Операторы выбора

Операторы выбора предназначены для условного управления потоком выполнения программы.

Оператор if

Оператор `if` выполняет некоторый оператор, если результатом вычисления выражения `bool` оказывается `true`. Например:

```
if (5 < 2 * 3)
  Console.WriteLine ("true");    // true
```

Оператором может быть блок кода:

```
if (5 < 2 * 3)
{
  Console.WriteLine ("true"); // true
  Console.WriteLine ("...")
}
```

Конструкция else

Оператор `if` может дополнительно содержать конструкцию

```
else: if (2 + 2 == 5)
  Console.WriteLine ("Не вычисляется");
  else
  Console.WriteLine ("False");    // False
```

Внутрь конструкции `else` можно помещать другой оператор `if`:

```
if (2 + 2 == 5)
  Console.WriteLine ("Не вычисляется");
Else if (2 + 2 == 4)
       Console.WriteLine ("Вычисляется"); // Вычисляется
```

Изменение потока выполнения с помощью фигурных скобок

Конструкция `else` всегда применяется к непосредственно предшествующему оператору `if` в блоке операторов. Например:

```
if  (true)
  if (false)
       Console.WriteLine();
    else
       Console.WriteLine ("выполняется");
```

Это семантически идентично следующему коду:

```
if (true)
{
  if (false)
    Console.WriteLine();
  else
    Console.WriteLine ("выполняется");
}
```

Переместив фигурные скобки, поток выполнения можно изменить:

```
if (true)
{
  if (false)
    Console.WriteLine(
    );
}
else
  Console.WriteLine ("не выполняется");
```

В языке C# отсутствует аналог ключевого слова «elseif»; однако приведенный ниже шаблон позволяет достичь того же результата:

```
static void TellMeWhatICanDo (int age)
{
  if (age >= 35)
    Console.WriteLine ("Ты можешь стать президентом!");
  else if (age >= 21)
    Console.WriteLine ("Ты можешь пить!");
  else if (age >= 18)
    Console.WriteLine ("Ты можешь голосовать!");
    else
    Console.WriteLine ("Ты должен ждать!");
}
```

Оператор switch

Операторы `switch` позволяют организовать ветвление потока выполнения программы на основе выбора из возможных значений, которые переменная может принимать. Операторы `switch` могут дать в результате более ясный код, чем множество операторов `if`, потому что они требуют только однократной оценки выражения. Например:

```
static void ShowCard (int cardNumber)
{
  switch (cardNumber)
  {
    case 13:
      Console.WriteLine ("Король"); break;
    case 12:
      Console.WriteLine ("Дама"); break;
    case 11:
      Console.WriteLine ("Валет"); break;
    default:    // Любое другое значение cardNumber
      Console.WriteLine (cardNumber);
      break;
  }
}
```

Значения в каждом выражении `case` должны быть константами, что ограничивает разрешенные типы встроенными целочисленными типами, типами `bool`, `char` и `enum`, а также типом `string`. В конце каждой конструкции `case` необходимо явно указывать, куда выполнение должно передаваться дальше, с помощью одного из операторов перехода. Ниже перечислены варианты:

- `break` (переход в конец оператора `switch`);
- `goto case x` (переход на другую конструкцию `case`);
- `goto default` (переход на конструкцию `default`);
- любой другой оператор перехода, в частности, `return`, `throw`, `continue` или `goto` *метка*.

Если для нескольких значений должен выполняться тот же самый код, то конструкции `case` можно записать последовательно:

```
switch (cardNumber)
{
  case 13:
  case 12:
  case 11:
    Console.WriteLine ("Фигурная карта"); break;
  default:
    Console.WriteLine ("Нефигурная карта"); break;
}
```

Такая особенность оператора `switch` может иметь решающее значение в плане получения более ясного кода, чем в случае множества операторов `if-else`.

Оператор switch с шаблонами (C# 7)

В версии C# 7 можно переключаться на основе *типа*:

```
static void TellMeTheType (object x)
{
  switch (x)
  {
    case int i:
      Console.WriteLine ("Это целочисленное значение!"); break;
    case string s:
      Console.WriteLine (s.Length); // Можно использовать s
      break;
    case bool b when b == true // Выполняется, когда b равно true
      Console.WriteLine ("True");
       break;
    case null:     // В версии C# 7 можно также переключаться по null
      Console.WriteLine ("null");
      break;
  }
}
```

(Тип `object` допускает переменную любого типа.)

В каждой конструкции `case` указывается тип, с которым следует сопоставлять, и

переменная, которой необходимо присвоить типизированное значение в случае успешного совпадения. В отличие от констант ограничения на применяемые типы отсутствуют. В необязательной конструкции `when` указывается условие, которое должно быть удовлетворено, чтобы произошло совпадение для `case`. Порядок следования конструкций `case` важен, когда производится переключение по типу (что отличается от случая переключения по константам). Исключением из этого правила является конструкция `default`, которая выполняется последней независимо от того, где она находится.

