Справочная информация по основам языка C#

В этом документе приведена минимально необходимая для выполнения лабораторных работ информация о языке C# и написании на нём программ в среде Visual Studio.

Более подробная информация о разработки программный приложений на языке C# содержится в курсах Программирование и обработка графического интерфейса 2 и 3 семестрах обучения.

Структура минимального консольного приложения C#:

При создании консольного приложения C#, Visual Studio автоматически генерирует минимально необходимый исходный код для работы в консольном режиме, который состоит из следующих блоков:

Подключение пространств имён:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

Для упрощения понимания, пространства имён, подключаемые в C#, можно считать аналогом библиотек в языке C. Каждое из перечисленных пространств имён, содержит функции для работы с соответствующим набором функций. Например, `using System`, подключает пространство имён, содержащее системные функции.

Объявления пространства имён проекта:

```
namespace test
{
    …
}
```

В процессе работы над вашим приложением, вы можете создавать переменные и функции с именами уже существующих переменных и функций. Для того, чтобы компилятор мог определить, с какими именно переменными или функциями вы работаете в данный момент, каждая из них приписывается к определённому пространству имён.

Объявление класса:

```
class Program
{
    …
}
```

Язык C# является объектно-ориентированным. Это означает, что весь программный код приложений должен быть частью классов. При создании приложения средствами Visual Studio, автоматически генерируется класс, внутри которого предлагается описать приложение.

Функция **Main**:

```
static void Main(string[] args)
{
   //исходный код программы
}
```

Для упрощения понимания, можно представить, что это аналог функции Main в языке C. Выполнение приложения начинается с этой функции. Все функции консольного приложения, принадлежащие классу Programm, должны быть объявлены как **static**.

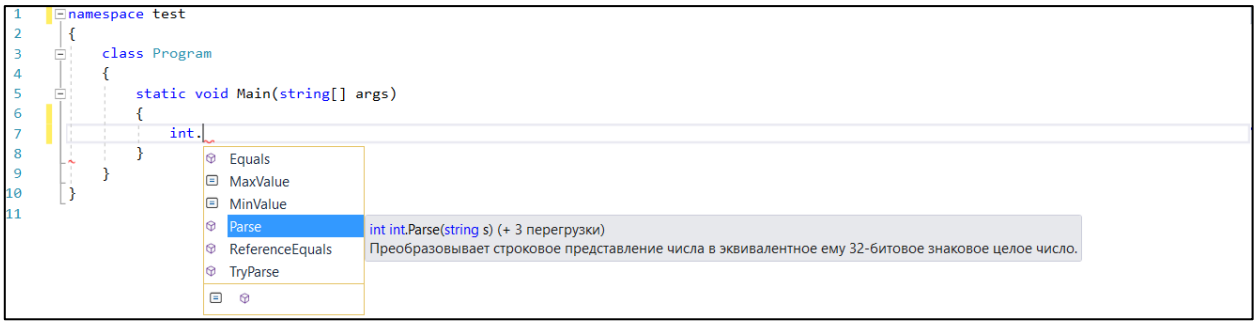
Основные типы данных:

C# тип	.NET Framework тип	Количество бит	Описание
bool	System.Boolean	8	Логический тип, может принимать только два значения: true и false
byte	System.Byte	8	Без знаковый байт
sbyte	System.SByte	8	Знаковый байт
char	System.Char	16	Символ Unicode
decimal	System.Decimal	128	Десятичное число с фиксированной точностью
double	System.Double	64	Число с плавающей запятой
float	System.Single	32	Число с плавающей запятой
int	System.Int32	32	Целое знаковое число
uint	System.UInt32	32	Целое без знаковое число
long	System.Int64	64	Целое знаковое число
ulong	System.UInt64	64	Целое без знаковое число
object	System.Object	-	Базовый тип данных, все остальные типы являются производными от него
short	System.Int16	16	Целое знаковое число
ushort	System.UInt16	16	Целое без знаковое число
string	System.String	-	Строка символов Unicode

Для преобразования типов данных может быть использован класс **Convert**.

