Основы программирования

Синтаксис языка C#

Базовые синтаксические конструкции

История языка C#

C# является

- прямым наследником C и C++
- тесно связан Java

Язык C

Появился в результате революции в *структурном программировании* в 1960-е годы.

И стал самым распространенным языком структурного программирования в 1980-е годы.

Язык C был разработан Деннисом Ритчи (Dennis Ritchie)

Язык C++

К концу 1970-х годов методики структурного программирования не могли справится с масштабными проектами.

Для решения этой проблемы было открыто новое направление в программировании — так называемое *объектно-ориентированное программирование* (ООП).

Язык C++ был разработан в 1979 году Бьярне Страуструпом (Bjarne Stroustrup).

Язык C++

Первоначально новый язык назывался "C с классами", но в 1983 году он был переименован в C++.

Язык C полностью входит в состав C++ .

Большая часть дополнений обеспечивает плавный переход к ООП.

К концу 1990-х годов он стал наиболее широко распространенным языком программирования.

Язык Java

Java это структурированный, объектно-ориентированный язык. В основу этого языка положен синтаксис C, а его объектная модель получила свое развитие из C++.

Толчком для разработки Java послужила потребность в независящем от платформы языке (не Интернет).

Переносимость программ достигалась благодаря преобразованию исходного кода в промежуточный, так называемый *байт-кодом.*

Этот байт-код затем выполнялся виртуальной машиной Java (JVM) — основной частью исполняющей системы Java.

Ограничения языка Java

- недостает *межъязыковой возможности взаимодействия,* называемой также *многоязыковым программированием (*возможность кода, написанного на одном языке, без труда взаимодействовать с кодом, написанным на другом языке).
  Требуется для построения крупных, распределенных программных систем
- отсутствует полная интеграция с платформой `Windows`. Хотя программы на `Java` могут выполняться в среде `Windows`, при условии, что установлена виртуальная машина `Java`, среды `Java` и `Windows` не являются сильно связанными

Язык C#

Разработан в конце 1990-х годов, как часть общей стратегии .NET.

Главный разработчик -- Андерс Хейльсберг, годы он был автором языка Turbo Pascal (1980-е).

Впервые C# был выпущен в виде альфа-версии в середине 2000 года компанией Microsoft и был использован для разработки большей части базовых классов и утилит платформы .NET.

Платформа Microsoft.NET

Microsoft.NET -- это интегрированная система средств разработки, развертывания и выполнения сложных распределенных программных систем.

CLR (Common Language Runtime) – основой платформы .NET. Активизирует исполняемый код, выполняет проверку безопасности, располагает этот код в памяти и исполняет его, управляет свободной памятью, обеспечивая автоматическое освобождения памяти (сборку мусора).

Программный код, получаемый после компиляции, представляется на общем промежуточном языке (Common Intermediate Language или CIL).

Платформа Microsoft.NET

Программа на языке CIL платформенно-независимая, но требует некоторой дополнительной настройки (компиляции) перед началом своего выполнения.

Программные файлы на языке CIL называются сборками (assembly). Сборки – это файлы с расширениями exe или dll, состоящие из программного кода на языке CIL и из дополнительных служебных данных (сведения о типах, данные о версии, ссылки на внешние сборки и т.п.)

Сборки перед исполнением проходят настройку с помощью JIT-компиляторов (Just-In-Time compilers), переводящих CIL-код в машинный (native) код платформы исполнения.

Исполнение машинного кода происходит в среде CLR.

Пример программы

```
using System;
namespace Hello
{
    /* программа, которая спрашивает у пользователя имя
       и выводит его на консоль   */
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Введите свое имя: ");
            string name = Console.ReadLine();
            Console.WriteLine("Привет" + name);
            Console.ReadLine();
        }
    }
}
```

Алфавит языка

Группы символов :

- буквы (латинские и национальных алфавитов) и символ подчеркивания «_», который используется в качестве буквы;
- цифры;
- специальные символы, например «-», «+», «’» и т.д.;
- пробельные символы (пробел и табуляция);
- символы перевода строки.

Регистрозависимость

C# является регистрозависимым языком.

