Т.2 Введение в C#

1.1 Язык C# и платформа .NET

На сегодняшний момент язык программирования C# один из самых мощных, быстро развивающихся и востребованных языков в ИТ-отрасли. В настоящий момент на нем пишутся самые различные приложения: от небольших десктопных программ до крупных веб-порталов и веб-сервисов, обслуживающих ежедневно миллионы пользователей.

C# уже не молодой язык и как и вся платформа .NET уже прошел большой путь. Первая версия языка вышла вместе с релизом Microsoft Visual Studio .NET в феврале 2002 года. Текущей версией языка является версия C# 11, которая вышла 8 ноября 2022 года вместе с релизом .NET 7.

C# является языком с Си-подобным синтаксисом и близок в этом отношении к C++ и Java. Поэтому, если вы знакомы с одним из этих языков, то овладеть C# будет легче.

C# является объектно-ориентированным и в этом плане много перенял у Java и C++. Например, C# поддерживает полиморфизм, наследование, перегрузку операторов, статическую типизацию. Объектно-ориентированный подход позволяет решить задачи по построению крупных, но в тоже время гибких, масштабируемых и расширяемых приложений. И C# продолжает активно развиваться, и с каждой новой версией появляется все больше интересных функциональностей.

Роль платформы .NET

Когда говорят C#, нередко имеют в виду технологии платформы .NET (Windows Forms, WPF, ASP.NET, Xamarin). И, наоборот, когда говорят .NET, нередко имеют в виду C#. Однако, хотя эти понятия связаны, отождествлять их неверно. Язык C# был создан специально для работы с фреймворком .NET, однако само понятие .NET несколько шире.

Основные черты:

Кроссплатформенность. .NET является переносимой платформой (с некоторыми ограничениями). Например, последняя версия платформы на данный момент - .NET 7 поддерживается на большинстве современных ОС Windows, MacOS, Linux. Используя различные технологии на платформе .NET, можно разрабатывать приложения на языке C# для самых разных платформ - Windows, MacOS, Linux, Android, iOS, Tizen.

Мощная библиотека классов. .NET представляет единую для всех поддерживаемых языков библиотеку классов. И какое бы приложение мы не собирались писать на C# -

текстовый редактор, чат или сложный веб-сайт - так или иначе мы задействуем библиотеку классов .NET.

Разнообразие технологий. Общеязыковая среда исполнения CLR и базовая библиотека классов являются основой для целого стека технологий, которые разработчики могут задействовать при построении тех или иных приложений. Например, для работы с базами данных в этом стеке технологий предназначена технология ADO.NET и Entity Framework Core. Для построения графических приложений с богатым насыщенным интерфейсом - технология WPF и WinUI, для создания более простых графических приложений - Windows Forms. Для разработки кроссплатформенных мобильных и десктопных приложений - Xamarin/MAUI. Для создания веб-сайтов и веб-приложений - ASP.NET и т.д.

Производительность. Согласно ряду тестов веб-приложения на .NET 7 в ряде категорий сильно опережают веб-приложения, построенные с помощью других технологий. Приложения на .NET 7 в принципе отличаются высокой производительностью.

Также еще следует отметить такую особенность языка C# и фреймворка .NET, как автоматическая сборка мусора. А это значит, что нам в большинстве случаев не придется, в отличие от C++, заботиться об освобождении памяти. Вышеупомянутая общеязыковая среда CLR сама вызовет сборщик мусора и очистит память.

.NET Framework и .NET 7
Стоит отметить, что .NET долгое время развивался преимущественно как платформа для Windows под названием .NET Framework. В 2019 вышла последняя версия этой платформы - .NET Framework 4.8. Она больше не развивается

1.2 Начало работы. Visual Studio

Чтобы облегчить написание, а также тестирование и отладку программного кода нередко используют специальные среды разработки, в частности, Visual Studio.

