Лекция Схема работы циклов и Операторы ветвления

Программа на языке C# состоит из последовательности операторов, каждый из которых определяет законченное описание некоторого действия и заканчивается точкой с запятой. Все *операторы* можно разделить на 4 группы: *операторы* следования, *операторы* ветвления, *операторы цикла* и *операторы передачи управления*.

Операторы цикла

Операторы цикла используются для организации многократно повторяющихся вычислений. К операторам цикла относятся: *цикл с предусловием* while, *цикл с постусловием* do while, цикл с параметром for и цикл перебора foreach..

Цикл с предусловием while

Оператор цикла while организует выполнение одного оператора (простого или составного) неизвестное заранее число раз. Формат цикла while:

while (B) S;

где B - выражение, истинность которого проверяется (условие завершения цикла); S - тело цикла - оператор (простой или составной).

Перед каждым выполнением тела цикла анализируется значение выражения B: если оно истинно, то выполняется тело цикла, и управление передается на повторную проверку условия B ; если значение B ложно - цикл завершается и управление передается на оператор, следующий за оператором S.

Если результат выражения B окажется ложным при первой проверке, то тело цикла не выполнится ни разу. Отметим, что если условие B во время работы цикла не будет изменяться, то возможна ситуация зацикливания, то есть невозможность выхода из цикла. Поэтому внутри тела должны находиться операторы, приводящие к изменению значения выражения B так, чтобы цикл мог корректно завершиться.

В качестве иллюстрации выполнения цикла while рассмотрим программу вывода на экран целых чисел из интервала от 1 до n.

```
static void Main()
{
  Console.Write("N= ");
  int n=int.Parse(Console.ReadLine());
  int i = 1;
  while (i <= n)          //пока i меньше или равно n
  Console.Write(" "+ i++ ); //выводим i на экран, затем увеличиваем его на 1
}
```

Результаты работы программы:

```
n           ответ
10           1  2  3  4  5  6  7  8  9  10
```

Цикл с постусловием do while

Оператор цикла do while также организует выполнение одного оператора (простого или составного) неизвестное заранее число раз. Однако в отличие от цикла while условие завершения цикла проверяется после выполнения тела цикла. Формат цикла do while:

```
do  S while (B);
```

где B - выражение, истинность которого проверяется (условие завершения цикла); S - тело цикла - оператор (простой или блок).

Сначала выполняется оператор S, а затем анализируется значение выражения B: если оно истинно, то управление передается оператору S, если ложно - цикл завершается, и управление передается на оператор, следующий за условием B. Так как условие B проверяется после выполнения тела цикла, то в любом случае тело цикла выполнится хотя бы один раз.

В операторе do while, так же как и в операторе while, возможна ситуация зацикливания в случае, если условие B всегда будет оставаться истинным.

В качестве иллюстрации выполнения цикла do while рассмотрим программу вывода на экран целых чисел из интервала от 1 до n.

```
static void Main()
{
  Console.Write("N= ");
  int n=int.Parse(Console.ReadLine());
  int i = 1;
  do
    Console.Write(" " + i++);
    //выводим i на экран, затем увеличиваем его на 1
  while (i <= n); //пока i меньше или равно n
}
```

Цикл с параметром for

Цикл с параметром имеет следующую структуру:

```
for ( <инициализация>; <выражение>; <модификация>) <оператор>;
```

Инициализация используется для объявления и/или присвоения начальных значений величинам, используемым в цикле в качестве параметров (счетчиков). В этой части можно записать несколько операторов, разделенных запятой. Областью действия переменных, объявленных в части инициализации цикла, является цикл и *вложенные блоки*. Инициализация выполняется один раз в начале исполнения цикла.

Выражение определяет условие выполнения цикла: если его результат истинен, цикл выполняется. Истинность выражения проверяется перед каждым выполнением тела цикла, таким образом, цикл с параметром реализован как *цикл с предусловием*. В блоке выражение через запятую можно записать несколько логических выражений, тогда запятая равносильна операции логическое И (&&).

Модификация выполняется после каждой итерации цикла и служит обычно для изменения параметров цикла. В части модификация можно записать несколько операторов через запятую.

Оператор (простой или составной) представляет собой тело цикла.

