Операторы цикла

Циклические операции являются часто употребляемыми операциями. Они служат для многократного выполнения последовательности операторов до тех пор, пока не выполниться некоторое условие. Условие может быть установлено заранее или меняться при выполнении тела цикла.

Цикл `while`

Общая форма цикла `while` имеет вид:

```
while (условие) {
    блок операторов;
}
```

Тело цикла может быть пустым, состоять из единственного оператора или блока операторов. Условие цикла может быть любым допустимым в языке Си выражением. Цикл `while` использует предусловие. Оно считается истинным, если значение условного выражения не равно нулю. Если условие выполняется, то и цикл выполняется. Если условие принимает значение `ЛОЖЬ`, то программа выходит из цикла, и выполняется оператор, следующий за циклом.

Пример: Вывести на экран числа от `0` до `10`.

```
int number = 0;
while (number <= 10) {
    printf("%3d", number); // вывод на экран
    number++; // переход к следующему числу
}
```

Вводим и предварительно обнуляем переменную `number`, которая будет отвечать за подсчет уже напечатанных чисел. Прежде чем выполнить тело цикла, проверяется условие, что значение переменной не превышает `10` `number <= 10`. Если это так, то выполняем тело цикла, которое состоит из двух операторов: функции вывода на экран значения переменной и увеличения значения этой переменной на единицу. Как только значение переменной превысит единицу, цикл перестанет выполняться.

Пример: Вычислить через сколько минут переполнится дырявая бочка с водой (`250` литров), если каждую минуту (кроме шестой) в нее вливается по `3` литра воды, а каждую шестую минуту единовременно вытекает `5` литров.

```
#define MAX_VOLUME 250
int volume = 0, time = 0;
```

```
while (volume < MAX_VOLUME) {
    time++;
    if (time % 6 == 0) {
        volume -= 5;
    }
    else {
        volume += 3;
    }
}
printf("Бочка переполнится через %d минут\n", time);
```

Введем макрос для максимального объема бочки и две переменные для текущего объема воды в бочке и времени. Предварительно обнулим переменные. Далее в цикле будем прибавлять минуты и прибавлять или отнимать литры в зависимости от того, какая идет минута. После выполнения цикла на экран выводится количество минут до переполнения бочки.

Цикл do-while

Общая форма цикла **do-while** имеет вид:

```
do {
    блок операторов;
} while();
```

В отличие от цикла **while**, в котором условие проверяется до выполнения, цикл **do-while** сначала выполняет тело цикла, а затем проверяет условие. Таким образом, цикл **do-while** выполняется по крайней мере один раз, а цикл **while** может не выполниться ни разу.

Пример: На какой раз выпадет число ноль при бросании случайного числа в пределах [0; 1].

```
long int count = 0;
double number = 0;
do {
    number = rand() / (double)RAND_MAX;
    count++;
} while (number != 0);
printf("Ноль выпадет на %li раз\n", count);
```

Объявляем одну целую перемененную для подсчета количества бросков и дробную переменную для записи случайного дробного числа от нуля до единицы. В теле цикла **do-while** будет генерироваться случайное число. После генерации числа счетчик количества генераций увеличивается на единицу и проверяется условие цикла. Если условие выполняется, то цикл повторяется.

Цикл **do-while** потенциально может приводить к ошибкам, так как тело цикла в обязательном порядке выполнится хотя бы один раз, даже если условие выполнения цикла окажется неверным. Поэтому следует использовать этот вариант циклического оператора с осторожностью и везде, где это возможно, заменять его на цикл **while**. В практике программирования можно найти совсем немного примеров необходимого использования цикла **do-while**.

Использование цикла **do-while** может быть эффективно в том случае, когда в условие входит величина или выражение, которое требует предварительного вычисления, а также повторного вычисления в процессе выполнения цикла. Это можно проиллюстрировать примером:

Пример: Задать с клавиатуры отрицательное значение для переменной.

- цикл **while**:

```
int number = 0;
printf("Enter a negative number\n");
scanf("%d", &number);
while (number >= 0) {
    printf("Enter a negative number\n");
    scanf("%d", &number);
}
printf ("The negative number is %d\n", number);
```

- цикл **do-while**:

```
int number = 0;
do {
    printf("Enter a negative number\n");
    scanf("%d", &number);
} while (number >= 0);
printf("The negative number is %d\n", number);
```

Здесь нужно заранее ввести значение целой переменной с клавиатуры для использования ее внутри условия выполнения цикла. То есть до цикла **while** требуется строка **scanf**`("%d", &number);` для ввода начального значения переменной. Иначе мы не сможем вычислить условное выражение, так как не будем

знать, чему равна переменная number. В случае цикла **do-while** мы можем избежать дополнительной строчки кода с предварительным вводом значения переменной, так как это произойдет прямо в цикле.

