Курс «Основы программирования»

Федорук Елена Владимировна ст. преподаватель каф РК-6 МГТУ им.Н.Э.Баумана

Лекция №6

Управляющий оператор if

Простая форма оператора if

Оператор if — управляющий оператор языка Си, реализующий ветвление алгоритма.

if (выражение)

оператор1

Блок-схема простой формы оператора **`if`** представлена на рис. 1.

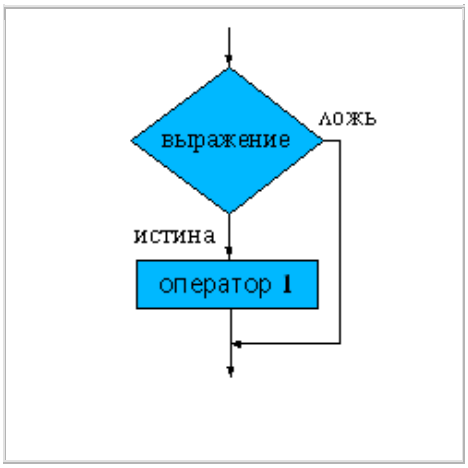

Рис. 1. Блок-схема простой формы оператора if

Если выражение в скобках истинно, то оператор выполняется, в противном случае оператор пропускается. Оператор может быть простым или составным (блоком). Скобки вокруг выражения обязательны.

Пример 1

```
if  (x > largest)
  largest = x;
```

Пример 2

```
if (val < min || val > max)
 {
  printf ("Значение %d выходит за пределы.\n", val);
  printf ("Пределы %d - %d.\n",min, max);
 }
```

Пример 3

```
if (num < 0)
  num = - num;
```

Полная форма оператора if

```
if (выражение)
    оператор1
else
    оператор2
```

Блок-схема полной формы оператора **if-else** представлена на рис. 2.

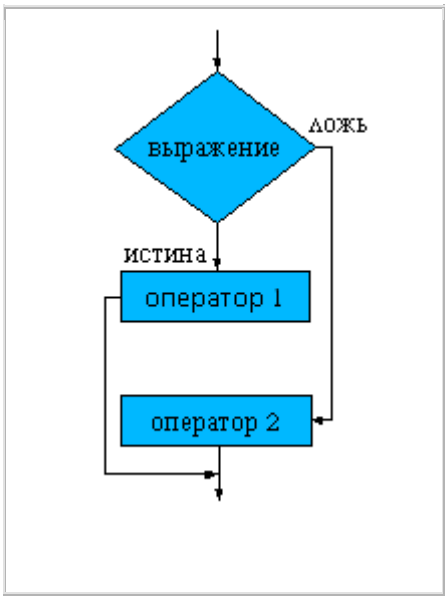

Рис. 2. Блок-схема полной формы оператора if

Если выражение в скобках истинно, то **оператор1** выполняется, а **оператор2** пропускается. Если выражение ложно, пропускается **оператор1**, выполняется **оператор2**.

Оператор1 и **оператор2** могут быть простыми или блоками. Скобки вокруг выражения, следующего за ключевым словом **if**, обязательны.

Пример 4

```
if (quantity < 1000)
  printf ("Количество недостаточное \n");
else
  printf ("Количество достаточное \n");
```

Пример 5

```
if ( x != 0)
    y = y/x;
else
   {
    printf ("Ошибка");
    printf ("Деление на нуль\n");
    y = 0;
   }
```

Обычными для многих программ являются серии вложенных операторов **`if-else`**. Такая серия может быть записана "ступеньками".

Пример 6

```
if (hour >= 8 && hour < 17)
   rate = rate * 1.02;
else
   if (hour >= 17 && hour < 23)
      rate = rate * 1.01;
```

Пример 7

```
if (age < 13)
   printf ("ребенок");
else
   if (age < 21)
     printf ("юноша");
   else
     printf ("мужчина");
```

При использовании вложенных операторов **`if-else`** может возникнуть проблема связывания **`else`**. **`else`** связывается с ближайшим **`if`**.

Пример 8

```
if (c > ' ')
  if ( c >= '0'&& c <= '9')
    digits += 1;
  else
    count += 1;
```

else связывается со вторым **if**. Оператор, следующий за первым ключевым словом **if**, является оператором **if-else**. Переменная **count** является счетчиком нецифровых символов.

