Тема: Лексические единицы языка C#

Мельникова Татьяна Федоровна

Цели занятия


В ходе занятия ***формировать общие компетенции***:

-Понимать сущность и социальную значимость своей будущей профессии, проявлять к ней устойчивый интерес;

-организовывать взаимосвязь своих знаний и умений, решать поставленную проблему;

Цели занятия



Усваивают знания:

- Алфавит языка C#;

-Основные лексические языка C#

-Типы констант

ЭЛЕМЕНТЫ ЯЗЫКА

А Л Ф А В И Т языка C# включает в себя:

1 Заглавные и строчные латинские буквы

A,B …Z, a,b, …z

2 Десятичные цифры:

0 1 2 3 4 5 6 7 8 9

3 Специальные символы

. , ; : ? ' ! | / \ ! ~ _ ( ) { } [ ] < > = + - * # % & ^

4 Символы перевода строки.

Эти символы пишутся по правилам их применения, а для компилятора они имеют специальный смысл.

ЭЛЕМЕНТЫ ЯЗЫКА

Лексема — это минимальная единица языка, имеющая самостоятельный смысл. Существуют следующие виды лексем:

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

Лексемы языка программирования аналогичны словам естественного языка. Например, лексемами являются:

число - 128, имя - Visa,

ключевое слово – while

и знак операции сложения- +

ЭЛЕМЕНТЫ ЯЗЫКА

Комментарии предназначены для записи пояснений к программе и формирования документации.

Однострочный комментарий

```
// задание1- найти….
// дата выполнения 30.10.20
```

многострочный заключается между символами- /* .. */

```
/*  строка комментария 1
    строка комментария 2
   …
    строка комментария n
*/
```

ЭЛЕМЕНТЫ ЯЗЫКА

Из лексем составляются выражения и операторы.

Выражение задает правило вычисления некоторого значения. Например,

-выражение a + b задает правило вычисления суммы двух величин.

-выражение (a > b) задает логическое выражение

Оператор задает законченное описание некоторого действия, данных или элемента программы.

Например:

```
int a= 10;  // оператор описания целочисленной переменной a  и
            // присвоение ей значения.
```

ЭЛЕМЕНТЫ ЯЗЫКА

Идентификаторы (имена) служат для того чтобы ***обращаться*** к программным ***объектам*** и ***различать*** их, то есть идентифицировать.

В идентификаторе могут использоваться буквы, цифры и символ подчеркивания. Прописные и строчные буквы **различаются,** например,

sysop, SySoP и SYSOP — три разных имени.

Первым символом идентификатора может быть буква или знак подчеркивания, но не цифра.

Пробелы внутри имен не допускаются.

идентификатор


Идентификатор создается на этапе объявления переменной (метода, типа и т. п.), после этого его можно использовать в последующих операторах программы.

Стили именования

- **Pascal case** – первая буква каждого слова в имени идентификатора начинается с верхнего регистра.

  *Пример: **T**heCategory;*

- **Camel case** – первая буква первого слова в идентификаторе в нижнем регистре, все первые буквы последующих слов – в верхнем.

  *Пример: **t**heCategory;*

- **UpperCase** – стиль используется только для сокращений, все буквы в имени идентификатора в верхнем регистре.

  *Пример: **ID**;*

- **Hungarian notation** – перед именем идентификатора пишется его тип в сокращенной форме.

  *Пример: **str**FirstName, **i**CurrentYear.*

Общие правила именования идентификаторов



Правила именования идентификаторов

- При именовании идентификаторов не используются аббревиатуры или сокращения, если только они не являются общепринятыми.

Пример: GetWindow(), а не ***GetWin();***

Общие правила именования идентификаторов



- Если имя идентификатора включает в себя сокращение –сокращение пишется в ***upper case***.

Исключение - когда имя идентификатора должно быть указано в ***camel case*** и сокращение стоит в начале имени идентификатора. В этом случае сокращение пишется в нижнем регистре.

Пример:

PPCAccount (PPC – сокращение от pay per click) для *pascal case*,

ppcAccount для *camel case*.

