БАЗОВЫЕ ЭЛЕМЕНТЫ ЯЗЫКА C#

Состав языка

Алфавит – совокупность допустимых в языке символов. Алфавит языка C# включает:

1. прописные и строчные латинские буквы и буквы национальных алфавитов (включая кириллицу);
2. арабские цифры от 0 до 9, шестнадцатеричные цифры от A до F;
3. специальные знаки: " { } , | ; [ ] ( ) + - / % * . \ ' : ? < = > ! & ~ ^ @ _
4. пробельные символы: пробел, символ табуляции, символ перехода на новую строку.

Из символов алфавита формируются лексемы языка: идентификаторы, ключевые (зарезервированные) слова, знаки операций, константы, разделители (скобки, точка, запятая, пробельные символы).

Границы лексем определяются другими лексемами, такими, как разделители или знаки операций. В свою очередь лексемы входят в состав выражений (выражение задает правило вычисления некоторого значения) и операторов (оператор задает законченное описание некоторого действия).

Идентификатор – это имя программного элемента: константы, переменной, метки, типа, класса, объекта, метода и т.д. Идентификатор может включать латинские буквы и буквы национальных алфавитов, цифры и символ подчеркивания. Прописные и строчные буквы различаются, например, myname, myName и MyName — три различных имени. Первым символом идентификатора может быть буква или знак подчеркивания, но не цифра.

Пробелы и другие разделители внутри имен не допускаются. Язык C# не налагает никаких ограничений на длину имен, однако для удобства чтения и записи кода не стоит делать их слишком длинными.

Для улучшения читабельности кода программным элементам следует давать осмысленные имена, составленные в соответствии с определенными правилами. Существует несколько видов нотаций – соглашений о правилах создания имен.

В нотации Pascal каждое слово, входящее в идентификатор, начинается с заглавной буквы. Например:Age, LastName, TimeOfDeath.

Венгерская нотация отличается от предыдущей наличием префикса, соответствующего типу величины. Например: fAge, sName, iTime.

В нотации Camel с заглавной буквы начинается каждое слово идентификатора, кроме первого. Например: age, lastName, timeOfDeath.

Наиболее часто используются нотации Pascal или Camel. Однако в простых программах будут использоваться однобуквенные переменные.

Ключевые слова – это зарезервированные идентификаторы, которые имеют специальное значение для компилятора, например, static, int и т.д. Ключевые слова можно использовать только по прямому назначению. Однако если перед ключевым словом поставить символ @, например, @int, @static, то полученное имя можно использовать в качестве идентификатора. С полным перечнем ключевых слов и их назначением можно ознакомиться в справочной системе C#.

Замечание. Другие лексемы (знаки операций и константы), а также правила формирования выражений и различные виды операторов будут рассмотрены чуть позже.

Типы данных

C# является языком со строгой типизацией. В нем необходимо объявлять тип всех создаваемых программных элементов (например, переменных, объектов, окон, кнопок и т.д.), что позволяет среде CLR предотвращать возникновение ошибок, следя за тем, чтобы объектам присваивались значения только разрешенного типа. Тип программного элемента сообщает компилятору о его размере (например, тип int показывает, что объект занимает 4 байта) и возможностях (например, кнопка может быть нарисована, нажата и т. д.).

В C# типы делятся на три группы:

1. *базовые* типы – предлагаемые языком;

2. типы, *определяемые пользователем;*

3. анонимные типы - типы, которые автоматически создаются на основе инициализаторов объектов (начиная с версии C# 3.0).

Кроме того, типы C# разбиваются на две другие категории: *размерные типы* (value type) и *ссылочные типы* (reference type). Почти все базовые типы являются размерными типами. Исключение составляют типы Object и String, которые являются базовыми, но ссылочными типами данных. Все пользовательские типы, кроме структур, являются ссылочными. Дополнительно к упомянутым типам, язык C# поддерживает типы *указателей,* однако они используются только с неуправляемым кодом.