Вначале откроем Visual Studio. На стартовом экране выберем Create a new project (Создать новый проект)

На следующем окне в качестве типа проекта выберем Console App, то есть мы будем создавать консольное приложение на языке C#

Далее на следующем этапе нам будет предложено указать имя проекта и каталог, где будет располагаться проект.

В поле Project Name дадим проекту какое-либо название. В моем случае это HelloApp.

На следующем окне Visual Studio предложит нам выбрать версию .NET, которая будет использоваться для проекта. Выберем последнюю на данный момент верси. - .NET 7.0:

Нажмем на кнопку Create (Создать) для создания проекта, и после этого Visual Studio создаст и откроет нам проект:

В большом поле в центре, которое по сути представляет текстовый редактор, находится сгенерированный по умолчанию код C#. Впоследствии мы изменим его на свой.

Справа находится окно Solution Explorer, в котором можно увидеть структуру нашего проекта. В данном случае у нас сгенерированная по умолчанию структура: узел Dependencies - это узел содержит сборки dll, которые добавлены в проект по умолчанию. Эти сборки как раз содержат классы библиотеки .NET, которые будет использовать C#. Однако не всегда все сборки нужны. Ненужные потом можно удалить, в то же время если понадобится добавить какую-нибудь нужную библиотеку, то именно в этот узел она будет добавляться.

Далее идет непосредственно сам файл кода программы Program.cs, который по умолчанию открыт в центральном окне и который имеет всего две строки:

Вторая строка собственно представляет собой код программы: Console.WriteLine("Hello World!");. Эта строка выводит на консоль строку "Hello World!".

Несмотря на то, что программа содержит только одну строку кода, это уже некоторая программа, которую мы можем запустить. Запустить проект мы можем с помощью клавиши F5 или с панели инструментов, нажав на зеленую стрелку. И если вы все сделали правильно, то при запуске приложения на консоль будет выведена строка "Hello World!".

Выполнение программы

Весь код программы на языке C# помещается в файлы с расширением .cs. По умолчанию в проекте, который создается в Visual Studio (а также при использовании .NET CLI) уже есть один файл с кодом C# - файл Program.cs.

Именно код файла Program.cs выполняется по умолчанию, если мы запустим проект на выполнение. Но при необходимости мы также можем добавлять другие файлы с кодом C#.

Инструкции

Базовым строительным блоком программы являются инструкции (statement). Инструкция представляет некоторое действие, например, арифметическую операцию, вызов метода, объявление переменной и присвоение ей значения. В конце каждой инструкции в C# ставится точка с запятой (;). Данный знак указывает компилятору на

конец инструкции. Например, в проекте консольного приложения, который создается по умолчанию, есть такая строка:

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

Данная строка представляет вызов метода Console.WriteLine, который выводит на консоль строку. В данном случае вызов метода является инструкцией и поэтому завершается точкой с запятой.

Набор инструкций может объединяться в блок кода. Блок кода заключается в фигурные скобки, а инструкции помещаются между открывающей и закрывающей фигурными скобками. Например, изменим код файла Program.cs на следующий:

```
{
    Console.WriteLine("Привет");
    Console.WriteLine("Добро пожаловать в C#");
}
```

Регистрозависимость

C# является регистрозависимым языком. Это значит, что в зависимости от регистра символов какие-то определенные названия могут представлять разные классы, методы, переменные и т.д. Например, для вывода на консоль используется метод WriteLine - его имя начинается именно с большой буквы: "WriteLine". Если мы вместо "Console.WriteLine" напишем "Console.writeline", то программа не скомпилируется, так как данный метод обязательно должен называться "WriteLine", а не "writeline" или "WRITELINE" или как-то иначе.

Комментарии

Важной частью программного кода являются комментарии. Они не являются собственно частью программы, при компиляции они игнорируются. Тем не менее комментарии делают код программы более понятным, помогая понять те или иные его части.

