ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

ЦИКЛЫ В С. ДВИЖЕНИЕ ПРИ ПОМОЩИ БЕСКОНЕЧНОГО ЦИКЛА. СЧЕТЧИКИ

Как мы уже говорили, суть технологии состоит в том, чтобы создать такие цепочки инструкций, которые гарантированно обеспечат превращение ресурсов в продукты. Такие действия очень часто необходимо многократно повторять. Или повторять до того момента, пока не будет выполнено определенное условие. Например, при вязании шарфа необходимо набирать определенное количество петель в каждом ряду, а по достижении нужного количества - начинать новый ряд. Если достигнуто необходимое количество рядов, вязание шарфа прекращается.

Для описания такой ситуации в языке программирования С существует специальная команда `while`, которая, по сути, является функцией с одним аргументом. Аргумент этой функции может принимать только два значения:

- **ложь, false или 0 –** в этом случае тело функции не выполняется;
- **истина, true, или 1,** или любое **иное значение –** в этом случае тело функции будет выполняться.

Давайте представим функцию `while` для набора петель:

```
while(0)
      {
      Набирать петлю ();
      }
```

Согласно этому коду не будет набрано ни одной петли. Ведь аргумент функции 0.

Обратите также внимание на правило написания кода. Тело функции `while ()`, заключенное между фигурными скобками, записано с одинаковым отступом от левого края. Это делать не обязательно, но такая запись очень облегчает чтение кода.

Теперь изменим аргумент на 1:

```
while(1)
      {
      Набирать петлю ();
      }
```

Программа будет выполняться следующим образом: тело функции будет выполнено, а затем будет осуществляться проверка того, чем является аргумент функции `while (1)`. В нашем случае он равен 1, а значит, и функция будет выполнена еще раз.

Одно повторение или выполнение тела цикла называется итерацией.

Согласно этому коду будет связан один бесконечный ряд шарфа. Пользоваться таким изделием будет совершенно невозможно!

Давайте договоримся, что в нашем шарфе в ряду будет 50 петель, а сам шарф будет состоять из 1000 рядов.

Очевидно, что необходимо создать цикл, который позволил бы вязать первые 50 петель, а затем прекращал бы вязание. Этого можно достичь в случае, если при попытке набрать 51-ую петлю аргумент функции `while` становился бы ложным. Помочь добиться подобного эффекта нам могут так называемые операторы сравнения.

Результатом действия логических операторов и операторов сравнения всегда является либо «истина», что соответствует логической «1», либо «ложь», что соответствует логическому «0» (если переменная принимает значения 0 и 1, то она соответствует типу данных bool).

Логические операторы и операторы сравнения можно представить в виде вопроса с двумя вариантами ответа. Операторы сравнения действуют на любые переменные, логические операторы же применяются только для переменных типа bool.

Существует всего шесть операторов сравнения. Основные из них приведены в таблице ниже:

`7==5;`	Осуществляется сравнение чисел 5 и 7. Результат этой операции - «ложь», или логический «0». Это действие эквивалентно вопросу: семь равно пяти?

`7>5;`	Осуществляется сравнение чисел 5 и 7. Результат этой операции - «истина», или логическая «1». Это действие эквивалентно вопросу: семь больше пяти?
`7<=5;`	Осуществляется сравнение чисел 5 и 7. Результат этой операции - «ложь», или логический «0». Это действие эквивалентно вопросу: семь меньше или равно пяти?
`7!=5;`	Осуществляется сравнение чисел 5 и 7. Результат этой операции - «истина», или логическая «1». Это действие эквивалентно вопросу: семь не равно пяти?