В языке C#, все встроенные типы данных являются классами и содержат в себе ряд полезных функций.

Узнать их можно, например, используя средства среды разработки Visual Studio:



Объявление массива в C# реализовано следующим образом:

```
тип_данных[] имя_массива = new тип__данных[размер];
```

Где:

`тип_данных` – название типа данных. (любой стандартный или пользовательский тип)

`[]` – спец. символ для определения массива.

`new` – оператор автоматического выделение памяти под структуру данных.

`размер` – число элементов массива.

Пример:

```
//объявление массива целых чисел с последующей его инициализацией
int[] a;
a = new int[5];

//объявление массива вещественных чисел с его константной инициализацией
float[] b = { 1.1f, 2.7f, 3.5f };

//объявление двумерного массива размерностью 5x5 элементов
int[,] d = new int[5, 5];
```

Обращение к элементам массивов осуществляется следующим образом:

```
//создание массива целых чисел состоящего из 5 элементов
int[] a = new int[5];
//запись числа в последнюю ячейку массива
a[4] = 777;

//создание двумерного массива строк, состоящего из 9 элементов
string[,] m = new string[3,3];
//запись строки в первую ячейку массива
m[0,0] = "Hello World!";
```

Операции ввода-вывода:

В языке C#, работа с консолью осуществляется при помощи встроенного класса **Console**. Ввод и вывод осуществляются при помощи двух групп методов: **Read** и **Write**, которые, в целом, являются аналогами функций **scanf** и **printf** языка C.

Для вывода на экран строки, с последующим переводом курсора на новую строку, используется метод **WriteLine**:

```
Console.WriteLine("Hello World!");
```

Метод **WriteLine**, так же может быть использован для вывода значений переменных:

```
int a = 69;
float b = 4.20f;
string st = "test";

Console.WriteLine("st = {2}, a = {0}, b = {1}", a, b, st);
```

Цифра в фигурных скобках {n} определяет номер параметра, значение которого будет выведено вместо фигурных скобок.

Для того, чтобы получить данные из консоли, можно использовать метод **ReadLine**:

```
//вывод строки без перехода на новую строку
Console.Write("Введите слово: ");
```

```
//получение строки до символа перехода на новую строку
//запись полученной строки в строковую переменную
string word = Console.ReadLine();

Console.Write("Введите число: ");
//получение строки до символа перехода на новую строку
//преобразование полученной строки в число и запись в целочисленную переменную
int digit = int.Parse(Console.ReadLine());
```

Для того, что бы считать из консоли отдельный символ, можно использовать метод **ReadKey**:

```
//считывание данных о нажатой клавиши
ConsoleKeyInfo key = Console.ReadKey();

//получение символа нажатой клавиши (если возможно)
char ch = key.KeyChar;
```

В целом, метод **ReadKey** можно назвать аналогом функции **getch** языка C.

Справка:

Поскольку запуск приложения осуществляется в режиме “окон”, сразу после завершения работы, консольное приложение будет закрыто. Что бы этого не случилось, в конце функции **Main**, можно вставить вызов метода **ReadKey**.

Арифметические и логические операторы:

Основные арифметические и логические операторы в C# :

Операция	Запись
Сложение	a + b
Вычитание	a - b
Деление	a / b
Умножение	a * b
Нахождение остатка от деления	a % b
Инкремент	a++
Декремент	a--
Сравнение, неравенство	== , !=
Больше, меньше	> , <
Больше либо равно, меньше либо равно	>= , <=
И, или	&& , ||

Пример:

```
int a = 1;
int b = 2;
int c = a + b;
int d = a * b;
double e = Convert.ToDouble(a)/b;
```

Условия и циклы:

Оператор условия и оператор выбора, в целом, аналогичны таковым в языке C.