есть два типа комментариев: однострочный и многострочный. Однострочный комментарий размещается на одной строке после двойного слеша //. А многострочный комментарий заключается между символами /* текст комментария */. Он может размещаться на нескольких строках. Например:

1.3 Переменные и константы

Для хранения данных в программе применяются переменные. Переменная представляет именованную область памяти, в которой хранится значение определенного типа. Переменная имеет тип, имя и значение. Тип определяет, какого рода информацию может хранить переменная.

Перед использованием любую переменную надо определить. Синтаксис определения переменной выглядит следующим образом:

тип имя_переменной;
Вначале идет тип переменной, потом ее имя. В качестве имени переменной может выступать любое произвольное название, которое удовлетворяет следующим требованиям:

имя может содержать любые цифры, буквы и символ подчеркивания, при этом первый символ в имени должен быть буквой или символом подчеркивания

в имени не должно быть знаков пунктуации и пробелов

имя не может быть ключевым словом языка C#. Таких слов не так много, и при работе в Visual Studio среда разработки подсвечивает ключевые слова синим цветом.

Хотя имя переменной может быть любым, но следует давать переменным описательные имена, которые будут говорить об их предназначении.

Например, определим простейшую переменную:

string name;
В данном случае определена переменная name, которая имеет тип string. то есть переменная представляет строку. Поскольку определение переменной представляет собой инструкцию, то после него ставится точка с запятой.

При этом следует учитывать, что C# является регистрозависимым языком, поэтому следующие два определения переменных будут представлять две разные переменные:

string name;
string Name;
После определения переменной можно присвоить некоторое значение:

```
string name;
name = "Tom";
```

Так как переменная name представляет тип string, то есть строку, то мы можем присвоить ей строку в двойных кавычках. Причем переменной можно присвоить только то значение, которое соответствует ее типу.

В дальнейшем с помощью имени переменной мы сможем обращаться к той области памяти, в которой хранится ее значение.

Также мы можем сразу при определении присвоить переменной значение. Данный прием называется инициализацией:

```
string name = "Tom";
```

Отличительной чертой переменных является то, что в программе можно многократно менять их значение. Например, создадим небольшую программу, в которой определим переменную, поменяем ее значение и выведем его на консоль:

```
string name = "Tom";  // определяем переменную и инициализируем ее

Console.WriteLine(name);    // Tom

name = "Bob";       // меняем значение переменной
Console.WriteLine(name);    // Bob
```

Консольный вывод программы:

```
Tom
Bob
```

Константы

Отличительной особенностью переменных является то, что мы можем изменить их значение в процессе работы программы. Но, кроме того, в C# есть константы. Константа должна быть обязательно инициализирована при определении, и после определения значение константы не может быть изменено

Константы предназначены для описания таких значений, которые не должны изменяться в программе. Для определения констант используется ключевое слово const, которое указывается перед типом константы:

```
const string NAME = "Tom";  // определяем константу
```

Так, в данном случае определена константа NAME, которая хранит строку "Tom". Нередко для название констант используется верхний регистр, но это не более чем условность.

При использовании констант надо помнить, что объявить мы их можем только один раз и что к моменту компиляции они должны быть определены. Так, в следующем случае мы получим ошибку, так как константе не присвоено начальное значение:

```
const string NAME;  // ! Ошибка - константа NAME не инициализирована
```

Кроме того, мы ее не сможем изменим в процессе работы программы:

```
const string NAME = "Tom";  // определяем константу
NAME = "Bob";   // !Ошибка - у константы нельзя изменить значение
```

Таким образом, если нам надо хранить в программе некоторые данные, но их не следует изменить, они определяются в виде констант. Если же их можно изменять, то они определяются в виде переменных.

1.4 Литералы

Литералы представляют неизменяемые значения (иногда их еще называют константами). Литералы можно передавать переменным в качестве значения. Литералы бывают логическими, целочисленными, вещественными, символьными и строчными. И отдельный литерал представляет ключевое слово null.