Цикл for

Общая форма цикла **for** имеет вид:

```
for (инициализация; условие; приращение) {
    блок операторов;
}
```

Цикл **for** может иметь большое количество вариаций. В наиболее общем виде принцип его работы следующий. Инициализация — это присваивание начального значения переменной, которая называется параметром цикла. Условие представляет собой условное выражение, определяющее, следует ли выполнять в очередной раз тело цикла. Оператор приращения осуществляет изменение параметра цикла при каждой итерации. Эти три оператора обязательно разделяются точкой с запятой. Цикл **for** выполняется, если условие принимает значение ИСТИНА. Как только условие примет значение ЛОЖЬ, то программа выходит из цикла и выполняется оператор, следующий за телом цикла **for**.

Пример: Вывести на экран все целые числа от 0 до 100.

```
int x;
for (x = 0; x <= 100; x++) {
    printf("%4d", x);
}
```

В этом примере параметр цикла x инициализирован числом 0, а затем до каждой итерации сравнивается с числом 100. Пока переменная x меньше или равна 100, вызывается функция **printf**() и цикл повторяется. После каждого выполнения тела цикла переменная x увеличивается на единицу и снова проверяется условие выполнения цикла.

Пример: Вычислить и вывести на экран координаты точек функции $y=7x-5x^2$ при значениях x [-15; 30] с шагом 3.

```
int x = 0, y = 0;
for (x = -15; x <= 30; x += 3) {
    y = 7 * x - 5 * x * x;
    printf("x = %3d\ty = %5d\n", x, y);
}
```

Параметр цикла изменяется от `-15` до `30`. Тело цикла состоит из вычисления значения переменной `y` и оператора вывода значений переменных `x` и `y` на экран в столбик.

Множество параметров цикла `for`

В качестве параметров для цикла `for` может служить несколько переменных. В этом случае они могут указываться через запятую.

Пример: Сколько потребуется итераций для того, чтобы сумма трех переменных стала больше либо равной `100`. Начальные значения переменных: `1, 0, 0`. Шаг изменения переменных: `+1, -1, +7`.

```
int x = 0, y = 0, z = 0, iteration = 0;
for (x = 1, y = 0; x + y + z < 100; x++, y--, z += 7) {
    iteration++;
    printf("x = %4d\ty = %4d\tz = %4d\n", x, y, z);
}
printf("\nПрошло %d итераций\n", iteration);
```

В этом примере можно заметить, что приращение можно выполнять не только с теми переменными, которым задаются в цикле начальные значения. Приращение можно осуществлять с любым количеством любых переменных. Инициализацию также можно проводить с несколькими переменными. Условие выполнения цикла должно быть одно, но оно может быть комбинированным, то есть содержать в себе несколько условных выражений, соединенных логическими операторами (например, `(x > 5) || (y < 7)`).

Кроме того, что в секциях цикла `for` может присутствовать несколько элементов, еще одной его особенностью является возможность полного отсутствия одной, двух или даже всех трех секций.

Вложенные циклы

Тело цикла может содержать в себе различные операторы. Это могут быть операторы присваивания, ветвления, циклические операторы и т.д. Примеры циклов, содержащих условные операторы, были представлены выше. Рассмотрим примеры циклов, которые содержат в себе другие циклы.

Пример: Вывести на экран таблицу умножения.

```
int i, j;
for (i = 1; i <= 9; i++) {
    for (j = 1; j <= 9; j++) {
        printf("%3d", i * j);
    }
    printf("\n");
}
```

На экране получим:

```
  1  2  3  4  5  6  7  8  9
  2  4  6  8 10 12 14 16 18
  3  6  9 12 15 18 21 24 27
  4  8 12 16 20 24 28 32 36
  5 10 15 20 25 30 35 40 45
  6 12 18 24 30 36 42 48 54
  7 14 21 28 35 42 49 56 63
  8 16 24 32 40 48 56 64 72
  9 18 27 36 45 54 63 72 81
```

Внешний цикл отвечает за номер выводимой строки, а внутренний цикл за номер выводимого результата умножения в текущей строке. Номера строк также являются числами, для которых мы составляем таблицу умножения, поэтому они участвуют в вычислении произведения перед его выводом на экран.