Пример 9

```
    if (c > ' ')
    {
    if ( c >= '0' && c <= '9')
    digits += 1;
    }
    else
    count += 1;
```

else связывается с первым **if**. Оператор, следующий за первым ключевым словом **if**, является блоком. Переменная **count** является счетчиком управляющих символов.

Оператор цикла while

Оператор цикла while — управляющий оператор языка Си, реализующий выполнение цикла-пока в алгоритме.

Синтаксис оператора цикла **while** определяется следующим образом:

```
while (выражение)
    оператор1
```

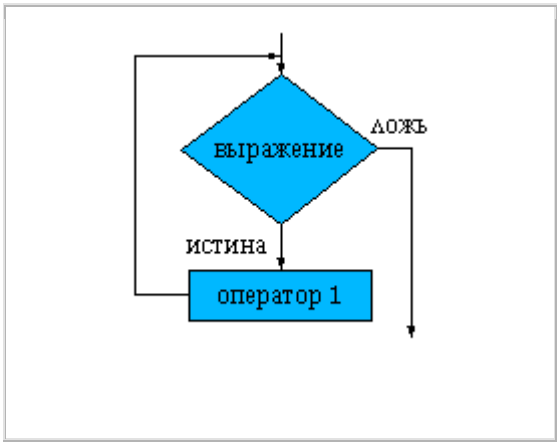

Рис. 1. Цикл while

На рис. 1 представлена блок-схема цикла **`while`**. Вычисляется выражение; если оно истинно (ненулевое), выполняется оператор. Выражение вычисляется снова и, если оно остается истинным, снова выполняется оператор, и т.д. Если выражение ложно, происходит выход из цикла. Как правило, значение выражения изменяется в результате выполнения некоторых действий в цикле. Оператор в цикле **`while`** выполняется 0 или более раз. Оператор может быть простым или составным (блоком). Частный случай простого оператора - пустой оператор.

Пример 1

```
count = 1;
while (count < 100)
   {
    оператор
    оператор
    count += 1;
   }
```

Пример 2

```
while ((c = getchar()) != '\n')
  {
   оператор
   оператор
   оператор
  }
```

Пример 3

```
/* Эта программа считает и строки */
/* и символы во входном потоке */
#include <stdio.h>
```

```
int main ()
{
 int lines = 0, chars = 0, c;
 /* читать до конца EOF */
 while ((c = getchar()) != EOF)
    {
     chars += 1;
     if (c == '\n')
        lines += 1;
     }
  printf ("%d lines \n", lines);
  printf ("%d characters \n", chars);
 }
```

Оператор цикла do-while

Оператор цикла do-while — управляющий оператор языка Си, реализующий выполнение цикла-до в алгоритме.

Синтаксис оператора цикла **do-while** определяется следующим образом:

```
do
     оператор
 while (выражение);
```

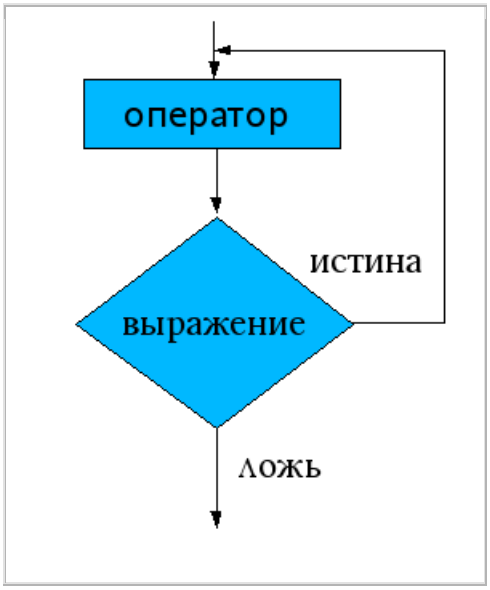

Рис. 1. Блок-схема цикла do-while

Цикл **do-while** похож на цикл **while** за исключением того, что значение выражения проверяется после выполнения оператора. Поэтому тело цикла выполняется как минимум один раз. Если значение выражения истинно, тело цикла выполняется повторно, выражение вновь вычисляется и т.д., до тех пор пока значение выражения не станет ложным. Оператор может быть как простым, так и составным.