Принципиальное различие между размерными и ссылочными типами состоит в способе хранения их значений в памяти. В первом случае фактическое значение хранится в стеке (или как часть большого объекта ссылочного типа). Адрес переменной ссылочного типа тоже хранится в стеке, но сам объект хранится в куче.

Стек – это структура, используемая для хранения элементов по принципу LIFO (Last input – first output или *первым пришел - последним ушел)*. В данном случае под стеком понимается область памяти, обслуживаемая процессором, в которой хранятся значения локальных переменных. Куча – область памяти, используемая для хранения данных, работа с которыми реализуется через указатели и ссылки. Память для размещения таких данных выделяется программистом динамически, а освобождается сборщиком мусора.

Сборщик мусора уничтожает программные элементы в стеке через некоторое время после того, как закончит существование раздел стека, в котором они объявлены. То есть, если в пределах блока (фрагмента кода, помещенного в фигурные скобки {}) объявлена локальная переменная, соответствующий программный элемент будет удален по окончании работы данного блока. Объект в куче подвергается сборке мусора через некоторое время после того, как уничтожена последняя ссылка на него.

Язык C# предлагает обычный набор базовых типов, каждому из них соответствует тип, поддерживаемый общеязыковой спецификацией .NET (CLS).

Тип в языке C#	**Размер в байтах**	**Тип .NET**	**Описание**
Базовый тип			
object		Object	Может хранить все что угодно, т.к. является всеобщим предком
Логический тип			
bool	1	Boolean	true или false
Целые типы			
sbyte	1	SByte	Целое со знаком (от -128 до 127)
byte	1	Byte	Целое без знака (от 0 до 255)
short	2	Int16	Целое со знаком (от -32768 до 32767)
ushort	2	UInt16	Целое без знака (от 0 до 65535)
int	4	Int32	Целое со знаком (от -2147483648 до 2147483647)
uint	4	UInt	Целое число без знака (от 0 до 4 294 967 295)
long	8	Int64	Целое со знаком (от -9223372036854775808 до 9223372036854775807)
ulong	8	UInt64	Целое без знака (от 0 до 0ffffffffffffff)

Вещественные типы			
float	4	Single	Число с плавающей точкой двойной точности. Содержит значения приблизительно от $\pm1.5*10^{-45}$ до $\pm3.4*10^{38}$ с 7 значащими цифрами
double	8	Double	Число с плавающей точкой двойной точности. Содержит значения приблизительно от $\pm5.0*10^{-324}$ до $\pm1.7*10^{308}$ с 15-16 значащими цифрами
Символьный тип			
char	2	Char	Символ Unicode
Строковый тип			
string		String	Строка из Unicode-символов
Финансовый тип			
decimal	12	Decimal	Число до 28 знаков с фиксированным положением десятичной точки. Обычно используется в финансовых расчетах и требует суффикса <<m>> или <<M>>

Переменные и константы

Переменная представляет собой типизированную область памяти. Программист создает переменную, объявляя ее тип и указывая имя. При объявлении переменной ее можно инициализировать (присвоить ей начальное значение), а затем в любой момент ей можно присвоить новое значение, которое заменит собой предыдущее.

```
static void Main()
{
        int i=10; //объявление и инициализация целочисленной переменной i
        Console.WriteLine(i); //просмотр значения переменной
        i=100; //изменение значение переменной
        Console.WriteLine(i);
}
```

В языках предыдущего поколения переменные можно было использовать без инициализации. Это могло привести к множеству проблем и долгому поиску ошибок. В языке C# требуется, чтобы переменные были явно проинициализированы до их использования. Проверим этот факт на примере.

```
static void Main()
{
        int i; //объявление переменной без инициализации
        Console.WriteLine(i); //просмотр значения переменной
}
```

При попытке скомпилировать этот пример в списке ошибок будет выведено следующее сообщение: *«Использование локальной переменной i, которой не присвоено значение*).
Инициализировать каждую переменную сразу при объявлении необязательно, но необходимо присвоить ей значение до того, как она будет использована.