Добавим в нашу таблицу немного красоты.

```
int i, j;
printf("%5c", ' ');
for (i = 1; i <= 9; i++) {
    printf("%3d", i);
}
printf("\n%5c", ' ');
for (i = 1; i <= 9; i++) {
    printf("---");
}
printf("\n");
for (i = 1; i <= 9; i++) {
    printf("%3d |", i);
```

```
    for (j = 1; j <= 9; j++) {
        printf("%3d", i * j);
    }
    printf("\n");
}
```

В результате получаем

```
      1  2  3  4  5  6  7  8  9
    -------------------------
1 | 1  2  3  4  5  6  7  8  9
2 | 2  4  6  8 10 12 14 16 18
3 | 3  6  9 12 15 18 21 24 27
4 | 4  8 12 16 20 24 28 32 36
5 | 5 10 15 20 25 30 35 40 45
6 | 6 12 18 24 30 36 42 48 54
7 | 7 14 21 28 35 42 49 56 63
8 | 8 16 24 32 40 48 56 64 72
9 | 9 18 27 36 45 54 63 72 81
```

Здесь мы добавили несколько дополнительных циклов для вывода ряда чисел, которые участвуют в умножении и вывода пунктирного разделения. Первые две строки пришлось сдвинуть вправо за счет пробелов. Кроме того, внутрь предыдущего варианта вложенных циклов был добавлен вывод числа для умножения и вертикальной черты.

Пример: Сгенерировать и вывести на экран случайное количество строчек (`1..25`) из случайного количества (`1..50`) случайных чисел (`-50..50`).

```
int num_str = 1 + rand() % 25, num_numb = 0;
int i, j;
for (i = 1; i <= num_str; i++) {
    num_numb = 1 + rand() % 50;
    for (j = 1; j <= num_numb; j++) {
        printf("%4d", -50 + rand() % 101);
    }
    printf("\n");
}
```

Объявляем и инициализируем переменные. Две переменные — для использования в качестве параметров цикла (для каждого цикла нужно использовать свой параметр цикла). И еще две переменные: одна будет отвечать за общее количество строк, другая — за количество случайных чисел в текущей строке. Внешний цикл `for (i = 1; i <= num_str; i++)` отвечает за строки, то есть параметр цикла `i` инициализируется со значением `1`, что, в свою очередь, символизирует первую строку. После каждой итерации к параметру будет прибавляться единица до тех пор, пока номер текущей строки не превысит значение переменной `num_str` (`i <= num_str`), которая содержит значение сгенерированного количества строк. Тело внешнего цикла состоит из трех операторов:

- первый оператор обеспечивает генерацию случайного количества чисел, которые будут сгенерированы для данной строки, и запись этого значения в переменную `num_numb`.
- второй оператор — это внутренний цикл, который отвечает за генерацию и вывод нужного количества случайных чисел на экран. Параметр цикла `j` отвечает за номер выводимого случайного числа. Он изменяется от `1` до величины `num_numb` с шагом `1`. Тело внутреннего цикла состоит из функции `printf()`, которая обеспечивает вывод на экран случайного числа, которое генерируется непосредственно внутри функции.
- третий оператор — это функция `printf()`, которая обеспечивает переход на новую строку.

Таким образом, получается три этапа работы тела внешнего цикла:

1. генерация количества чисел в текущей строке
2. генерация и вывод чисел в строку
3. переход на новую строку

Бесконечные циклы

Для создания бесконечных циклов можно использовать любой из трех операторов цикла. Многие программисты чаще всего выбирают цикл `while`, или цикл `for`.

Самый простой способ создания бесконечного цикла `for` — это оставить пустыми все три секции.

```
for ( ; ; ) {
    printf("Это бесконечный цикл\n");
}
```

Если условие в цикле отсутствует, то предполагается, что оно — ИСТИНА. В оператор можно добавить инициализацию и/или приращение каких-либо параметров цикла, хотя обычно все же используют пустую конструкцию.

Для контроля над бесконечностью таких циклов существует оператор прерывания **break**. В случае, если программа встретит слово **break** в обычном или в бесконечном цикле, то она сразу же выйдет из него и продолжит работу, выполняя операторы следующие за циклом. Неконтролируемые бесконечные циклы очень опасны, они могут привести к утечкам памяти и различного рода сбоям. Хотя и обычные циклы могут стать бесконечными в результате ошибки или опечатки программиста. Поэтому следует быть очень внимательным при написании циклов любого рода.

Пример. Найти первое положительное число, прибавляя к числу -389 по 5.

```
int number = -389;
for ( ; ; ) {
    if (number > 0) {
        printf("First \"+\" number is %d\n", number);
        break;
    }
    number += 5;
}
```

Для циклов **while** и **do-while** также можно образовать бесконечные циклы с выходом из них с помощью оператора **break**. Чаще всего это реализуют с помощью задания «всегда истинного» условия. Например, при указании в условии целого числа, отличного от нуля (чаще всего указывают единицу).

