Основы программирования

Синтаксис языка C#

Операторы
Условные операторы

Пустой оператор

Частный случай выражений является пустой оператор – «;». Используется когда по синтаксису оператор требуется, а по смыслу – нет.

Допустима конструкция:

```
for (int j=1; j<5; j++)
    {;;;};
```

которая может рассматриваться как задержка по времени.

Составной оператор или блок

- *Блок* или *составной оператор, -* это последовательность описаний и операторов, заключенная в фигурные скобки.
- Блок воспринимается компилятором как один оператор и может использоваться всюду, где синтаксис требует одного оператора, а алгоритм - нескольких.
- Блок может содержать один оператор или быть пустым.

Условные операторы (операторы ветвления)

Обеспечивают выполнение разных последовательностей операторов в зависимости от исходных данных:

- оператор `if` – передает управление на одну из двух ветвей;
- оператор `switch` – передает управление на одну из произвольного числа ветвей;
- тернарный оператор `? : ;`.

Условный оператор. Полная форма

```
if (условие)
{
      оператор_1;
}
else
{
      оператор_2;
}
```

где

`условие` - логическое или арифметическое выражение, истинность которого проверяется;

`оператор` - простой или составной оператор.

Условный оператор. Пример

```
int num1 = 8, num2 = 6;

if(num1 > num2)
{
   Console.WriteLine("Число {0} больше числа {1}",
                        num1, num2);
}
else
{
    Console.WriteLine("Число {0} меньше числа {1}",
                        num1, num2);
}
```

Условный оператор. Полная форма

```
if (условие)
{
    оператор1;
    оператор2;
}
else
{
    оператор1;
    оператор2;
}
```

```
if (условие) {
    оператор1;
    оператор2;
    }
else {
    оператор1;
    оператор2;
    }
```

(из спецификаци на C#)

Конструкция if - else-if

```
if   (условие_1)
     оператор_1;
else if (условие_2)
     оператор_2;
else if (условие_3)
     оператор_3;
else
     оператор_4;
```

Конструкция if - else-if

- Условия вычисляются сверху вниз.
- Как только обнаружится истинное условие, то будет выполнен оператор, связанный с соответствующей ветвью, а все остальные опущены.
- Если ни одно из условий не является истинным, то будет выполнен последний оператор `else`.
- Если последний оператор `else` не задан, а все остальные ложны, то никакое действие не будет выполнено.

Пример использования if - else-if

```
int num1 = 8, num2 = 6;
if(num1 > num2)    {
    Console.WriteLine("Число {0} больше числа {1}",
                          num1, num2);
}
else if (num1 < num2)    {
    Console.WriteLine("Число {0} меньше числа {1}“,
                  num1, num2);
}
else  {
    Console.WriteLine("Число {0} равно числу {1}“,
                  num1, num2);
}
```

Пример использования if - else-if

```
int num1 = 8, num2 = 6;
if(num1 > num2)    {
    Console.WriteLine("Число {0} больше числа {1}",
                          num1, num2);
}
else if (num1 < num2)    {
    Console.WriteLine("Число {0} меньше числа {1}“,
                  num1, num2);
}
else  {
    Console.WriteLine("Число {0} равно числу {1}“,
                  num1, num2);
}
```

Пример использования двух условий

```
int num1 = 8, num2 = 6;

if(num1 > num2 && num1==8)
{
   Console.WriteLine("Число {0} больше числа {1}", num1, num2);
}
```

Особенность сравнения вещественных чисел

```
double a, b; ...

if (a==b)                          // не надежно
if (Math.Abs(a-b)<=0.0001)         // надежно
```

Оператор switch. Синтаксис

```
switch (выражение)  {
  case константа_1:
     операторы_1; break;
  case константа_2:
     операторы_2; break;
  …
  default:
     операторы_3; break;
}
```

где выражение имеет тип целочисленный или строковый.

Константы в `case` должны иметь тот же тип, что и *`switch`*-выражение.

Оператор switch. Синтаксис

Поскольку последний оператор `case`-ветви является *оператором перехода* (чаще всего это оператор `break`), то обычно он завершает выполнение *оператора* `switch`.
При его отсутствии управление "проваливается" в следующую `case`-ветвь.

Мнение:
«Оператор `switch` - это самый неудачный оператор языка `C#` как с точки зрения синтаксиса, так и семантики. Неудачный синтаксис порождает запутанную семантику, являющуюся источником плохого стиля программирования».

Оператор switch. Пример 1

```
int num = Int32.Parse(Console.ReadLine());

switch (num % 3) {
  case 1:
     Console.WriteLine(”Остаток = 1”);     break;
  case 2:
     Console.WriteLine(”Остаток = 2”);     break;
  ...
  default:
     Console.WriteLine(”Остаток = 0”);     break;
  }
```

Оператор switch. Пример 2

```
Console.WriteLine("Введите язык C# или C++)");
string myLanguage = Console.ReadLine();
switch (s) {
  case "C#":
    Console.WriteLine("Вы выбрали язык C#");
    break;
  case "C++":
    Console.WriteLine("Вы выбрали язык C++");
    break;
  default:
    Console.WriteLine("Такой язык я не знаю");
    break; }
```

Оператор switch. Замечания

- Считается ошибкой, если последовательность операторов, относящаяся к одной `case`-ветви, переходит в последовательность инструкций, связанную со следующей.
- Поэтому `case`-последовательности чаще всего оканчиваются инструкцией `break`.
- Оператор `break`, завершающий последовательность `case`-инструкций, приводит к выходу из всей конструкции `switch` и передаче управления к следующей инструкции, находящейся вне конструкции `switch`.

Вложенные операторы switch

```
switch(chl) {
case 'A':
  Console.WriteLine("буква A-внешняя");
  switch(ch2) {
    case 'A':
      Console.WriteLine("буква A-внутренняя");
      break;
      case 'B': // ...
  } // конец внутренней инструкции switch,
break;
case 'B': // ...
```

Оператор switch. Пример 3

```
int i ;
for(i=1; i < 4; i++)
switch(i) {
case 1:
case 2:
     Console.WriteLine("i равно 1 или 2");
     break;
case 3:
     Console.WriteLine (" i равно 3 ");
     break;
}
```

Тернарный оператор. Синтаксис

```
выражение1 ? выражение2 : выражение3;
```

где

`выражение1` имеет тип `bool`;

`выражение2` и `выражение3` имеют одинаковый тип или приводятся к одному типу.

Тернарный оператор. Примеры

```
int a=11, b=4;
int max = b > a ? b : a;
Console.WriteLine(max);          // = 11


if ( i != 0 ? true : false)
   Console.WriteLine("10/"+i+равно"+10/i);
```

Тернарный оператор. Тип результата

- если выражения одного тип, то он становится типом результата;
- иначе, если существует неявное преобразование тип от `выражения2` к `выражению3`, но не наоборот, то тип результата операции тип `выражения3`;
- иначе, если существует неявное преобразование тип от `выражения3` к `выражению2`, но не наоборот, то тип результата операции тип `выражения2`;
- иначе ошибка компиляции.