Константа, в отличие от переменной, не может менять свое значение. Константы бывают трех видов: *литералы, типизированные константы* и *перечисления.*

В операторе присваивания:

x=32;

число 32 является *литеральной константой*. Его значение всегда равно 32 и его нельзя изменить.

Типизированные константы именуют постоянные значения. Объявление типизированной константы происходит следующим образом:

const <тип> <идентификатор> = <значение>;

Рассмотрим пример:

```
static void Main()
{
        const int i=10; //объявление целочисленной константы i
        Console.WriteLine(i); //просмотр значения константы
        i=100; //ошибка – недопустимо изменять значение константы
        Console.WriteLine(i);
}
```

Задание. Измените программу так, чтобы при объявлении константы не происходила инициализация. Как на это отреагирует компилятор и почему?

Перечисления (enumerations) являются альтернативой константам. Перечисление - это особый размерный тип, состоящий из набора именованных констант (называемых *списком перечисления).*
Синтаксис объявления перечисления следующий:

[атрибуты] [модификаторы] **enum** <имя> [: базовый тип]
{список-перечисления констант(через запятую)};

Замечание*. Атрибуты и модификаторы являются необязательными элементами этой конструкции. Более подробные сведения о их можно найти в дополнительных источниках информации.*
Базовый тип - это тип самого перечисления. Если не указать базовый тип, то по умолчанию будет использован тип int. В качестве базового типа можно выбрать любой целый тип, кроме char. Пример использования перечисления:

```
class Program
{
        enum gradus:int
        {
          min=0,
          krit=72,
          max=100,    //1
        }
        static void Main()
        {
          Console.WriteLine("минимальная температура=" + (int) gradus.min);
          Console.WriteLine("критическая температура=" + (int)gradus.krit);
          Console.WriteLine("максимальная температура=" + (int)gradus.max);
        }
}
```

Замечания
1. В общем случае последнюю запятую в объявлении перечисления можно не ставить (см. строку 1). Но лучше ее поставить: если вам придется добавить еще несколько строк в перечисление, такая предусмотрительность избавит вас от возможных синтаксических ошибок.
2. Запись (int) gradus.min используется для явного преобразования перечисления к целому типу. Если убрать (int), то на экран будет выводиться название констант.
3. Символ + в записи "минимальная температура=" + (int) gradus.min при обращении к методу WriteLine означает, что строка "минимальная температура=" будет «склеена» со строковым представлением значения (int) gradus.min. В результате получится новая строка, которая и будет выведена на экран.

Операции

Полный список операций языка C# в соответствии с их приоритетами (по убыванию приоритетов, операции с разными приоритетами разделены чертой) приведен в Приложении 1. В данном разделе мы подробно рассмотрим только часть операций, остальные операции будут вводиться по мере необходимости.

Замечание. *Операции можно классифицировать по количеству операндов: унарные – воздействуют на один операнд, бинарные – воздействуют на два операнда, тернарные – воздействуют на три операнда. Некоторые символы используются для обозначения как унарных, так и бинарных операций. Например, символ «минус» используется как для обозначения унарной операции – арифметического отрицания, так и для обозначения бинарной операции вычитания. Будет ли данный символ обозначать унарную или бинарную операцию, определяется контекстом, в котором он используется.*

1. Инкремент (++) и декремент(--).

Эти операции имеют две формы записи — *префиксную,* когда операция записывается перед операндом, и *постфиксную* – операция записывается после операнда. Префиксная операция инкремента (декремента) увеличивает (уменьшает) свой операнд и возвращает измененное значение как результат. Постфиксные версии инкремента и декремента возвращают первоначальное значение операнда, а затем изменяют его.