Пример 1

```
/*Печатать меню, пока не введен символ 'q'*/
. . .
do {
   printf ("\t Основное меню \n");
   printf ("d - Ввод данных \n");
   printf ("r - Вывод отчета\n");
   printf ("q - Выход\n");
   scanf ("%c", &choice);
   if (choice == 'd')
     getdata();
   else
     if (choice == 'r')
       report();
     else
       if (choice != 'q')
         printf("Неверный ввод. Попробуйте еще раз.\n");
   while (getchar() != '\n')
    ; /* Удаление оставшихся символов */
  }
while (choice != 'q');
```

Оператор цикла for

Оператор цикла for — управляющий оператор языка Си, реализующий выполнение цикла в алгоритме.

Этот оператор очень удобен для представления счетных циклов, однако он может использоваться и для итерационных и поисковых циклов.

Синтаксис оператора цикла **for** определяется следующим образом:

```
for (выражение1; выражение2; выражение3)
     оператор
```

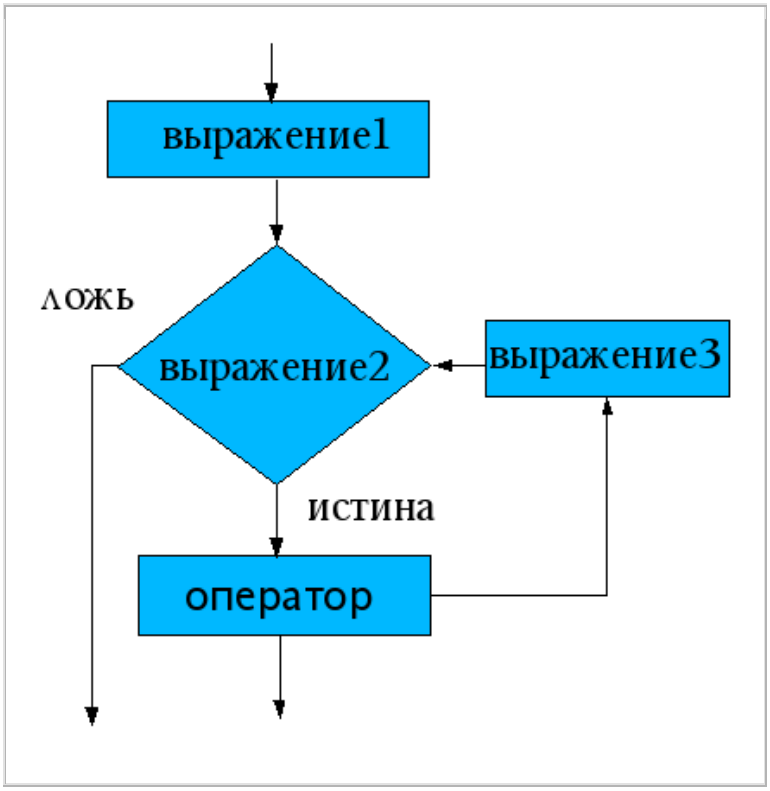

Рис. 1. Блок-схема цикла for

Цикл **for** используется для описания определенных действий для инициализации переменных цикла, циклического повторения тела цикла и изменения значений его переменных.

Выражение2 цикла **for** всегда используется в качестве условия его окончания. **Выражение1** часто применяется для инициализации переменных, а**выражение3** для их изменения. **Выражение1** вычисляется один раз при входе в цикл; **выражение2** используется многократно как условие окончания; **выражение3** вычисляется многократно после выполнения операторов тела цикла. Оператор **for** эквивалентен следующей последовательности операторов:

```
выражение1;
while (выражение2)
{
    оператор
    выражение3;
}
```

Преимущество синтаксиса оператора цикла **for** по сравнению с синтаксисом оператора цикла while заключается в том, что управляющие выражения цикла собраны вместе, что повышает удобочитаемость программы.

Пример 1

```
i = 0;
for (num = 100 ; num > 0; num -= 10)
    tens[i++] = num;
```

Любое из трех или все три выражения в операторе **for** могут отсутствовать, однако разделяющие их символы ; опускать нельзя. Если опущено **выражение2**, то оно считается по умолчанию истинным.

Циклы **for** могут быть вложенными. Их вложение часто используется для обработки многомерных массивов. Более того, любые из циклов языка Си могут быть вложены друг в друга, например цикл **while** внутри цикла **for**, цикл **for** внутри цикла **do-while** и т.д.

