Цикл со счётчиком for

Данная циклическая конструкция обычно используется, когда заранее известно, сколько раз необходимо повторить какую-то команду или набор команд. Вспомним урок про генерацию случайных чисел: там была забавная задача для самостоятельного исследования. Вот её условие:

Пример: Равномерность случайных чисел

Числа, генерируемые функцией **rand()**, имеют равномерное распределение. Это значит, что если запускать функцию rand очень много раз и каждый раз записывать, какое число выпало, то различные числа выпадут примерно одинаковое число раз.
Например, если генерировать только числа 0 и 1, то через 100 запусков примерно 50 раз выпадет ноль и 50 раз единичка. Обратите внимание, что я говорю примерно. Может быть, например, 49 и 51, или 53 и 47.

Если рассматривать отношение этих чисел к общему количеству генераций, получим 49/100 и 51/100 или 53/100 и 47/100. Но чем больше экспериментов мы проведём, тем отношение количества единичек к количеству испытаний будет ближе к 1/2.
Проведите самостоятельно эксперимент с 10, 50 и 100 запусками. Это муторно и долго, если делать руками, но что поделать? В будущем мы напишем программу, чтобы проверить свойство равномерности распределения этих псевдослучайных чисел.

Давайте проведём подобный эксперимент. Пусть программа генерирует одно из трёх чисел: 0, 1 или 2. Вот, посмотрите на её код.

Листинг 1.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void) {

  srand(time(NULL));
// счётчики для сгенерированых чисел
// если выпадет 1, то увеличим на единицу count_1
  int count_0 = 0, count_1 = 0, count_2 = 0;

  int rand_number;

  rand_number = rand()%3;
  switch (rand_number){
    case 0 : count_0 = count_0 + 1; break;
    case 1 : count_1 = count_1 + 1; break;
    case 2 : count_2 = count_2 + 1; break;
  }

  rand_number = rand()%3;
```

```
switch (rand_number){
  case 0 : count_0 = count_0 + 1; break;
  case 1 : count_1 = count_1 + 1; break;
  case 2 : count_2 = count_2 + 1; break;
}

rand_number = rand()%3;
switch (rand_number){
  case 0 : count_0 = count_0 + 1; break;
  case 1 : count_1 = count_1 + 1; break;
  case 2 : count_2 = count_2 + 1; break;
}

rand_number = rand()%3;
switch (rand_number){
  case 0 : count_0 = count_0 + 1; break;
  case 1 : count_1 = count_1 + 1; break;
  case 2 : count_2 = count_2 + 1; break;
}

rand_number = rand()%3;
switch (rand_number){
  case 0 : count_0 = count_0 + 1; break;
  case 1 : count_1 = count_1 + 1; break;
  case 2 : count_2 = count_2 + 1; break;
}

rand_number = rand()%3;
switch (rand_number){
  case 0 : count_0 = count_0 + 1; break;
  case 1 : count_1 = count_1 + 1; break;
  case 2 : count_2 = count_2 + 1; break;
}

rand_number = rand()%3;
switch (rand_number){
  case 0 : count_0 = count_0 + 1; break;
  case 1 : count_1 = count_1 + 1; break;
  case 2 : count_2 = count_2 + 1; break;
}

rand_number = rand()%3;
switch (rand_number){
  case 0 : count_0 = count_0 + 1; break;
  case 1 : count_1 = count_1 + 1; break;
  case 2 : count_2 = count_2 + 1; break;
}

rand_number = rand()%3;
switch (rand_number){
  case 0 : count_0 = count_0 + 1; break;
  case 1 : count_1 = count_1 + 1; break;
  case 2 : count_2 = count_2 + 1; break;
}
```