В общем виде, оператор **условия** записывается как:

```
if (условие)
{
    если_условие_истино;
}
else
{
    иначе;
}
```

Пример:

```
int a = 1;

if (a == 1)
    Console.WriteLine("a == 1");
else
    Console.WriteLine("a != 1");
```

В общем виде, оператор **выбора** записывается как:

```
switch (переменная)
{
    case значение_1:
        действие_1;
        break;
    case значение_2:
        действие_2;
        break;
    default:
        действие_по_умолчанию;
        break;
}
```

Пример:

```
int a = 1;

switch (a)
{
    case 1:
        Console.WriteLine("a == 1");
        break;
    case 2:
        Console.WriteLine("a == 2");
        break;
    default:
        Console.WriteLine("a == idk wtf lol");
        break;
}
```

В языке C# реализованы циклы **for** и **do**/**while** аналогичные таковым в языке C.

Цикл **for**, в обобщённом виде, выглядит следующим образом:

```
for(блок_инициализации; блок_условий; блок_изменения)
{
    тело_цикла
}
```

`блок_инициализации` - задает начальное условие. Операторы в этом разделе выполняются только один раз, перед входом в цикл.

`блок_условий` - содержит логическое выражение, которое вычисляется для определения необходимости выхода цикла или его повторного выполнения.

`блок_изменения` - определяет, что происходит после каждой итерации тела цикла.

Пример цикла **for**, печатающего в консоль цифры от 1 до 10:

```
for(int i = 0; i < 10; i++)
{
    Console.Write("{0} ", i+1);
}
```

Цикл **while**, в обобщённом виде, выглядит следующим образом:

```
while (пока_истина)
{
    тело_цикла
}
```

Пример цикла **while**, печатающего в консоль цифры от 1 до 10:

```
int i = 1;

while (i <= 10)
{
    Console.Write("{0} ", i);
    i++;
}
```

Помимо циклических операторов **for** и **while**, в языке C#, существует итеративный оператор **foreach**, который может выполнять функцию цикла.

Пример оператора **foreach**, печатающего в консоль цифры от 1 до 10:

```
//массив содержачий цифры от 0 до 9
int[] numbers = new int[] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

//на каждой итерации создаётся переменная element
//которой присваивается соответствующее значение из массива numbers
foreach (int element in numbers)
{
    System.Console.WriteLine("{0}", element);
}
```

Процедуры и функции.

Поскольку язык C# является ООП языком, любая процедура или функция должна являться методом того или иного класса. В рамках данной лабораторной работы, этим классом является класс **Programm**. Помимо этого, в рамках выполнения лабораторной работы, функции должны быть объявлены с модификатором **static**.

Пример функции сложения двух целых чисел, передаваемых в качестве параметров:

```
static int add(int a, int b)
{
    return a + b;
}
```

Где:
`static` – ключевое слово, означающее что метод можно вызывать без создания класса в котором он описан

`int` – тип возвращаемого значения

`add` – имя метода

`(int a, int b)` – параметры метода

Вызов функции осуществляется следующим образом:

```
static void Main(string[] args)
{
    int a = add(5, 7);

    Console.ReadKey();
}
```

Если возникает необходимость изменения переменных передаваемых в функцию, можно использовать следующую конструкцию:

```
static void init(ref int a)
{
    //создание генератора случайных чисел
    Random rnd = new Random();

    //запись в переменную случайного числа в диапазоне от 5 до 10
    a = rnd.Next(5, 10);
}

static void Main(string[] args)
{
    int a;

    init(ref v1);
}
```

Где ключевое слово `ref` – означает что в качестве параметра передаётся переменная, а не её копия.

Список литературы:

Более подробную информацию об основах языка C# можно получить в следующих источниках:

1. Шилдт Г. - C# 4.0: полное руководство. Издательство: Вильямс, 2011 г. (страницы с 67 по 143)
2. Основы программирования на C#: http://www.intuit.ru/studies/courses/2247/18/info (лекции с 1 по 16)