Использование верхнего и нижнего регистра в именах

Запрещается создавать два различных имени, функции, типа или свойства с одинаковыми именами, отличающиеся только регистром.

Запрещается создавать функции с именами параметров, отличающимися только регистром. Ниже приведены примеры **НЕправильных названий**.

Пример: *KeywordManager и Keywordmanager;*

*KeywordManager.**Keyword** иKeywordManager.**KEYWORD**;*

*int **id** {get, set} и int **ID** {get, set};*

findByID(int id) и FindByID(int id);

*void MyFunction(string **s**, string **S**).*

ЭЛЕМЕНТЫ ЯЗЫКА

При выборе идентификатора необходимо следить, чтобы он не совпадал с ключевыми словами.

Ключевые слова — это зарезервированные идентификаторы, которые имеют специальное значение для компилятора.

Их можно использовать только в том смысле, в котором они определены.

КЛЮЧЕВЫЕ СЛОВА

Ключевые - это предопределенные идентификаторы, которые имеют специальное значение для компилятора.

Типы данных:	Операторы:	Специальные слова
• char • float • short • typedef • void • decimal • double • int • class • bool • enum • long • struct • unsigned	• break • default • for • return • while • case • do • goto • sizeof • continue • if .. else • switch	• const • false • finally • catch • try • object • interface • delegate • event • fixed • extern

Знаки операций и разделители

Знак операции — это один или более символов, определяющих действие над операндами.

Внутри знака операции пробелы не допускаются. Например, выражении **c += b**
знак += является знаком операции, переменные **c и b** — операндами.

Символы, составляющие знак операций, могут быть как специальными, например, **&&,** | и <, так и буквенными, такими как **as** или **new**.

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.

КОНСТАНТЫ



Величины, значения которых не может быть изменено в программе называют константами.

Константы могут быть

- числовые: целые и вещественные,
- символьные - один символ и строка символов,
- логические.

Объявление константы:

const тип имя = значение;

Для именования констант используется стиль *pascal case.*

КОНСТАНТЫ

1. Целые константы - это десятичные или шестнадцатеричные данные.

- Десятичные константы записываются последовательностью цифр от 0 до 9;

  8 0 199226

- шестнадцатеричная константа может содержать цифры от 0 до 9 , буквы от A до F, а ведущим будет 0x или 0X, т.е. признак системы счисления.

  0xA 0x1B8 0X00FF

2. Константы вещественные

Константа с плавающей точкой - это действительное число, которое представлено

- **с фиксированной точкой:**

[цифры][.][цифры][суффикс]

-с плавающей точкой (с порядком):

[цифры][.][цифры]{E|e}[+|–][цифры] [суффикс]

Суффикс — один из символов F, f, D, d, M, m

2. Константы вещественные

-с фиксированной точкой:

5.7 .001 35.

5.7F .001d 35.5F .001f 35m

-с порядком:

0.2e1 -6 .11e+3 5e-10

пример :

```
const double x =2.1, y =0.59;
const float  z = -4.0658;
```

3. Символьные константы - это любой символ, заключенный **в апострофы.**

Если в виде символа требуется записать апостроф или обратный слеш, то тогда перед символом ставится обратный слеш.

```
const char chZv='*';
const char chLet='C';
const char chAp='\'';   //апостроф
const char chSl='\\';  //обратный слеш
```

Символьные строки

4. **Строка символов.**

Символьные строки - это последовательность символов , заключенная в двойные кавычки.

Строка рассматривается как массив символов за исключением символов ("), (\) и (\n).

Если их нужно представить как символы, то выполняется вышеописанное требование их представления.

Пример:

"\t Значение r = 0xF5 \n"

"d:\\temp\\file1.txt" – путь к файлу

ESCAPE последовательности

Управляющая escape-последовательность интерпретируется как одиночный символ и используется для представления:

• кодов, не имеющих графического изображения (например, \n);

• символов, имеющих специальное значение в строковых и символьных литералах, например, апострофа (').