```
  rand_number = rand()%3;
  switch (rand_number){
    case 0 : count_0 = count_0 + 1; break;
    case 1 : count_1 = count_1 + 1; break;
    case 2 : count_2 = count_2 + 1; break;
  }

  printf("0 - %d\n1 - %d\n2 - %d\n", count_0, count_1, count_2);

  return 0;
}
```

Если вам непонятно, как работает данная программа, то проработайте уроки «Приручаем случайность» и «Оператор выбора».

Программа получилась объемной, но довольно простой. При этом легко видеть, что одни и те же операции (генерация случайного числа и оператор выбора) повторяются буквально без изменений. Результат работы этой программы на рисунке ниже.

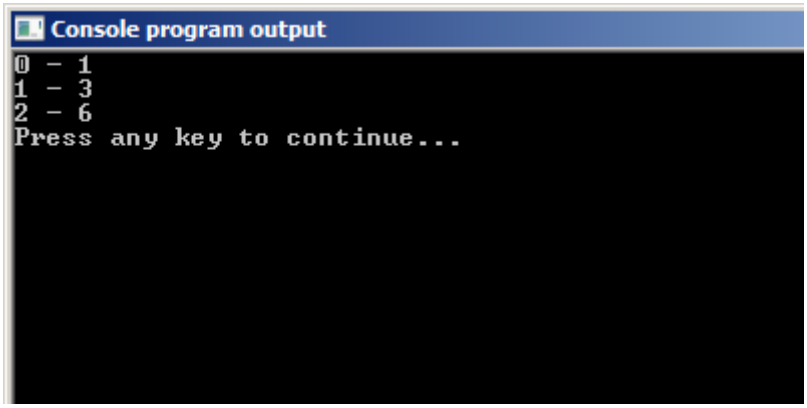

Рис.1 Пример работы программы Листинг 1

Как видите, пока что-то не очень похоже на то, что все цифры выпадают одинаковое количество раз. Запустите программу у себя, возможно, у вас получится более хороший результат. Ну, а мы продолжим. Давайте увеличим количество опытов до 100. Можно было бы, конечно, ещё 90 раз скопировать эту конструкцию или запустить программу ещё 9 раз и вручную складывать результаты, но это не дело. Лучше воспользуемся циклической конструкцией **for**.

Общий шаблон цикла for

Листинг 2.

```
for (инициализация счетчика; условие; изменение счетчика)
  оператор;
```

Цикл **for** состоит из двух частей: заголовок цикла (первая строка) и тело цикла (вторая строка).

Тело цикла – это команда, которую необходимо выполнить несколько раз. Если необходимо выполнить несколько команд, а не одну, то можно воспользоваться составным оператором **{}**. Мы уже проделывали такой трюк, когда изучали условный оператор.

В нашем случае тело цикла **for** будет выглядеть так:

Листинг 3.

```
{
  rand_number = rand()%3;
  switch (rand_number){
    case 0 : count_0 = count_0 + 1; break;
    case 1 : count_1 = count_1 + 1; break;
    case 2 : count_2 = count_2 + 1; break;
  }
}
```

Так как в теле цикла две команды (присваивание, оператор выбора), то пришлось использовать составной оператор.

Разберём подробно заголовок цикла **for**. В нём, кроме ключевого слова **for** и скобок, имеется ещё три выражения.

Первое выражение – инициализация счетчика.

В этой части нам нужно завести переменную-счетчик и присвоить ей какое-нибудь начальное значение. Например:

Листинг 4.

```
for (int i = 0; ; )
// или
for (int j = 13; ; )
// или
for (int k = -100; ; )
// или
for (int m = 255; ; )
// или
for (int q = 1; ; )
```

Второе выражение – условие

В этой части необходимо записать условное выражение, при котором должно выполняться тело цикла. Цикл **for** будет работать, пока условие истинно. Например:

Листинг 5.

```
for (int i = 0; i <= 100; )
// или
```

```
for (int j = 13; j > 0; )
// или
for (int k = -100; k != 0; )
// или
for (int m = 255; m >= 0; )
// или
for (int q = 1; q < 1000; )
```

И последняя третья часть – изменение счетчика.

Здесь записывается то, что должно происходить после каждого выполнения тела цикла. Обычно здесь пишут то, как должен измениться счётчик после каждой итерации. Одно повторение тела цикла в серьёзных книгах иногда называют итерацией.

Например:

Листинг 6.

