

1.3. Циклы. Ввод с клавиатуры.

Wednesday, March 18, 2020 10:55

В данном уроке мы разберем, как работают циклы в языке C. Будут рассмотрены операторы:

- ♦ for
- ♦ while
- ♦ do
- ♦ break
- ♦ continue

Напишем программу, вычисляющую сумму n последовательных чисел (треугольное число), используя оператор for:

```
1 #include <stdio.h>
2
3 int main (void)
4 {
5     int n, triangularNumber = 0;
6
7     for (n = 1; n <= 200; n = n + 1)
8         triangularNumber = triangularNumber + n;
9
10    printf("The 200th triangular number is %i.\n", triangularNumber);
11
12    return 0;
13 }
```

Программа выдаст:

The 200th triangular number is 20100.

Синтаксис оператора for выглядит следующим образом:

```
1 for (init_expression; loop_condition; loop_expression)
2 {
3     program statement (or statements)
4 }
```

Возможно, некоторые обратили внимание, что в строке 8 нашей программы тело цикла не взято в фигурные скобки { }. Так можно делать, если оно состоит из одной строки.

Условий цикла может быть несколько[2], например:

```
for (i=1, j=1; i<10 && j<10; i++, j++)
```

или

```
for (i=1, j=1; i<30 || j<50; i++, j++)
```

Условием окончания (loop_expression) цикла обычно является неравенство (или более сложное логическое выражение), которое возвращает логическое TRUE или FALSE. Как только условие окончания возвращает FALSE, цикл завершается. Рассмотрим операторы неравенств в C:

Operator	Meaning	Example
<code>==</code>	Equal to	<code>count == 10</code>
<code>!=</code>	Not equal to	<code>flag != DONE</code>
<code><</code>	Less than	<code>a < b</code>
<code><=</code>	Less than or equal to	<code>low <= high</code>
<code>></code>	Greater than	<code>pointer > endOfList</code>
<code>>=</code>	Greater than or equal to	<code>j >= 0</code>

Приоритет операторов неравенств ниже, чем у остальных арифметических операторов. Так, выражение

`a < b + c`

будет обработано как

`a < (b + c)`

Что в принципе очевидно, но упомянуть стоит.

Также стоит не путать оператор сравнения `==` с оператором присваивания `=`

И нужно помнить, что нельзя ставить точку с запятой `;` после описания цикла `for ()`, иначе это сразу завершит цикл и тело цикла не будет выполнено.

Подытожим работу цикла `for`:

1. Рассчитывается начальное значение (`init_expression`). Здесь присваивается начальное значение (а иногда и объявляется) переменной счетчика, – обычно 0 или 1.
2. Проверяется условие цикла (`loop_condition`). Если оно не удовлетворяется (`FALSE`), цикл завершается. Если удовлетворяется, то цикл переходит к выполнению своего тела.
3. Выполняется тело цикла.
4. Рассчитывается выражение цикла (`loop_expression`, напр. `i++`).
5. Возврат к пункту 2.

Давайте теперь упакуем промежуточные результаты расчета нашей суммы чисел в красивую таблицу. Но для компактности будем считать 10 чисел, а не 200. И приукрасим вывод для лучшей читабельности:

```

1 #include <stdio.h>
2
3 int main (void)
4 {
5     int n, triangularNumber = 0;
6
7     printf ("TABLE OF TRIANGULAR NUMBERS\n\n");
8     printf (" n      Sum from 1 to n\n");
9     printf ("---  ----- \n");
10
11     for (n = 1; n <= 10; n++) {
12         triangularNumber += n;
13         printf(" %i          %i\n", n, triangularNumber);
14     }
15
16     return 0;
17 }
```

Программа выдаст:

TABLE OF TRIANGULAR NUMBERS

n	Sum from 1 to n
1	1
2	3
3	6
4	10
5	15
6	21
7	28
8	36
9	45
10	55

Чет кривовато вышло. Давайте подровняем таблицу. Чтобы двузначные числа слева нам не смещали столбец справа, укажем длину поля (**field width specification**). Оператор `%2i` укажет компилятору, что мы не просто хотим отобразить переменную, но также то, что она может занимать до двух знакомест. Любые числа с одним знакоместом будут отображены с ведущим пробелом (*leading space*). Это называется выравниванием по левому краю. Если нужно выровнять по левому краю, нужно дописать минус: `%-2i`.