Например, название обязательного метода Main начинается именно с большой буквы: "Main".

"main" ≠ "Main" ≠ "MAIN".

Конструкции языка

- директивы препроцессора
  (не тема настоящего курса)
- комментарии
  (способы пояснения кода программы);
- лексемы (или слово)
  (минимальная единица языка, имеющая собственный смысл)

Комментарии

- однострочный

```
//Это текст однострочного комментария.
```

- многострочный, также может содержать только часть строки)

```
/*А это текст
многострочный комментарий*/
```

- и еще комментарий документирования

```
///Как-то так ...
```

Не могут быть вложенными, могут содержать любой текст

Лексемы

Лексема (или слово) – это минимальная единица языка, имеющая собственный смысл.

- идентификаторы (имена объектов);
- ключевые слова;
- операции (операторы);
- разделители;
- литералы (константы).

Идентификаторы

Идентификатор – это имя присвоенное созданному пользователем объекту программы (методу, переменной или иному элементу).

- Так как прописные и строчные буквы в C# различаются, то `myVar, MyVar, myvar` – это три различных идентификатора.
- Обычно идентификатор отражает назначение или смысловую характеристику элемента.

Правила составления идентификаторов

- идентификатор может состоять только из букв, символа «_» и цифр;
- первым символом в идентификаторе может быть только буква или символ «_»;
- количество символов в идентификаторе не регламентируется;
- пробелы в идентификаторе не допускаются;
- допустимо использование букв национальных алфавитов, хотя это и не приветствуется;
- совпадение идентификатора с ключевыми словами не допускается.

Правила хорошего стиля ...

- имена должны быть понятными и соответствовать контексту использования;
- имена составляются из слов, описывающих контекст использования именуемого объекта;
- имя должно соответствовать одной из принятых нотаций – правилу создания имён.

Нотации часто используемые а C#

- **Паскаля**
  Каждое слово в идентификаторе начинается с большой буквы.
  Например: `CountSqareOfRectangle`.

- **Camel**
  Каждое слово в идентификаторе начинается с большой буквы, кроме первого.
  Например: `countSqureOfRectangle`.

Другие варианты нотаций

- **Венгерская**
  В начале имени используется префикс, представляющий собой первый символ названия типа, которому принадлежит именуемый объект.
  Например: `dCountSqareOfRectangle`.

- и еще …
  `PRINT_ARRAY`
  `print_array`
  `_printArray`

Ключевые слова

Это идентификаторы, которые являются зарезервированными и имеют специальное значение.

Например: `if, do, for, int, goto` и другие.

Ключевые слова 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			

Операции

Операция – это действие над операндами:

- унарные – над одним операндом,
- бинарные – над двумя операндами,
- тернарные – над несколькими операндами.

Обозначаются одним или несколькими символами, могут записываться как с помощью специальных символов, так буквенным обозначением (такие операции являются ключевыми словами).

Операция представляет собой отдельную лексему - внутри операции пробелы не допускаются. Исключения: ( ), [ ] и ? : .

Примеры операций

```
a+=b, c=d
```
(символы в выражении)

```
a>b && c<=d
```
(специальные символы)

```
pow, sqrt , new, as
```
(буквы)

Порядок выполнения операций

При выполнении сразу нескольких арифметических операций следует учитывать порядок их выполнения.

Приоритет операций (от наивысшего к низшему):

- инкремент, декремент
- умножение, деление, получение остатка
- сложение, вычитание

Для изменения порядка следования операций применяются скобки.

Порядок выполнения операций

```
int a = 3;
int b = 5;
int c = 40;
int d = c---b*a;          // a=3  b=5  c=39  d=25
Console.WriteLine("a="+a + "b="+b + "c="+c + "d="+d);
```

Фактически выполняется как

```
int d = (c--)-(b*a);      // a=3  b=5  c=39  d=25
```

Изменяем порядок выполнения операций

```
int d = (c-(--b))*a;      // a=3  b=4  c=40  d=10
```

Ассоциативность операторов

Когда операции имеют один и тот же приоритет, порядок вычисления определяется ассоциативностью операторов.