Любая из частей оператора *for* (инициализация, выражение, модификация, оператор) может отсутствовать, но точку с запятой, определяющую позицию пропускаемой части, надо оставить.

```
static void Main()
{
Console.Write("N= ");
```

```
int n=int.Parse(Console.ReadLine());
for (int i=1; i<=n;)     //блок модификации пустой
Console.Write(" " + i++);
}
```

Вложенные циклы

Циклы могут быть простые или вложенные (кратные, циклы в цикле). Вложенными могут быть циклы любых типов: while, do while, for. Каждый внутренний цикл должен быть полностью вложен во все внешние циклы. "Пересечения" циклов не допускаются.

Рассмотрим пример использования вложенных циклов, который позволит вывести на экран числа следующим образом:

```
22222
22222
22222
22222
static void Main()
{
  for (int i = 1; i <= 4; ++i, Console.WriteLine())     //1
  for (int j=1; j<=5; ++j)
  Console.Write(" " + 2);
}
```

Замечание. В строке 1 в блоке модификации содержится два оператора ++i и Console.WriteLine(). В данном случае после каждого увеличения параметра i на 1 курсор будет переводиться на новую строку.

Оператор foreach

Можно уже по одному названию (в переводе это "для каждого") догадаться, что делает этот оператор. Это — оператор организации цикла, он похож на оператор for. У него есть тело и заголовок. Правила формирования тела — как у for. Заголовок несколько другой. Например, запись

```
foreach(int item in m)
{
тело
}
```

читается так: "для каждого элемента item из массива m выполнить тело". При этом элементы массива m последовательно выбираются в переменную item (здесь предполагается, что массив содержит элементы типа int).
Пусть у нас есть массив из 5 элементов типа int. Составим программу инициализации и суммирования всех элементов этого массива. Текст программы приведен в листинге.

```
using System;
namespace app16_foreach
{
class Program
{
public static void Main()
{
   int []A = new int [5];
   string s;
   int i=0;
Console.WriteLine("Введите числа массива, после " + "каждого <Enter>, в конце <Enter> <Ctrl+z> ");
```

```
while(true)
{
  s=Console.ReadLine();
  if(s==null || i > (A.Length-1))
  break;
  A[i]=Convert.ToInt32(s);
   i++;
}
Console.WriteLine("Введено чисел — {0}",i);
i=0;
foreach(int j in A) i += j;
Console.WriteLine("Сумма элементов массива = {0}", i);
Console.Read();
}
}
}
```

Первый раз было введено пять элементов, а второй — только три. Во втором случае сумма подсчитана верно. Это говорит о том, что в массиве недостающие элементы — нулевые. Комментирования заслуживает только оператор if, в заголовке которого проверяется, надо ли выходить из бесконечного цикла. Первая часть неравенства срабатывает, когда нажата комбинация клавиш <Ctrl>+<Z>, вторая, если превзойден размер массива. То есть лишние члены в массив не попадут и прерывания программы не будет. В этом же месте мы видим, что для массива, как и для строки, существует функция, которая возвращает длину (Length). У массива длина — это количество его элементов. Отметим, что foreach работает только на чтение элементов из массива. Отметим также, что foreach так просто работает только для массивов из базовых типов данных (чисел, строк).

Операторы безусловного перехода

В C# есть несколько операторов, изменяющих естественный порядок выполнения команд: *оператор безусловного перехода* goto, оператор выхода break, оператор перехода к следующей итерации цикла continue, оператор возврата из метода return и оператор генерации исключения throw.

Оператор безусловного перехода goto

Оператор безусловного перехода goto имеет формат:

goto <метка>;

В теле той же функции должна присутствовать ровно одна конструкция вида:

<метка>: <оператор>;

Оператор *goto* передает управление на помеченный меткой оператор. Рассмотрим пример использования оператора goto:

```
static void Main()
{
 float x;
 metka: Console.WriteLine("x="); //оператор, помеченный меткой
 x = float.Parse(Console.ReadLine());
 if (x!=0) Console.WriteLine("y({0})={1}", x, 1 / x );
 else
 {
  Console.WriteLine("функция не определена");
```

```
    goto metka; // передача управление метке
  }
}
```

Следует учитывать, что использование оператора goto затрудняет чтение больших по объему программ, поэтому использовать метки нужно только в крайних случаях, например, в операторе switch.

Оператор выхода break

Оператор break используется внутри операторов ветвления и цикла для обеспечения перехода в точку программы, находящуюся непосредственно за оператором, внутри которого находится break.

Мы уже применяли оператор break для выхода из оператора switch, аналогичным образом он может применяться для выхода из других операторов.