Управляющие символы - ESCAPE последовательности

символ	Назначение
\a	Соответствует знаку колокольчика (будильника) \0007.
\b	Соответствует BACKSPACE \0008
\t	Соответствует знаку табуляции \0009.
\n	Соответствует знаку новой строки \000A.
\v	Соответствует знаку вертикальной табуляции \000B.
\"	Двойная кавычка
\'	Апостроф
\\	Обратный слеш

дословные литералы



Дословные литералы предваряются символом **@**, который ***отключает обработку управляющих последовательностей и позволяет получать строки в том виде, в котором они записаны.***

Чаще всего дословные литералы применяются при задании полного пути файла.

Сравните два варианта записи одного и того же пути:

```
"C:\\app\\bin\\debug\\a.exe"

@"C:\app\bin\debug\a.exe"
```

null



Константа **null** представляет собой значение, задаваемое по умолчанию для величин ***ссылочных типов***

Программа

Для того чтобы изучать именно язык программирования будем работать с консольными приложениями.

При запуске консольного приложения операционная система создает так называемое консольное окно, через которое идет весь ввод-вывод программы.

Программа



Любая ***программа на языке C# - это набор классов,*** которые взаимодействуют друг с другом.

В одном из классов программы должна находиться, так называя «точка входа» - статический метод Main.

Наличие или отсутствие этого метода определяет тип получаемого результата компиляции –сборки.

Если метод присутствует – получаем ***исполняемую программу EXE***, в противном случае – библиотеку ***DLL***.

Классы могут быть вложены друг в друга.

Но точка входа должна быть только в одном.

Заготовка консольной программы



```
using System;
namespace ConsoleApplication1
{
    class Class1
    {
        static void Main(string[] args)
        {
            // команды

        }
    }
}
```

Заготовка консольной программы



Директива **using System** разрешает использовать имена стандартных классов из пространства имен System

Средством "навигации" по пространствам имен, которое позволяет сокращать имена классов, является оператор

using <ИмяПространстваИмен>;

В приложении может объявляться собственное пространство имен, а также могут использоваться ранее объявленные пространства.

Ключевое слово **namespace** создает для проекта собственное пространство имен, названное по умолчанию **ConsoleApplication1.**

Это сделано для того, чтобы можно было давать программным объектам имена, не заботясь о том, что они могут совпасть с именами в других пространствах имен.

Заготовка консольной программы


В заготовке программы всего один класс, которому по умолчанию задано имя Class1.

Описание класса начинается с ключевого слова **class,** за которым следуют его **имя** и далее в фигурных скобках — список элементов класса (его данных и функций, называемых также методами).

В данном случае внутри класса только один элемент — метод **Main.** Каждое приложение должно содержать метод Main — с него начинается выполнение программы.

Все методы описываются по единым правилам.

Упрощенный синтаксис метода:

[спецификаторы] тип имя_метода ([параметры])
{ тело метода: *действия, выполняемые методом*
}

Метод **Main()** может быть определен как ***public*** и как ***static***.

Ключевое слово ***public*** в определении метода означает, что этот метод будет доступен извне.

Ключевое слово ***static*** говорит о том, что этот метод позиционируется на уровне класса, а не отдельного объекта и будет доступен даже тогда, когда еще не создано ни одного экземпляра объекта данного класса.

```
using System;
// program1 в C#
namespace HelloWorld
{
    class Hello
    {
        static void Main()
        {
        string myName;
        Console.WriteLine("введите свое имя
                                пожалуйста!");
        myName = Console.ReadLine( );
        Console.WriteLine("Hello{0}",myName);
        }
    }
}
```

Метод Main

Метод ***Main()*** содержит :

- описание строковой переменной;

-вызов метода ***WriteLine()*** класса ***Console*** из пространства имен ***System*** для вывода сообщения на экран «о приглашении ввести имя»

-организация ввода имени;

-вывод сообщения.

Использование using позволяет вместо полного имени класса **System.Console** записать короткое имя **Console.**

Спасибо за внимание!