Можно указывать несколько конструкций `case` подряд. Вызов `Console.WriteLine()` в приведенном ниже коде будет выполняться для значения любого типа с плавающей точкой, которое больше 1000:

```
switch (x)
{
  case float f when f > 1000:
  case double d when d > 1000:
  case decimal m when m > 1000:
    Console.WriteLine ("Переменные f, d и m находятся вне области
видимости");
    break;
```

В этом примере компилятор позволяет задействовать переменные `f`, `d` и `m` *только* в конструкциях `when`. Во время вызова метода `Console.WriteLine()` неизвестно, какой из трех переменных будет присвоено значение, поэтому компилятор помещает их все за пределы области видимости.

Операторы итераций

Язык C# позволяет выполнять последовательность операторов повторяющимся образом с помо- щью операторов `while`, `do-while`, `for` и `foreach`.

Циклы while и do-while

Циклы `while` многократно выполняют код в своем теле до тех пор, пока результатом вычисления выражения `bool` является `true`. Выражение проверяется *перед* выполнением тела цикла. Например, следующий код выведет `012`:

```
int i = 0;
while (i < 3)
{ // Фигурные скобки не обязательны
  Console.Write (i++);
}
```

Циклы `do-while` отличаются по функциональности от циклов `while` только тем, что выра- жение в них проверяется *после* выполнения блока операторов (гарантируя, что блок выполняется, по крайней мере, один раз). Ниже приведен предыдущий пример, переписанный для использования цикла `do-while`:

```
int i = 0; do
{
  Console.WriteLine (i++);
}
```

```
while (i < 3);
```

Циклы for

Циклы `for` похожи на циклы `while`, но имеют специальные конструкции для *инициализации* и *итерирования* переменной цикла. Цикл `for` содержит три конструкции:

```
for (конструкция-инициализации; конструкция-условия; конструкция-итерация)
     оператор-или-блок-операторов
```

Часть *конструкция-инициализации* выполняется перед началом цикла и обычно инициали- зирует одну или больше переменных *итерации*.

Часть *конструкция-условия* представляет собой выражение типа `bool`, которое проверяется *перед* каждой итерацией цикла. Тело цикла выполняется до тех пор, пока условие дает `true`.

Часть *конструкция-итерации* выполняется *после* каждой итерации цикла. Эта часть обычно применяется для обновления переменной итерации.

Например, следующий код выводит числа от 0 до 2:

```
for (int i = 0; i < 3; i++)
  Console.WriteLine (i);
```

Показанный далее код выводит первые 10 чисел Фибоначчи (где каждое число является суммой двух предыдущих):

```
for (int i = 0, prevFib = 1, curFib = 1; i < 10; i++)
{
  Console.WriteLine (prevFib);
  int newFib = prevFib + curFib;
  prevFib = curFib;
  curFib = newFib;
}
```

Любая из трех частей оператора `for` может быть опущена. Бесконечный цикл можно реализовать так (взамен допускается использовать `while(true)`):

```
for (;;) Console.WriteLine ("прервите меня");
```

Циклы foceach

Оператор `foreach` обеспечивает проход по всем элементам в перечислимом объекте. Большинство типов в C# и .NET Framework, которые представляют набор или список элементов, являются перечислимыми. Ниже приведен пример перечисления символов в строке, от первого до последнего:

```
foreach (char c in "горы")

Console.WriteLine (c + " ");//г о р ы
```

Операторы перехода

К операторам перехода в C# относятся `break`, `continue`, `goto`, `return` и `throw`. Ключевое слово `throw` будет рассмотрено на лекции «Исключительные ситуации».

Оператор break

Оператор `break` завершает выполнение тела итерации или оператора `switch`:

```
int x = 0;
while (true)
{
  if (x++ > 5) break; // прекратить цикл
}
// После break выполнение продолжится здесь
...
```

Оператор continue

Оператор `continue` пропускает оставшиеся операторы в цикле и начинает следующую итера- цию. Показанный далее цикл *пропускает* четные числа:

```
for (int i = 0; i < 10; i++)
{
  if ((i % 2) == 0) continue;
    Console.Write (i + " ");      // 1 3 5 7 9
}
```

Оператор goto

Оператор `goto` переносит выполнение на метку (обозначаемую с помощью суффикса в виде двоеточия) внутри блока операторов. Следующий код выполняет итерацию по числам от 1 до 5, имитируя поведение цикла `for`:

```
int i = 1;
startLoop:
if (i <= 5)
{
  Console.Write (i + " "); // 1 2 3 4 5
  i++;
  goto startLoop;
}
```

Оператор return

Оператор `return` завершает метод и должен возвращать выражение с возвращаемым типом метода, если метод не является `void`:

```
static decimal AsPercentage (decimal d)
{
  decimal p = d * 100m;
  return p; // Возвратиться в вызывающий метод со значением
}
```

Оператор `return` может находиться в любом месте метода (кроме блока `finally`).