Оператор перехода к следующей итерации цикла continue

Оператор перехода к следующей итерации цикла continue пропускает все операторы, оставшиеся до конца тела цикла, и передает управление на начало следующей итерации (повторение тела цикла). Рассмотрим оператор continue на примере.

```
static void Main()
{
  Console.WriteLine("n=");
  int n = int.Parse(Console.ReadLine());
  for (int i = 1; i <= n; i++)
  {
    if (i % 2 == 0) continue;
    Console.Write(" " + i);
  }
}
```

Операторы следования

Операторы следования выполняются компилятором в естественном порядке: начиная с первого до последнего. К операторам следования относятся: выражение и составной оператор.

Любое *выражение*, завершающееся точкой с запятой, рассматривается как оператор, выполнение которого заключается в вычислении значения выражения или выполнении законченного действия, например, вызова метода. Например:

```
++i;                        //оператор инкремента
x+=y;                       //оператор сложение с присваиванием
Console.WriteLine(x);       //вызов метода
x=Math.Pow(a,b)+a*b;        //вычисление сложного выражения
```

Частным случаем оператора выражения является *пустой оператор* ; Он используется тогда, когда по синтаксису оператор требуется, а по смыслу - нет. В этом случае лишний символ ; является пустым оператором и вполне допустим, хотя и не всегда безопасен. Например, случайный символ ; после условия оператора while или if может совершенно поменять работу этого оператора.

Составной оператор или *блок* представляет собой последовательность операторов, заключенных в фигурные скобки {}. Блок обладает собственной *областью видимости*: объявленные внутри блока имена доступны только внутри данного блока или блоков, вложенных в него. Составные операторы

применяются в случае, когда правила языка предусматривают наличие только одного оператора, а логика программы требует нескольких операторов. Например, тело цикла while должно состоять только из одного оператора. Если заключить несколько операторов в фигурные скобки, то получится блок, который будет рассматриваться компилятором как единый оператор.

Операторы ветвления

Операторы ветвления позволяют изменить порядок выполнения операторов в программе. К операторам ветвления относятся условный оператор if и оператор выбора switch.

Условный оператор if

Условный оператор *if* используется для разветвления процесса обработки данных на два направления. Он может иметь одну из форм: *сокращенную* или *полную*.

Форма *сокращенного оператора if*:

```
if (B)  S;
```

где B - логическое или арифметическое выражение, истинность которого проверяется; S - оператор: простой или составной.

При выполнении сокращенной формы оператора if сначала вычисляется выражение B, затем проводится анализ его результата: если B истинно, то выполняется оператор S ; если B ложно, то оператор S пропускается. Таким образом, с помощью сокращенной формы оператора if можно либо выполнить оператор S, либо пропустить его.

Форма *полного оператора if*:

```
if (B) S1; else S2;
```

где B - логическое или арифметическое выражение, истинность которого проверяется; S1, S2 - оператор: простой или составной.

При выполнении полной формы оператора if сначала вычисляется выражение B, затем анализируется его результат: если B истинно, то выполняется оператор S1, а оператор S2 пропускается; если B ложно, то выполняется оператор S2, а S1 - пропускается. Таким образом, с помощью полной формы оператора if можно выбрать одно из двух альтернативных действий процесса обработки данных.

Рассмотрим несколько примеров записи условного оператора if:

```
if (a > 0) x=y;                    // Сокращенная форма с простым оператором
if (++i) {x=y; y=2*z;}                    // Сокращенная форма с составным оператором
if (a > 0 || b<0) x=y; else  x=z;          // Полная форма с простым оператором
if (i+j-1) { x= 0; y= 1;} else {x=1; y:=0;}        // Полная форма с составными операторами
```

Рассмотрим пример использования условного оператора.

```
static void Main()
{
  Console.Write("x= ");
  float x = float.Parse(Console.ReadLine());
            Console.Write("y=");
  float y = float.Parse(Console.ReadLine());
  if (x < y ) Console.WriteLine("min= "+x);
```

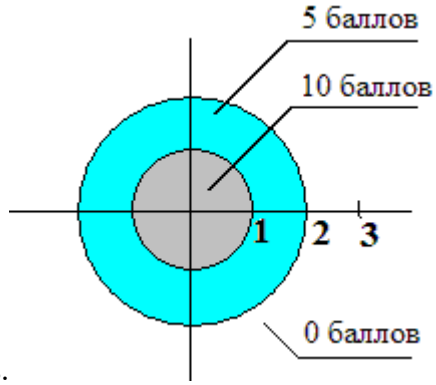
```
  else Console.WriteLine("min= "+y);
}
```

Результат работы программы:

x	y	min
0	0	0
1	-1	-1
-2	2	-2

Задание. Измените программу так, чтобы вычислялось наибольшее значение из x и y.