```
for (int i = 0; i <= 100; i = i + 1) // увеличиваем счётчик на
единицу
// или
for (int j = 13; j > 0; j = j / 2) // уменьшаем вдвое
// или
for (int k = -100; k != 0; k = k - 1) // уменьшаем счётчик на
единицу
// или
for (int m = 255; m >= 0; m = m - 5) // уменьшаем счётчик на 5
// или
for (int q = 1; q < 1000; q = q * 2) //увеличиваем счётчик в два
раза
```

Давайте перепишем нашу программу с использованием цикла **for**.

Листинг 7.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void) {
  srand(time(NULL));
// счётчики для сгенерированных чисел
// если выпадет 0, то увеличим на единицу count_0
// аналогично для count_1 и count_2
  int count_0 = 0, count_1 = 0, count_2 =0;

  int rand_number;

  // i = 0 начинаем отсчёт с нуля
  // i < 100 повторяем, пока i меньше 100
```

```
  // i = i + 1 на каждом шаге увеличиваем счётчик на единицу
  // итого тело цикла выполнится 100 раз
  for (int i = 0; i < 100; i = i + 1){
    rand_number = rand()%3;
    switch (rand_number){
      case 0 : count_0 = count_0 + 1; break;
      case 1 : count_1 = count_1 + 1; break;
      case 2 : count_2 = count_2 + 1; break;
    }
  }

  printf("0 - %d\n1 - %d\n2 - %d\n", count_0, count_1, count_2);
  return 0;
}
```

Гораздо понятнее и нагляднее, не так ли? Вот-вот. Опишем подробно алгоритм работы цикла **for**.

1. Программа встречает ключевое слово **for**, а значит дальше идёт циклическая конструкция.
2. Сначала создаётся переменная счетчик (выражение 1).
3. Проверяется условие выполнение цикла (выражение 2).
4. Если условие **ЛОЖЬ**, то программа выходит из цикла **for** и продолжает свое выполнение.
5. Если условие **ИСТИНА**, то выполняется тело цикла.
6. Когда тело цикла выполнится, программа возвращается к заголовку и выполняет инструкции из третьей части заголовка. Изменяется значение переменной-счетчика (выражение 3).
7. Возвращаемся к пункту три.

А теперь то же самое для нашей программы. Встречаем цикл **for**.

- Инициализируем переменную счётчик **i = 0**;
- Проверяем условие **i < 100**. Т.к. **i = 0**, то условие **ИСТИНА**;
- Выполняем тело цикла. Генерируем число от 0 до 2. В операторе switch определяем, какое число было сгенерировано, и увеличиваем соответствующий счётчик;
- Т.к. тело цикла закончено, то выполняем Выражение 3. Теперь **i = 1**.
- Возвращаемся к условию;

- Проверяем условие **i < 100**. Т.к. **i = 1**, то условие **ИСТИНА**
- Выполняем тело цикла. Генерируем число от 0 до 2. В операторе switch определяем, какое число было сгенерировано, и увеличиваем соответствующий счётчик;
- Изменяем счётчик. Теперь **i = 2**. Возвращаемся к условию.

И так далее, пока **i** не станет равным **100**. В этом случае при проверке условия получим **ЛОЖЬ**. Выполнение цикла прекратится. Программа продолжит выполнять инструкции, расположенные за циклом.

Давайте посмотрим ещё на один пример.

Напишем программу, которая выводит на экран сумму натуральных чисел из промежутка **[A;B]**. Числа **A** и **B (B>A)** вводятся пользователем.

Листинг 8.

```
#include <stdio.h>

int main(void) {

  int a = 0, b = 0;
  scanf("%d %d", &a, &b);

  // сюда будем записывать результат
  int sum = 0;


  // k++ эквивалентно k = k + 1, но короче
  // ++ оператор инкремента

  for (int k = a; k <= b; k++){
    sum = sum + k;
    // на каждой итерации добавляем к уже имеющейся сумме
    // очередное число
  }

  printf("%d\n", sum);

  return 0;
}
```

Все неясные моменты я постарался отразить в комментариях к программе.

Практика

1. Решите предложенные задачи с автоматической проверкой решения.

Интернет версия: http://youngcoder.ru/lessons/7/cikl_for.php