Логические литералы

Есть две логических константы - true (истина) и false (ложь):

```
Console.WriteLine(true);
Console.WriteLine(false);
```

Целочисленные литералы

Целочисленные литералы представляют положительные и отрицательные целые числа, например, 1, 2, 3, 4, -7, -109. Целочисленные литералы могут быть выражены в десятичной, шестнадцатеричной и двоичной форме.

С целыми числами в десятичной форме все должно быть понятно, так как они используются в повседневной жизни:

```
Console.WriteLine(-11);
Console.WriteLine(5);
Console.WriteLine(505);
```

Числа в двоичной форме предваряются символами 0b, после которых идет набор из нулей и единиц:

```
Console.WriteLine(0b11);        // 3
Console.WriteLine(0b1011);      // 11
Console.WriteLine(0b100001);    // 33
```

Для записи числа в шестнадцатеричной форме применяются символы 0x, после которых идет набор символов от 0 до 9 и от A до F, которые собственно представляют число:

```
Console.WriteLine(0x0A);    // 10
Console.WriteLine(0xFF);    // 255
Console.WriteLine(0xA1);    // 161
```

Вещественные литералы

Вещественные литералы представляют дробные числа. Этот тип литералов имеет две формы. Первая форма - вещественные числа с фиксированной запятой, при которой дробную часть отделяется от целой части точкой. Например:

```
Console.WriteLine(3.2e3);   // по сути равно 3.2 * 10<sup>3</sup> = 3200
Console.WriteLine(1.2E-1);  // равно 1.2 * 10<sup>-1</sup> = 0.12
```

Символьные литералы

Символьные литералы представляют одиночные символы. Символы заключаются в одинарные кавычки.

Символьные литералы бывают нескольких видов. Прежде всего это обычные символы:

```
'2'
'A'
'T'
```

Также мы можем передать их вывести на консоль с помощью Console.WriteLine:

```
Console.WriteLine('2');
Console.WriteLine('A');
Console.WriteLine('T');
```

Специальную группу представляют управляющие последовательности Управляющая последовательность представляет символ, перед которым ставится слеш. И данная последовательность интерпретируется определенным образом. Наиболее часто используемые последовательности:

'\n' - перевод строки

'\t' - табуляция

'\\' - слеш

И если компилятор встретит в тексте последовательность \t, то он будет воспринимать эту последовательность не как слеш и букву t, а как табуляцию - то есть длинный отступ.

Строковые литералы

Строковые литералы представляют строки. Строки заключаются в двойные кавычки:

```
Console.WriteLine("hello");
Console.WriteLine("фыва");
Console.WriteLine("hello word");
```

Если внутри строки необходимо вывести двойную кавычку, то такая внутренняя кавычка предваряется обратным слешем:

```
Console.WriteLine("Компания \"Рога и копыта\"");
```

Также в строках можно использовать управляющие последовательности. Например, последовательность '\n' осуществляет перевод на новую строку:

```
Console.WriteLine("Привет \nмир");
```

При выводе на консоль слово "мир" будет перенесено на новую строку:

```
Привет
мир
```

null

null представляет ссылку, которая не указывает ни на какой объект. То есть по сути отсутствие значения.

1.5 Типы данных

Последнее обновление: 10.11.2021

Как и во многих языках программирования, в C# есть своя система типов данных, которая используется для создания переменных. Тип данных определяет внутреннее представление данных, множество значений, которые может принимать объект, а также допустимые действия, которые можно применять над объектом.

В языке C# есть следующие базовые типы данных:

ТИП	ОБЛАСТЬ ЗНАЧЕНИЙ	РАЗМЕР
sbyte	-128 до 127	Знаковое 8-бит целое
byte	0 до 255	Беззнаковое 8-бит целое

char	U+0000 до U+ffff	16-битовый символ Unicode
bool	true или false	1 байт*
short	-32768 до 32767	Знаковое 16-бит целое
ushort	0 до 65535	Беззнаковое 16-бит целое
int	-2147483648 до 2147483647	Знаковое 32-бит целое
uint	0 до 4294967295	Беззнаковое 32-бит целое
long	-9223372036854775808 до 9223372036854775807	Знаковое 64-бит целое
ulong	0 до 18446744073709551615	Беззнаковое 64-бит целое
float	±1,5*10-45 до ±3,4*1033	4 байта, точность — 7 разрядов
double	±5*10-324 до ±1,7*10306	8 байтов, точность — 16 разрядов
decimal	(-7,9 * 1028 до 7,9 * 1028) / (100–28)	16 байт, точность — 28

bool: хранит значение true или false (логические литералы). Представлен системным типом System.Boolean

bool alive = true;
bool isDead = false;
byte: хранит целое число от 0 до 255 и занимает 1 байт. Представлен системным типом System.Byte

byte bit1 = 1;
byte bit2 = 102;
sbyte: хранит целое число от -128 до 127 и занимает 1 байт. Представлен системным типом System.SByte

sbyte bit1 = -101;
sbyte bit2 = 102;
short: хранит целое число от -32768 до 32767 и занимает 2 байта. Представлен системным типом System.Int16

short n1 = 1;
short n2 = 102;
ushort: хранит целое число от 0 до 65535 и занимает 2 байта. Представлен системным типом System.UInt16

1
2
ushort n1 = 1;
ushort n2 = 102;
int: хранит целое число от -2147483648 до 2147483647 и занимает 4 байта. Представлен системным типом System.Int32. Все целочисленные литералы по умолчанию представляют значения типа int:

1
2
3
int a = 10;
int b = 0b101; // бинарная форма b =5
int c = 0xFF;   // шестнадцатеричная форма c = 255
uint: хранит целое число от 0 до 4294967295 и занимает 4 байта. Представлен системным типом System.UInt32

1
2
3
uint a = 10;
uint b = 0b101;
uint c = 0xFF;
long: хранит целое число от –9 223 372 036 854 775 808 до 9 223 372 036 854 775 807 и занимает 8 байт. Представлен системным типом System.Int64

1

```
2
3
long a = -10;
long b = 0b101;
long c = 0xFF;
```
ulong: хранит целое число от 0 до 18 446 744 073 709 551 615 и занимает 8 байт. Представлен системным типом System.UInt64

```
1
2
3
ulong a = 10;
ulong b = 0b101;
ulong c = 0xFF;
```
float: хранит число с плавающей точкой от -3.4*1038 до 3.4*1038 и занимает 4 байта. Представлен системным типом System.Single

double: хранит число с плавающей точкой от ±5.0*10-324 до ±1.7*10308 и занимает 8 байта. Представлен системным типом System.Double

decimal: хранит десятичное дробное число. Если употребляется без десятичной запятой, имеет значение от ±1.0*10-28 до ±7.9228*1028, может хранить 28 знаков после запятой и занимает 16 байт. Представлен системным типом System.Decimal

char: хранит одиночный символ в кодировке Unicode и занимает 2 байта. Представлен системным типом System.Char. Этому типу соответствуют символьные литералы:

```
1
2
3
char a = 'A';
char b = '\x5A';
char c = '\u0420';
```
string: хранит набор символов Unicode. Представлен системным типом System.String. Этому типу соответствуют строковые литералы.

```
1
2
string hello = "Hello";
string word = "world";
```
object: может хранить значение любого типа данных и занимает 4 байта на 32-разрядной платформе и 8 байт на 64-разрядной платформе. Представлен системным типом System.Object, который является базовым для всех других типов и классов .NET.

```
1
2
3
object a = 22;
object b = 3.14;
object c = "hello code";
```

Например, определим несколько переменных разных типов и выведем их значения на консоль:

```
1
2
3
4
5
6
7
8
9
string name = "Tom";
int age = 33;
bool isEmployed = false;
double weight = 78.65;

Console.WriteLine($"Имя: {name}");
Console.WriteLine($"Возраст: {age}");
Console.WriteLine($"Вес: {weight}");
Console.WriteLine($"Работает: {isEmployed}");
```