Таким образом, `%2i` нам гарантирует, что будут заняты два знакоместа и двузначные числа не будут смещать второй столбец. Исправим строку вывода таблицы:

```
printf(" %2i          %i\n", n, triangularNumber);
```

И получим:

TABLE OF TRIANGULAR NUMBERS

n	Sum from 1 to n
1	1
2	3
3	6
4	10
5	15
6	21
7	28
8	36
9	45
10	55

Ввод с клавиатуры

Наша первая программа просто рассчитывает 200е треугольное число и больше ничего. Но, допустим, мы хотим рассчитать 50е или 100е число. Не переписывать же программу каждый раз. Напишем программу, которая будет считывать с клавиатуры число, до которого мы хотим досчитать. Для считывания с клавиатуры есть функция `scanf()`, которая довольно похожа общей идеей на `printf()`:

```
1 #include <stdio.h>
2
3 int main (void)
4 {
5     int n, number, triangularNumber = 0;
6
7     printf (" What triangular number do you want? ");
8     scanf ("%i", &number);
9     printf ("\n");
10
11     for (n = 1; n <= number; n++)
12         triangularNumber += n;
```

```

13
14     printf("    Triangular number %i is %i\n", number, triangularNumber);
15
16     return 0;
17 }

```

Программа выдаст:

```
What triangular number do you want? 100
```

```
Triangular number 100 is 5050
```

То есть цикл выполнился именно столько раз, сколько мы задали с клавиатуры. Как и в `printf()`, для считывания целочисленной переменной в функции `scanf()` мы использовали оператор `%i`. Второй оператор указывает, в какую переменную записывать считанное значение. Знак `&` обязателен перед названием переменной, иначе функция будет работать неправильно. Этот момент мы разберем позже.

Вложенные циклы `for` (Nested `for` Loops)

Допустим, мы хотим посчитать не одно треугольное число, а несколько чисел. Дополним нашу программу вложенным циклом, который 5 раз запросит число и выведет соответствующее ему треугольное число:

```

1  #include <stdio.h>
2
3  int main (void)
4  {
5      int n, number, triangularNumber, counter;
6
7      for (counter = 1; counter <=5; counter++){
8          printf ("    What triangular number do you want? ");
9          scanf ("%i", &number);
10
11          triangularNumber = 0;
12
13          for (n = 1; n <= number; n++)
14              triangularNumber += n;
15
16          printf("    Triangular number %i is %i\n\n", number,
17              triangularNumber);
18      }
19
20     return 0;
21 }

```

Программа выдаст:

```
What triangular number do you want? 12
```

```
Triangular number 12 is 78
```

```
What triangular number do you want? 25
```

```
Triangular number 25 is 325
```

```
What triangular number do you want? 50
```

```
Triangular number 50 is 1275
```

```
What triangular number do you want? 75
```

```
Triangular number 75 is 2850
```

```
What triangular number do you want? 83
```

```
Triangular number 83 is 3486
```

Обратите внимание, что обнуления переменной `triangularNumber` при объявлении уже не достаточно, ее нужно обнулять каждый раз перед началом расчетов, в цикле, иначе к ней будут прибавляться результаты предыдущих вычислений.

Разные варианты цикла `for`

Для цикла `for` допускаются различные вариации синтаксиса. Рассмотрим их.

Множественные выражения (multiple expressions)

В любое из трех полей описания цикла `for` можно подставлять любые выражения. Например, цикл

```
for (i = 0, j = 100; i < 10; i++, j = j - 10)
```

имеет две переменных-счетчика, — `i` и `j`. Для их корректного описания, их нужно разделять запятой, как и остальные выражения. Каждую итерацию цикла `i` увеличивается на 1, а `j` уменьшается на 10.

Опускание полей описания цикла (omitting fields)

Иногда нам не нужно задавать какие-то поля в цикле. Так можно делать, но нужно не забывать ставить разделитель — точку с запятой `;` после незаполненных полей.

Например, мы можем обойтись без задания начального условия (`init_expression`):

```
for ( ; j != 100; ++j)
```

Так можно делать, если у этой переменной уже есть какое-то значение, присвоенное ранее.

Еще можно обойтись без условия окончания цикла (`looping_condition`). Так можно делать, если мы используем другие операторы для выхода из цикла, такие как `return`, `break`, или `goto`. Иначе получится бесконечный цикл.

Объявление переменных (declaring variables)

Переменные можно объявлять не в основной программе, а прямо в описании цикла. Синтаксис такой же, как обычно. Например, инициализация переменной счетчика будет выглядеть так:

```
for (int counter = 1; counter <= 5; ++counter)
```

Переменная `counter` относится только к телу цикла и не будет доступна вне цикла. Приведем еще один пример объявления переменных, в этот раз и счетчика, и внутренней переменной для тела цикла:

```
1 for ( int n = 1, triangularNumber = 0; n <= 200; ++n )
2     triangularNumber += n;
```

Обратите внимание, что чтобы работало объявление переменных в цикле, нужно включить режим C99 в настройках компилятора, иначе Code:Blocks не скомпилирует программу.