В зависимости от ассоциативности есть два типа операторов:

- левоассоциативные операторы, которые выполняются слева направо;
- правоассоциативные операторы, которые выполняются справа налево.

Ассоциативность операторов

Как трактовать выражение `int x = 10 / 5 * 2;`

как `(10 / 5) * 2` или как `10 / (5 * 2)`?

Верная трактовка `(10 / 5) * 2`, так как все арифметические операторы (кроме префиксного инкремента и декремента) являются левоассоциативными, то есть выполняются слева направо.

Разделители

Предназначены для отделения элементов друг от друга или группировки элементов в единую конструкцию.

Примеры разделителей в C#: `{ } [ ] () . , : ;`

Литералы

Это неизменяемые величины-значения (константы).

Относятся к какому-либо типу данных в соответствии со своим внешним представлением.

- логические – `true` и `false`;
- целые – в десятичном или шестнадцатеричном коде, например, `10` или `0xA`;
- вещественные – записываются как в десятичной, так и в экспоненциальной форме, например `31.5`, `0.315e2`;
- символьные – символ, заключенный в «'»;
- строковые – строка, заключенная в «"»;
- `null` – константа, означающая отсутствие объекта (пустая ссылка).

Литералы

Литералы вычленяются из текста программы и относятся к какому-либо типу данных в соответствии со своим внешним представлением прямо на этапе компиляции программы.

Целочисленные литералы

- положительные и отрицательные целые числа, Целочисленные литералы могут быть выражены в десятичной, шестнадцатеричной и двоичной форме.

```
1  Console.WriteLine(-11);
2  Console.WriteLine(5);
3  Console.WriteLine(505);
```

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

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

Вещественные литералы

Этот тип литералов имеет две формы:

- с фиксированной запятой, при которой дробную часть отделяется от целой части точкой.

```
3.14
100.001
-0.38
```

- экспоненциальной форме MeP,
  где M — мантисса, e - экспонента, которая фактически означает "*10^" (умножить на десять в степени), а P — порядок. 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

- 'a', 'ю', '*'
- символ, записанный 16-ричным кодом ASCII ('\x')

```
1  Console.WriteLine('\x78');    // x
2  Console.WriteLine('\x5A');    // Z
```

- escape-последовательность:
  '\a' – звуковой сигнал,
  '\b' – возврат на символ,
  '\n' – перевод строки и т.д.
- символ в кодировке Unicode ('\u')

```
1  Console.WriteLine('\u0420');    // P
2  Console.WriteLine('\u0421');    // C
```

Управляющие последовательности

Вид	Наименование
\a	Звуковой сигнал
\b	Возврат на шаг
\f	Перевод страницы (формата)
\n	Перевод строки
\r	Возврат каретки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\\	Обратная косая черта
\'	Апостроф
\"	Кавычка
\0	Нуль-символ

Строковые литералы

```
1  Console.WriteLine("hello");
2  Console.WriteLine("фыва");
3  Console.WriteLine("hello word");
```

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

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

Управляющая последовательность

Интерпретируется как один символ, поэтому она может быть включена в состав строкового литерала.

Например: “`Мама\nмыла\nраму`“ выводится на экран как:

```
Мама
мыла
раму
```

Дословные строковые литералы

Снабжены префиксом «@». Для такого литерала отключена обработка управляющих последовательностей.

Например: `@"Мама\nмыла\nраму"` выводится на экран как

```
Мама\nмыла\nраму¶
```

Дословные литералы. Использование

Удобны при наличии большого количества символов, представимых только с помощью управляющей последовательности – символов «\», «”» или «’».

Например:

```
”C:\\MyDirectory\\MyFile.cs”
@”C:\MyDirectory\MyFile.cs”
```

Пустые литералы

Содержимое строкового литерала может быть пустым (обозначается “”), а вот пустой символьный литерал не допускается.

Рекомендуемая литература

1. Павловская - C#. Программирование на языке высокого уровня.
2. Шилдт Г. - Полный справочник по C#.
3. Троелсен Э. - Язык программирования C# 2010 и платформа .NET 4.