Операторы S1 и S2 могут также являться операторами if. Такие операторы называют вложенными. При этом ключевое слово else связывается с ближайшим предыдущим словом if, которое еще не связано ни с одним else. Рассмотрим пример программы, использующей вложенные условные операторы.



Пример: Дана мишень.

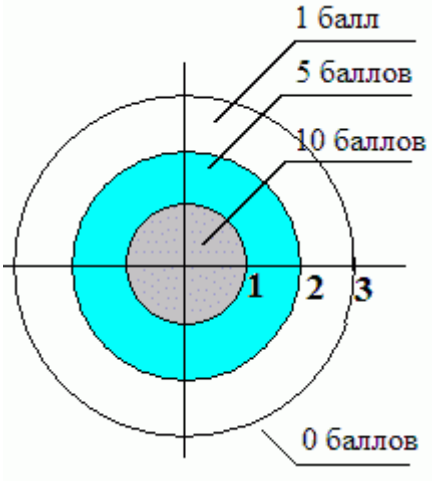
Подсчитать количество очков после выстрела по данной мишени.

```
static void Main()
{
  int Ball=0;
            Console.Write("x= ");
  float x = float.Parse(Console.ReadLine());
  Console.Write("y= ");
  float y = float.Parse(Console.ReadLine());
  if (x * x + y * y <=1) Ball = 10;        //окружность с радиусом 1
  else  if (x * x + y * y <= 4) Ball = 5; //окружность с радиусом 2
  Console.WriteLine("Ball= "+ Ball);
}
```

Результат работы программы:

x	y	Ball
0	0	10
1	-1	5
-2	2	0

Задание.Измените программу так, чтобы подсчитывалось количество очков для мишени вида



Оператор выбора switch

Оператор выбора switch предназначен для разветвления процесса вычислений по нескольким направлениям. Формат оператора:

```
switch ( <выражение> )
{
 case <константное_выражение_1>:
  [<оператор 1>];  <оператор перехода>;
 case <константное_выражение_2>:
  [<оператор 2>]; <оператор перехода>;
 ...
 case <константное_выражение_n>:
  [<оператор n>]; <оператор перехода>;
  [default: <оператор>; ]
}
```

Замечание. Выражение, записанное в квадратных скобках, является необязательным элементом в *операторе switch*. Если оно отсутствует, то может отсутствовать и оператор перехода.

Выражение, стоящее за ключевым словом switch, должно иметь арифметический, символьный, строковый тип или тип указатель. Все константные выражения должны иметь разные значения, но их тип должен совпадать с типом выражения, стоящим после switch или приводиться к нему. Ключевое слово case и расположенное после него константное выражение называют также меткой case.

Выполнение оператора начинается с вычисления выражения, расположенного за ключевым словом switch. Полученный результат сравнивается с меткой case. Если результат выражения соответствует метке case, то выполняется оператор, стоящий после этой метки, за которым обязательно должен следовать оператор перехода: break, goto и т.д. При использовании оператора break происходит выход из switch и управление передается оператору, следующему за switch. Если же используется оператор goto, то управление передается оператору, помеченному меткой, стоящей после goto.

Пример. По заданному виду арифметической операции (сложение, вычитание, умножение и деление) и двум операндам, вывести на экран результат применения данной операции к операндам.

```
static void Main()
{
 Console.Write("OPER= ");
 char oper=char.Parse(Console.ReadLine());
 bool ok=true;
 Console.Write("A= ");
```

```
  int a=int.Parse(Console.ReadLine());
  Console.Write("B= ");
  int b=int.Parse(Console.ReadLine());
  float res=0;
  switch (oper)
  {
    case '+': res = a + b; break;              //1
    case '-': res = a - b; break;
    case '*': res = a * b; break;
    case ':': if (b != 0)
    {
      res = (float)a / b; break;
    }
    else goto default;
    default: ok = false; break;
  }
  if (ok) Console.WriteLine("{0} {1} {2} = {3}", a, oper, b, res);
  else Console.WriteLine("error");
}
```

Результат выполнения программы:

oper	x	y	rez
+	4	5	9
:	4	0	error
%	4	3	error