Пример 2

```
/* Сумма столбцов 0, 1, 2 и 3 размещается в столбце 4 */
cols =5;
for (r = 0; r<rows; r++)        /* для каждой строки */
   for (c = 0; c<cols-1; c++)  /* для каждого столбца */
      table[r][cols - 1] += table[r][c];
```

Пример 3

```
/*печатает 25 элементов массива */
for (i = 0; i<25; i++)
    printf ("%d ", ray[i]);
```

Пример 4

```
/* суммирует элементы массива */
sum = 0;
for (i = 0; i<max; )
   sum += quantity[i++];
```

Пример 5

```
/* бесконечный цикл */
for (;;)
  doinit();
```

Оператор switch

Оператор switch — управляющий оператор языка Си, реализующий алгоритмическую структуру «выбор».

Часто в программах необходимо произвести выбор одного из нескольких вариантов дальнейших действий. Это можно сделать, используя конструкцию, состоящую из цепочки операторов **if-else**, но во многих случаях оказывается более удобным использование оператора **switch**.

Синтаксис оператора **switch** определен следующим образом:

```
switch (выражение)
  {
   case константа1: операторы
   case константа2: операторы
    ...
   default:    операторы
   }
```

Выражение в скобках должно быть целого типа (можно использовать символьные константы, так как их тип целый). Выражение вычисляется и сравнивается с различными константами, записанными после ключевого слова **case**. Допускается использование констант целого или символьного типа или константное выражение указанных типов (например, **5**, **'a'**, **2048/512**). Если подходящая константа найдена, вычисления продолжаются с оператора, следующего за словом **case**, соответствующим этой константе. Затем выполняются все последующие операторы вплоть до появления оператора **break** или завершающей оператор **switch** скобки **}**. Метки **case** могут располагаться в любом порядке, но значения соответствующих констант должны быть различными.

switch, **case**, **default** и **break** — ключевые слова. Оператор **break** вызывает немедленный выход из оператора **switch**. Распространенная ошибка состоит в пропуске оператора **break**, когда необходим выход из оператора **switch**.

Метка **default** может отсутствовать. Если же она есть и среди указанных констант не найдено подходящей, управление передается операторам, следующим за меткой **default**. Если **default** отсутствует и среди указанных констант не найдено подходящей, оператор **switch** пропускается.

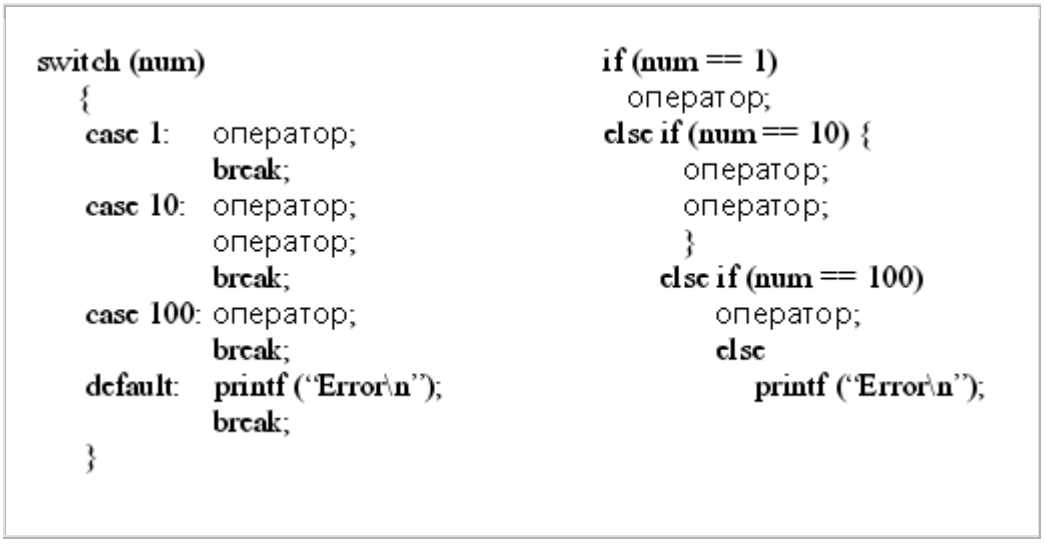

Рис. 1. Сравнение операторов `switch` и `if`

При использование оператора **`switch`** константы четко выделены, в противоположность операторам **`if-else`**, где они разбросаны по тексту. Разница в удобочитаемости возрастает с увеличением числа анализируемых случаев.