Рассмотрим эти операции на примере.

```
static void Main()
{
   int i = 3, j = 4;
   Console.WriteLine("{0} {1}", i, j);
   Console.WriteLine("{0} {1}", ++i, --j);
   Console.WriteLine("{0} {1}", i++, j--);
   Console.WriteLine("{0} {1}", i, j);
}
```

Результат работы программы:

```
3 4
4 3
4 3
5 2
```

Задание. Выясните, допустимы ли следующие способы записи ++(++i), (i--)--, ++(i--) и т.д. И почему.

2. Операция new. Используется для создания нового объекта. С помощью ее можно создавать как объекты ссылочного типа, так и размерные, например:

```
object z=new object();
int i=new int(); // то же самое, что и int i =0;
```

3. Отрицание:

1. Арифметическое отрицание (-) – меняет знак операнда на противоположный.
2. Логическое отрицание (!) – определяет операцию инверсии для логического типа.

   Рассмотрим эти операции на примере.

```
static void Main()
{
   int i = 3, j=-4;
   bool a = true, b=false;
   Console.WriteLine("{0} {1}", -i, -j);
   Console.WriteLine("{0} {1}", !a, !b);
}
```

Результат работы программы:

```
-3   4
False True
```

Задание. Выясните, допустимы ли следующие способы записи !(-i), -(!a). И почему.

4. Явное преобразование типа. Используется для явного преобразования из одного типа в другой. Формат операции:

(<тип>) <выражение>;

Рассмотрим эту операцию на примере.

```
static void Main()
{
   int i = -4;
   byte j = 4;
   int a = (int)j;  //преобразование без потери точности
   byte b = (byte)i; //преобразование с потерей точности
   Console.WriteLine("{0} {1}", a, b);
}
```

Результат работы программы:

```
4   252
```

Задание. Объясните, почему операция (byte)i вместо ожидаемого значения -4 дала нам в качестве результата значение 252.

Замечание. *В Pascal, C++ и других языках допускается неявное преобразование типов, которое в рамках предыдущего примера позволило бы записать: b=i. В этом случае происходит потеря точности вычислений, о чем компилятор либо "умалчивает", либо сообщает в виде предупреждения. Возможность неявного преобразования чревата вычислительными ошибками, которые очень трудно найти. Чтобы избежать подобных ошибок, в C# запрещены некоторые виды неявных преобразований. Более подробно преобразование типов мы рассмотрим в следующем разделе.*

5. Умножение (*), деление (/) и деление с остатком (%). Операции умножения и деления применимы для целочисленных и вещественных типов данных. Для других типов эти операции применимы, если для них возможно неявное преобразование к целым или вещественным типам. При этом тип результата равен «наибольшему» из типов операндов, но не менее int. Если оба операнда при делении целочисленные, то и результат тоже целочисленный.

Рассмотрим эти операции на примере.

```
static void Main()
{
  int i = 100, j = 15;
  double a = 14.2, b = 3.5;
  Console.WriteLine("{0} {1} {2}", i*j, i/j, i%j);
  Console.WriteLine("{0} {1} {2}", a * b, a / b, a % b);
}
```

Результат работы программы:

```
1500    6    10
49.7    4.05714285714286    0.1999999999999999
```

Задания.

1. Выполните фрагмент программы и объясните полученный результат:

```
double a=100, b=33;
Console.WriteLine(a/b);
double d=100/33;
Console.WriteLine(d);
```

2. Выясните, чему будет равен результат операции, и объясните, как получился данный результат:

а) 1.0/0; б) 1/0

6. Сложение (+) и вычитание (-).Операции сложения и вычитания применимы для целочисленных и вещественных типов данных. Для других типов эти операции применимы, если для них возможно неявное преобразование к целым или вещественным типам.

