа шире, чем у исходного типа.

Если оба эти условия удовлетворяются, то происходит *расширяющее преобразование*. Например, тип `int` достаточно крупный, чтобы вмещать в себя все действительные значения типа `byte`, а кроме того, оба типа, `int` и `byte`, являются совместимыми целочисленными типами, и поэтому для них вполне возможно неявное преобразование.

Автоматические преобразования типов

```
byte byte_N = 5;            // Инициализируем байтовую переменную числом 5

// ======================== Преобразования "Целое число <-> целое число"

short short_N = byte_N;     // можно, т.к. short может "вместить в себя" byte

int int_N = byte_N;         // можно
int_N = short_N;            // можно

byte_N = short_N;           // нельзя (автоматически), т.к. 1 байт меньше 2 байт (спасибо, кэп)

byte_N = (byte)short_N;     // можно, но если число выходит за диапазон 0..255, то результат будет другой


// ======================== Преобразования "Вещественное число <-> целое число"

double double_N = int_N;    // можно

float float_N = int_N;      // можно

int x = float_N;            // нельзя автоматически, хотя размер одинаков
```

Приведение (конвертация) типов

```
// ======================== Конвертация "Вещественное число -> целое число"
float floatX = 15.765f;
int x = (int)floatX;                // приведение вещ.числа к целому (способ 1)
x = Convert.ToInt32( float_N );     // приведение вещ.числа к целому (способ 2)
```

```
// ======================== Конвертации между строками и целыми типами
string stringInt = "157";
x = int.Parse( stringInt );           // приведение строки к числу (способ 1)
x = Convert.ToInt32( stringInt );     // приведение строки к числу (способ 2)
string newString = x.ToString();    // приведение к строке числа
```

Приведение (конвертация) типов

```
// ================================ Конвертации между строками и вещественными числами

string stringDouble = "2.65";

// ============================================================================== способ 1
double doubleX = Convert.ToDouble( stringDouble,
                        System.Globalization.CultureInfo.InvariantCulture );

// ============================================================================== способ 2
doubleX = double.Parse( stringDouble,
                        System.Globalization.CultureInfo.InvariantCulture );
```

Арифметические операторы

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

```
static void Main(string[] args)
{
    int x = 18, y = 12;

    int sum = x + y;
    int diff = x - y;
    int mul = x * y;
    int div = x / y;     // целая часть деления (частное)
    int mod = x % y;     // остаток от деления

    Console.WriteLine("Sum = \t\t{0}\nDifference = \t{1}\nProduct = \t{2}\nQuotient = \t{3}\nRemainder = \t{4}\n",
                      sum, diff, mul, div, mod);
```

Инкременты и декременты

```
// инкремент справа (сначала идет присваивание, а потом sum наращивается на 1)
int newSum = sum++;

Console.WriteLine( "NOT incremented sum = " + newSum );      // что здесь выведется на экран?


// инкремент слева (сначала sum наращивается на 1, а потом идет присваивание newSum)
newSum = ++sum;

Console.WriteLine("Incremented sum = " + newSum);            // и что здесь выведется на экран?

// это эквивалентно записи  sum = sum + 1
sum++;
```

```
Sum =              30
Difference =       6
Product =          216
Quotient =         1
Remainder =        6

NOT incremented sum = 30
Incremented sum = 32
```

Составной оператор присваивания

```
int number = 30;

number *= 10;       // то же самое, что  number = number * 10

number += 17;       // то же самое, что  number = number + 17
```

Поразрядные операторы

Оператор	Значение
&	Поразрядное И
|	Поразрядное ИЛИ
^	Поразрядное исключающее ИЛИ
>>	Сдвиг вправо
<<	Сдвиг влево
~	Дополнение до 1 (унарный оператор НЕ)

p	q	p & q	p | q	p ^ q	~p
0	0	0	0	0	1
1	0	0	1	1	0
0	1	0	1	1	1
1	1	1	1	0	0

Приоритет операций

Первичные	x++	Постфиксный инкремент
	x--	Постфиксный декремент
	new	Выделение памяти
Унарные	+	
	-	
	!	Логическое отрицание
	++x	Префиксный инкремент
	--x	Префиксный декремент
	(тип)x	Преобразование типа
Мультипликативные	*	Умножение
	/	Деление
	%	Остаток от деления

Преобразование типов в выражениях

Преобразования типов выполняются по принятым в C# *правилам продвижения типов*. Ниже приведен алгоритм, определяемый этими правилами для операций с двумя операндами.

ЕСЛИ один операнд имеет тип `decimal`, ТО и второй операнд продвигается к типу `decimal` (но если второй операнд имеет тип `float` или `double`, результат будет ошибочным).

ЕСЛИ один операнд имеет тип `double`, ТО и второй операнд продвигается к типу `double`.

ЕСЛИ один операнд имеет тип `float`, ТО и второй операнд продвигается к типу `float`.

ЕСЛИ один операнд имеет тип `ulong`, ТО и второй операнд продвигается к типу `ulong` (но если второй операнд имеет тип `sbyte`, `short`, `int` или `long`, результат будет ошибочным).

ЕСЛИ один операнд имеет тип `long`, ТО и второй операнд продвигается к типу `long`.

ЕСЛИ один операнд имеет тип `uint`, а второй — тип `sbyte`, `short` или `int`, ТО оба операнда продвигаются к типу `long`.

ЕСЛИ один операнд имеет тип `uint`, ТО и второй операнд продвигается к типу `uint`.

ИНАЧЕ оба операнда продвигаются к типу `int`.

Класс Math из пространства имен System

$\pi \cdot r^2$

$\max\{5,17\}$

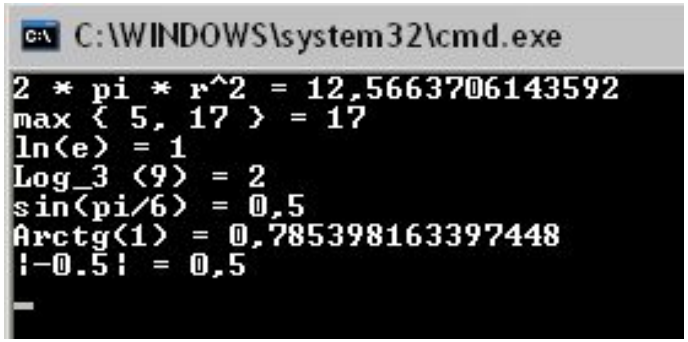
$\ln(e) = \log_e e$

$\log_3 9$

$\sin(\frac{\pi}{6})$

$|-0.5|$

```
static void Main(string[] args)
{
    double radius = 2.0;
    double area = Math.PI * Math.Pow( radius, 2 );
    Console.WriteLine( "2 * pi * r^2 = " + area );

    int maxValue = Math.Max( 5, 17 );
    Console.WriteLine( "max { 5, 17 } = " + maxValue );

    double res1 = Math.Log( Math.E );
    Console.WriteLine( "ln(e) = " + res1 );

    double res2 = Math.Log( 9, 3 );
    Console.WriteLine( "Log_3 (9) = " + res2 );

    double res3 = Math.Sin(Math.PI / 6);
    Console.WriteLine( "sin(pi/6) = " + res3 );

    double res4 = Math.Atan( 1.0 );
    Console.WriteLine( "Arctg(1) = " + res4 );

    double res5 = Math.Abs( -0.5 );
    Console.WriteLine( "|-0.5| = " + res5 );

    Console.ReadKey();
}
```



Г.Шилдт «C# 4.0. Полное руководство» [Стр. 721-726]