Если значение в операторе **`switch`** совпадает с одной из констант, вычисления продолжаются после соответствующей метки. Последующие операторы выполняются до тех пор, пока не встретится оператор **`break`**, не зависимо от того, предшествуют этим операторам какие-либо метки **`case`**. Говорят, что выполнение "проваливается", если не встретился оператор **`break`**. В прим. 1 демонстрируется особенность оператора **`switch`**.

Пример 1

```
input = getchar();
switch (input)
{
case 'a':
case 'A': add_record();
        break;
case 'd':
case 'D': find_record();
        delete_record();
        break;
case 'u':
case 'U': find_record();
        query_change();
        change_record();
```

```
        break;
default:  printf("Illegal choice\n");
        break;
}
```

Когда требуется использовать оператор **switch**, а когда конструкцию **if-else**? Нельзя применять оператор **switch**, когда выбор вариантов основан на вычислении значения переменной или выражения вещественного типа. Удобного способа применить оператор **switch** в случае, когда возможные значения переменной попадают в некоторый диапазон, также не существует. Проще написать, например, так:

```
if (integer < 1000 && integer > 2)
```

При замене этой строки оператором **switch** придется ввести в программу метки для всех целых чисел от 3 до 999. Тем не менее, если есть возможность использования оператора **switch**, ею надо воспользоваться, так как программа будет выполняться более эффективно.

Операторы break и continue

Операторы **break** и **continue** являются операторами передачи управления, не входящими в число базовых алгоритмических структур, на основании которых строятся структурные программы. Однако, они используются достаточно часто.

Управляющий *оператор break* осуществляет немедленный выход из циклов while, do-while и for. Одиночный оператор **break** не может использоваться для выхода из более чем одного уровня вложения циклов.

Оператор **break** может использоваться и для выхода из оператора switch.

Управляющий *оператор continue* осуществляет прекращение выполнения текущей итерации цикла, вызывая немедленное выполнение следующей итерации. Оператор **continue** используется только внутри циклов. Управление передается на управляющую часть (заголовок) цикла, с пропуском оставшихся операторов тела цикла.

Особенности выполнения операторов **break** и **continue** представлены на следующих рисунках.

Рис. 1. Цикл while

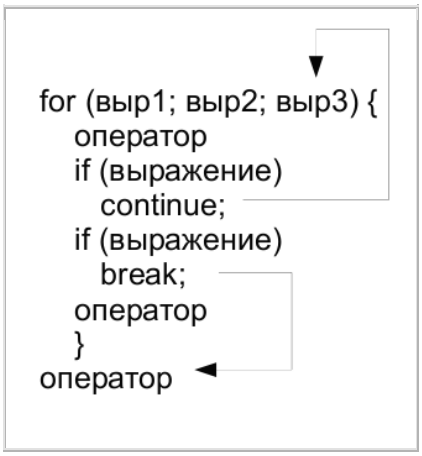

Рис. 2. Цикл for

Рис. 3. Цикл do - while

Примечание 1

Стрелки показывают передачу управления.

Оператор goto

Управляющий *оператор goto* вызывает безусловный переход управления на выполнение оператора, перед которым указана соответствующая метка.

```
goto метка;
```

Метка — это идентификатор, завершаемый двоеточием, который помечает оператор.

Метка должна предшествовать оператору, в вырожденном случае — пустому оператору. Можно переходить только на метки, указанные внутри текущей функции, нельзя перейти на метку, указанную в другой функции.

Часто говорят, что структурное программирование - это программирование без **goto**. Строго говоря, в этом операторе нет никакой необходимости. Оператор **goto** можно использовать только для выхода из глубоко вложенных циклов. В противном случае использование **goto** делает структуру программу запутанной и трудной для отладки. На использование оператора **goto** провоцирует плохо продуманный алгоритм.

Примечание 1

В функции не должно быть больше одного оператора **goto**.

Пример 1

```
main ()
{
. . .
/* не рекомендуется */
start: оператор
оператор
if (выр)
goto start;
оператор
оператор
...
}
```

прим. 1 демонстрирует плохой стиль программирования. Такое использование оператора **goto** делает программу плохо структурированной, а, следовательно, плохо отлаживаемой, ненадежной и неэффективной.

Пример 2

```
main ()
{
...
/* рекомендуется */
while (выр)
  {
   while (выр)
     {
      оператор
      while (выр)
        {
         while (выр)
           {
            оператор
            if (выр)
               goto end;
            оператор
            }
          оператор
          }
     }
  }
end:;
}
```

прим. 2 демонстрирует ситуацию, когда использование оператора **goto** оправдано.