Пример: Осуществить ввод символов с клавиатуры до тех пор пока не будет введен знак «!».

```
char letter;
for ( ; ; ) {
    scanf("%c", &letter);
    if (letter == '!') {
       break;
    }
}
```

Пример: Вывод на экран некоторого множества случайных чисел.

```
int count = 1 + rand() % 30;
printf("%d случайных чисел будет\n", count);
```

```
while (1) {
    printf("%5d", - 500 + rand() % 1000);
    count--;
    if (count == 0) {
        break;
 }
}
```

Здесь для выхода из бесконечных циклов используется оператор перехода **break**. Этот оператор, как и оператор **continue,** используется для управления циклическими операциями.

Оператор break

Этот оператор применяется в двух случаях:

1. для немедленного выхода из любых циклических операций
2. для прекращения работы оператора условия **switch**

Оператор **break** прерывает выполнение условного оператора **switch** или любого циклического оператора и передает управление следующему за ними оператору.

Стоит заметить, что если у вас присутствуют вложенные циклы, то **break** прервет работу только одного из них. Для полного выхода из обоих циклов следует использовать два оператора **break**.

Пример. Выполнение цикла 100 раз.

```
for (t = 1; t <= 100; t++) {
    count = 1;
    for ( ; ; ) {
        printf("%3d", count);
        count++;
        if (count == 10) {
           break;
        }
    }
}
```

Если оператор **break** присутствует внутри оператора **switch**, который вложен в какие-либо циклы, то он относится только к **switch**, и выход из цикла не происходит. Таким образом, оператор **break** влияет только на тот цикл или условный оператор **switch**, к которому он непосредственно относится.

Оператор `continue`

Оператор **`continue`** прерывает текущую операцию цикла и осуществляет переход к следующей итерации цикла. При этом пропускаются все операторы тела цикла, которые стоят после **`continue`**.

Пример. Вывести на экран символы, которые следуют за символами, введенными с клавиатуры. Например, если с клавиатуры ввели цифру «3», то на экран выводится цифра «4», а если ввели букву «m», то выводится буква «n». Ввод производить по одному символу. Если встретится точка, то ввод символов следует прекратить.

```
char ch;
int finish = 0;
while (finish == 0) {
    scanf("%c", &ch);
    if (ch == '.') {
        finish = 1;
        continue;
    }
    printf("%c ", ch + 1);
}
```

Объявляем переменную, которая будет содержать введенный с клавиатуры символ, и переменную для определения окончания программы, которая изначально равна нулю (`0` – программа не закончила работу, `1` – программа закончила работу). Перед выполнением очередной итерации цикла будем проверять, закончила ли работу программа. В теле цикла происходит ввод символа с клавиатуры и его запись в переменную `ch`. Затем идет проверка: если введена точка, то переменная `finish` становится равной единице, означая, что программа выполнила свою задачу и должна закончиться. Следующий оператор – **`continue`** осуществляет немедленный переход в начало цикла к этапу проверки условия его выполнения. Теперь это условие не выполняется, и цикл завершается. В случае, когда введенный символ не точка, происходит печать следующего по алфавиту символа.

Оператор **`continue`** используется довольно редко, но он бывает необходим при реализации сложных алгоритмов.

Оператор `goto`

Оператор **`goto`** — это один из операторов перехода. В отличие от остальных он может находиться практически в любом месте программы, а не только в циклах как **`break`** и **`continue`**. И переходить с его помощью можно также практически в

любое место в пределах текущей функции. Но, несмотря на эти, казалось бы, сильные преимущества, программисты стараются избегать употребление этого оператора именно из-за его “неограниченности”.

Так как переход может осуществляться куда угодно, то не всегда понятно, куда же именно он осуществится. При большом количестве строк кода и множестве переходов **goto** придется сильно постараться, прежде чем понять, как работает программа. В результате чрезмерного использования операторов **goto** программы чрезвычайно плохо читаются. То есть оператор **goto** весьма непопулярен, более того, считается, что в программировании не существует ситуаций, в которых нельзя обойтись без оператора **goto**. Но в некоторых случаях его применение все же уместно. Иногда, при умелом использовании, этот оператор может оказаться весьма полезным, например, если нужно покинуть глубоко вложенные циклы.

Для оператора **goto** всегда необходима метка. Метка — это идентификатор с последующим двоеточием. Метка должна находиться в той же функции, что и оператор **goto**, переход в другую функцию невозможен.

Общая форма оператора **goto** следующая:

```
goto метка;
/* ---------
---------- */
метка:
/* ---------
---------- */
```

или

```
/* ---------
---------- */
метка:
/* ---------
---------- */
goto метка;
/* ---------
---------- */
```

Метка может находиться как до, так и после оператора **goto**. Например, используя оператор **goto**, можно выполнить цикл от 1 до 100:

```
x = 1;
metka:
```

```
x++;
if (x <= 100) {
    goto metka;
}
```

В этом пособии оператор **goto** дается только для ознакомления. В лабораторной практике он использоваться не будет.