7. Операции отношения (<, <=, >, >=, ==, !=). Операции отношения сравнивают значения левого и правого операндов. Результат операции логического типа: true – если значения совпадают, false – в противном случае. Рассмотрим операции на примере:

```
static void Main()
{
  int i = 15, j = 15;
  Console.WriteLine(i<j); //меньше
  Console.WriteLine(i<=j);  //меньше или равно
  Console.WriteLine(i>j);  //больше
  Console.WriteLine(i>=j); //больше или равно
  Console.WriteLine(i==j);  //равно
  Console.WriteLine(i!=j);  //не равно
}
```

Результат работы программы:

```
False
True
False
True
True
False
```

Задание. Выясните, чему равен результат данного выражения:
1) 10<25<30 2) true<false
И объясните, как получился данный результат.

8. Логические операции: И (&&), ИЛИ (||).

Логические операции применяются к операндам логического типа.

Результат логической операции И имеет значение истина тогда и только тогда, когда оба операнда принимают значение истина.

Результат логической операции ИЛИ имеет значение истина тогда и только тогда, когда хотя бы один из операндов принимает значение истина.

Рассмотрим операции на примере:

```
static void Main()
 {
   Console.WriteLine("x     y     x и y    x или y");
   Console.WriteLine("{0} {1} {2} {3}", false, false, false&&false, false||false);
   Console.WriteLine("{0} {1} {2} {3}", false, true, false&&true, false||true);

   Console.WriteLine("{0} {1} {2} {3}", true, false, true&&false, true||false);
   Console.WriteLine("{0} {1} {2} {3}", true, true, true&&true, true||true);
 }
```

Результат работы программы:

```
x       y       x и y   x или y
False   False   False   False
False   True    False   True
True    False   False   True
True    True    True    True
```

Замечание. *Фактически была построена таблица истинности для логических операций И и ИЛИ.*

Задание. Объясните, какое значение примет переменная t в данном фрагменте программы:

```
int a=10, b=3;
bool t=(a>=b && a!=2*b || a<0);
```

9. Условная операция.

Формат: (<операнд1>)? <операнд2> : <операнд3>;

Операнд1 – это логическое выражение, которое оценивается с точки зрения его эквивалентности константам *true* и *false*. Если результат вычисления операнда1 равен true, то результатом условной операции будет значение операнда2, иначе — операнда3. Фактически условная операция является сокращенной формой условного оператора if, который будет рассмотрен позже.

Пример использования условной операции:

```
static void Main()
{
   int x=5; int y=10;
   int max = (x > y) ? x : y;
   Console.WriteLine(max);
}
```

Задание. Измените программу так, чтобы:
1) вычислялось наименьшее значение из двух вещественных чисел x и y;
2) если число двузначное, то на экран выводилось «Да», и «Нет» в противном случае.

10. Операции присваивания: =, +=, -= и т.д.

Формат операции *простого присваивания* (=):

операнд_2 = операнд_1;

В результате выполнения этой операции вычисляется значение операнда_1, и результат записывается в операнд_2. Можно связать воедино сразу несколько операторов присваивания, записывая такие цепочки: a=b=c=100. Выражение такого вида выполняется справа налево: результатом выполнения c=100 является число 100, которое затем присваивается переменной b, результатом чего опять является 100, которое присваивается переменной a.

Кроме простой операции присваивания существуют *сложные операции присваивания*, например, умножение с присваиванием (*=), деление с присваиванием (/=), остаток от деления с присваиванием (%=), сложение с присваиванием (+=), вычитание с присваиванием (-=) и т.д.

В сложных операциях присваивания, например, при *сложении с присваиванием*, к операнду_2 прибавляется операнд_1, и результат записывается в операнд_2. То есть, выражение c += a является более компактной записью выражения c = c + a .

Задание. Объясните, какие значения примут переменные t и b после выполнения данного фрагмента программы:

```
int a=10, b=3;
int t=(a++)-b;
int b+=t*a;
```