Цикл `while` (The `while` Statement)

Оператор `while` дополняет возможности создания циклов. Синтаксис выглядит следующим образом:

```
1 while (looping_condition) {
2     тело цикла
3 }
```

Условие цикла (`looping_condition`) проверяется каждый раз перед выполнением тела цикла. Если оно выполняется, цикл "проворачивается" один раз, до следующей проверки условия. Как только проверка условия выдает `FALSE`, программа выходит из цикла.

Напишем простую программу, выводящую текущее значение счетчика цикла.

```
1 // Program to introduce the while statement
```

```

2
3 #include <stdio.h>
4
5 int main (void)
6 {
7     int i = 1;
8
9     while (i <= 5) {
10         printf ("%i\n", i);
11         i++;
12     }
13
14     return 0;
15 }

```

Программа выдаст:

```

1
2
3
4
5

```

Как видим, у нас в отличие от цикла `for`, выражение цикла (`loop expression`) находится в теле цикла. Так что ничто нам не мешает преобразовать один тип цикла в другой.

Раз уж мы выучили два разных типа циклов, возможно возникнет вопрос, когда какой цикл лучше использовать. В целом, цикл `for` удобнее использовать когда цикл нужно выполнить конкретное количество раз и когда все три поля описания цикла используют одни и те же переменные.

Для закрепления знаний напишем программу, вычисляющую наибольший общий делитель.

```

1 /* Finding a greatest common divisor
2    of two integer numbers          */
3
4 #include <stdio.h>
5
6 int main (void)
7 {
8     unsigned int num1, num2, temp;
9
10    printf ("Please type in two nonnegative numbers.\n");
11    scanf ("%u%u", &num1, &num2);
12
13    while (num2 != 0) {
14        temp = num1 % num2;
15        num1 = num2;
16        num2 = temp;
17    }
18
19    printf ("Their greatest common divisor is %i.\n", num1);
20
21    return 0;
22 }

```

Здесь операторы `%u%u` в функции `scanf()` задают ввод два раза подряд. Сначала считывается `num1`, затем `num2`.

Программа выдаст:

```

Please type in two nonnegative numbers.
1026
405
Their greatest common divisor is 27.

```

Напишем еще одну программу; которая переворачивает введенное с клавиатуры число.

```
1 // Reversing digits in a number
2
3 #include <stdio.h>
4
5 int main (void)
6 {
7     long long int number, right_digit;
8
9     printf ("Enter your number.\n");
10    scanf ("%lli", &number);
11
12    while (number != 0) {
13        right_digit = number % 10;
14        printf ("%lli", right_digit);
15        number = number / 10;
16    }
17
18    printf ("\n");
19
20    return 0;
21 }
```

Здесь используется особенность деления целочисленных переменных: деление на 10 фактически будет отсекать цифры из числа. Ну а `number % 10` будет равняться как раз последней цифре в числе. Таким образом, мы будем выводить по очереди последнюю цифру в числе, а затем удалять ее из числа. И так пока не удалим все числа.

Программа выдаст:

```
Enter your number.
13579
97531
```

Оператор do (The do Statement)

В двух типах циклов, которые мы рассмотрели ранее, условие цикла проверялось в начале цикла. Таким образом, тело цикла вообще не выполняется, если условие цикла не выполнено. Иногда возникает необходимость проверять условие цикла в его конце, после того как тело цикла выполнится хотя бы один раз. Для этого в языке C существует оператор `do`, который используется в связке с циклом `while`:

```
1 do
2     тело цикла
3 while ( looping_condition );
```

Такая конструкция выполняется один раз до проверки условия цикла а, затем крутится в цикле, пока выполняется условие цикла. Если оно не выполнилось изначально, программа просто продолжает работать, выполнив тело цикла один раз.

Не будем далеко ходить и поставим в пример нашу предыдущую программу, где мы переворачивали числа. Если мы введем 0, программа ни разу не выполнит тело цикла и ничего не выведет. Хотя, формально, она должна выводить 0. Если мы используем оператор `do`, то у нас будет гарантия того, что тело цикла будет выполнено хотя бы один раз и программа всегда будет что-то выводить. Переделаем программу, не забыв поставить `;` после `while()`, как это сделал я.

```
1 // Reversing digits in a number
2
3 #include <stdio.h>
4
```

```

5 int main (void)
6 {
7     long long int number, right_digit;
8
9     printf ("Enter your number.\n");
10    scanf ("%lli", &number);
11
12    do {
13        right_digit = number % 10;
14        printf ("%lli", right_digit);
15        number = number / 10;
16    }
17    while (number != 0);
18
19    printf ("\n");
20
21    return 0;
22 }

```

Теперь программа работает корректно.

Оператор `break`

Иногда возникает необходимость выйти из цикла, как только будет выполнено какое-то условие, не дожидаясь конца выполнения тела цикла. Например, если была обнаружена какая-то ошибка в данных. После выхода из цикла программа продолжает выполнять код, который следует сразу после тела цикла. Если выйти из вложенного цикла, то продолжится выполнение того цикла, в который прерванный цикл был вложен. Оператор работает с любыми циклами: `for`, `while` и `do`. Синтаксис прост:

```
break;
```

Оператор `continue`

Оператор `continue` похож на оператор `break` за тем исключением, что он не приводит к выходу из цикла, вместо этого он заставляет цикл пропустить весь код, который следует после оператора и начать следующую итерацию цикла. Например, цикл

```

1     for (int i=0; i<=8; i++)
2     {
3         if (i==4)
4             continue;
5
6         printf("%i ", i);
7     }

```

выведет числа от 0 до 8, пропустив число 4:

```
0 1 2 3 5 6 7 8
```

Операторы `break` и `continue` мы разберем более подробно в будущих уроках. Рекомендуется ими не злоупотреблять, т.к. это может усложнить чтение кода и повысить вероятность допущения ошибки.

Упражнения

2. Постарался использовать максимум выученного:

```

1 // Table of n and n^2, for n = 1 to 10
2
3 #include <stdio.h>
4
5 int main (void)
6 {
7     int n;
8

```



```

9      printf (" EXCERCISE 2 \n\n");
10     printf (" n      n^2 \n");
11     printf (" ---      -----\n");
12
13     for (n = 1; n <= 10; n++)
14         printf ("%3i      %i \n", n, n*n);
15
16     return 0;
17 }

```

3. Здесь то же самое, что и в прошлом задании, изменится только цикл:

```

1 // Every 5th triangular number from 5 to 50
2
3 #include <stdio.h>
4
5 int main (void)
6 {
7     int n;
8
9     printf (" EXCERCISE 3 \n\n");
10    printf (" n      Triangle \n");
11    printf (" ----      -----\n");
12
13    for (n = 5; n <= 50; n += 5)
14        printf (" %2i      %i \n", n, n*(n+1)/2);
15
16    return 0;
17 }

```

4. Факториалы. Здесь будет нужен вложенный цикл. А еще я сюда прилепил 6 упражнение, написав %-2i вместо %2i:

```

1 // Table of factorials from 1 to 10
2
3 #include <stdio.h>
4
5 int main (void)
6 {
7     int n, result;
8
9     printf (" EXCERCISE 4 \n\n");
10    printf (" n      n! \n");
11    printf (" ---      -----\n");
12
13    for (n = 1; n <= 10; n++) {
14        result = 1; //reset the result variable
15
16        for (int i = 1; i <= n; i++)
17            result *= i;
18
19        printf (" %-2i      %i \n", n, result);
20    }
21    return 0;
22 }

```

Действительно таблица получилась чуть красивее.

7. Здесь показывается, что при выравнивании можно вывести нули вместо пробелов. Так, если мы выведем число 3 через оператор %.2i, то будет выведено 03. А если через оператор %.4i, то 0003.
8. Тут достаточно добавить считывание переменной с клавиатуры:

```

1  #include <stdio.h>
2
3  int main (void)
4  {
5      int  iter, n, number, triangularNumber, counter;
6
7      printf ("How many calculations do you want? ");
8      scanf ("%i", &iter);
9      printf ("\n");
10
11     for ( counter = 1; counter <= iter; ++counter ) {
12         printf ("What triangular number do you want? ");
13         scanf ("%i", &number);
14
15         triangularNumber = 0;
16
17         for ( n = 1; n <= number; ++n )
18             triangularNumber += n;
19
20         printf ("Triangular number %i is %i\n\n", number,
21                 triangularNumber);
22     }
23
24     return 0;
25 }

```

10. Нарисуются минусы после каждой ненулевой цифры. Что логично, т.к. деление отрицательного числа на положительное дает отрицательное число.
11. Здесь нужно поправить пару строчек в программе, в которой мы переворачивали число:

```

1  // Calculating the sum of digits of the number
2
3  #include <stdio.h>
4
5  int main (void)
6  {
7      int  number, right_digit, result = 0;
8
9      printf ("Enter your number: ");
10     scanf ("%i", &number);
11
12     while ( number != 0 ) {
13         right_digit = number % 10;
14         result += right_digit;
15         number = number / 10;
16     }
17
18     printf ("Sum of its digits = %i\n", result);
19
20     return 0;
21 }

```

Источники:

1. Stephen Kochan – Programming in C (4th Edition); chapter 4
2. <https://beginnersbook.com/2014/01/c-for-loop/>
3. <http://www.c-cpp.ru/content/printf>