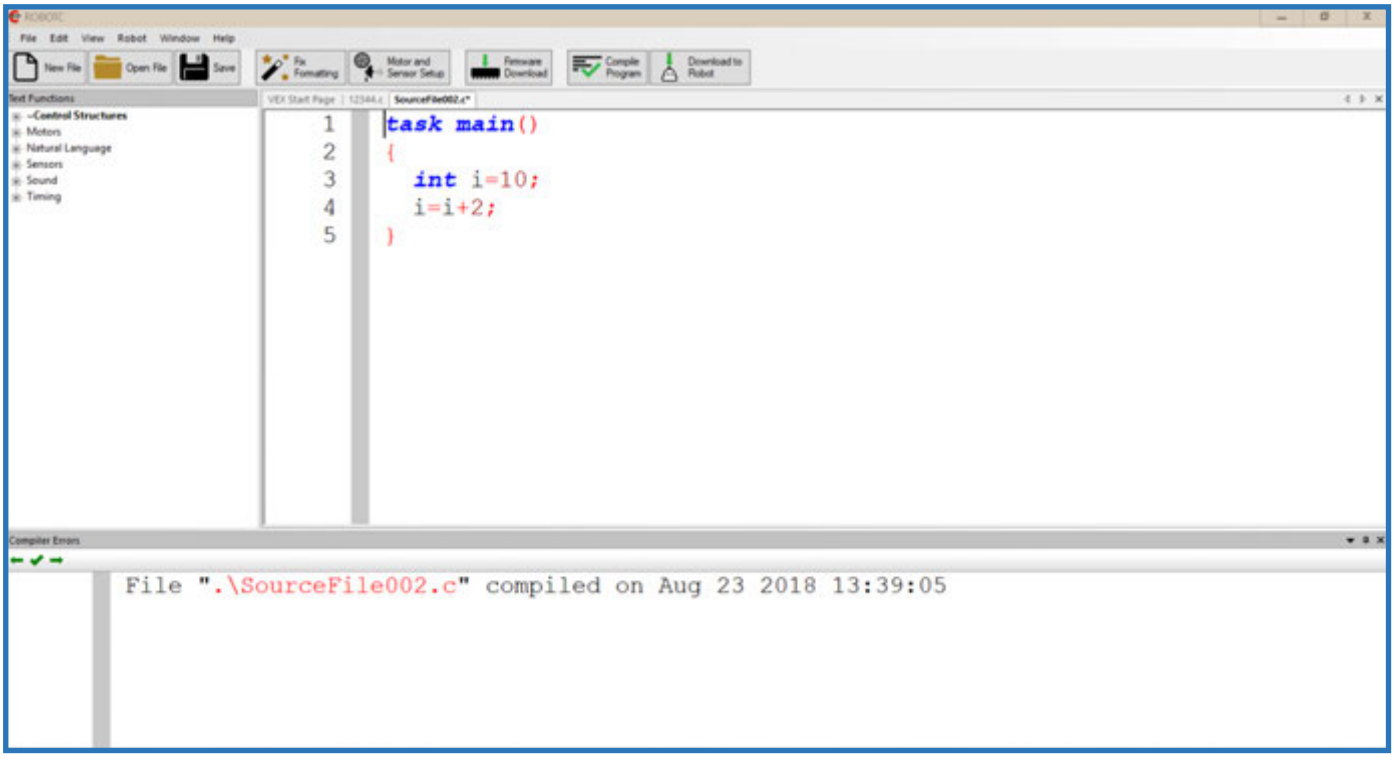
Сравнение переменных осуществляется оператором «==». Обратите внимание на то, что **«a=b» - это операция присваивания**, и в результате ее выполнения произойдет не сравнение переменных a и b, а переменной a будет присвоено значение переменной b.

Модернизируем нашу функцию:

```
while(количество петель < 51)
      {
      набирать петлю ();
      }
```

Но и эта функция будет выполняться бесконечно, ведь внутри функции не происходит подсчета количества набранных петель.

Для того чтобы осуществить подсчет количества итераций, в языке С используется особенность операции присваивания.



После выполнения этих инструкций переменная **i** станет равной **12**. Давайте разберемся, как это происходит.

В третьей строке переменной **i** присваивается значение **10**. В четвертой же строке к **i** прибавляется **2**, а затем это значение присваивается переменной **i**. Такой эффект достигается за счет того, что у операции присваивания самый низкий приоритет среди всех операций, подобно тому как приоритет умножения выше, чем приоритет сложения.

Таким образом, цикл для набора одного ряда петель на шарфе будет выглядеть следующим образом:

```
количество петель = 1;
while(количество петель < 51)
      {
      Набирать петлю ();
      количество петель = количество петель + 1;
      }
```

После каждой набранной петли программа будет увеличивать их количество на 1, пока оно не станет равным 51. Как только это значение будет достигнуто, цикл перестанет выполняться и программа перейдет к следующим инструкциям, например, как в нашем случае, к следующему ряду шарфа.

```
количество петель = 1;
while(количество петель < 51)
      {
      Набирать петлю ();
      количество петель = количество петель + 1;
      }
Перейти к следующему ряду ();
```

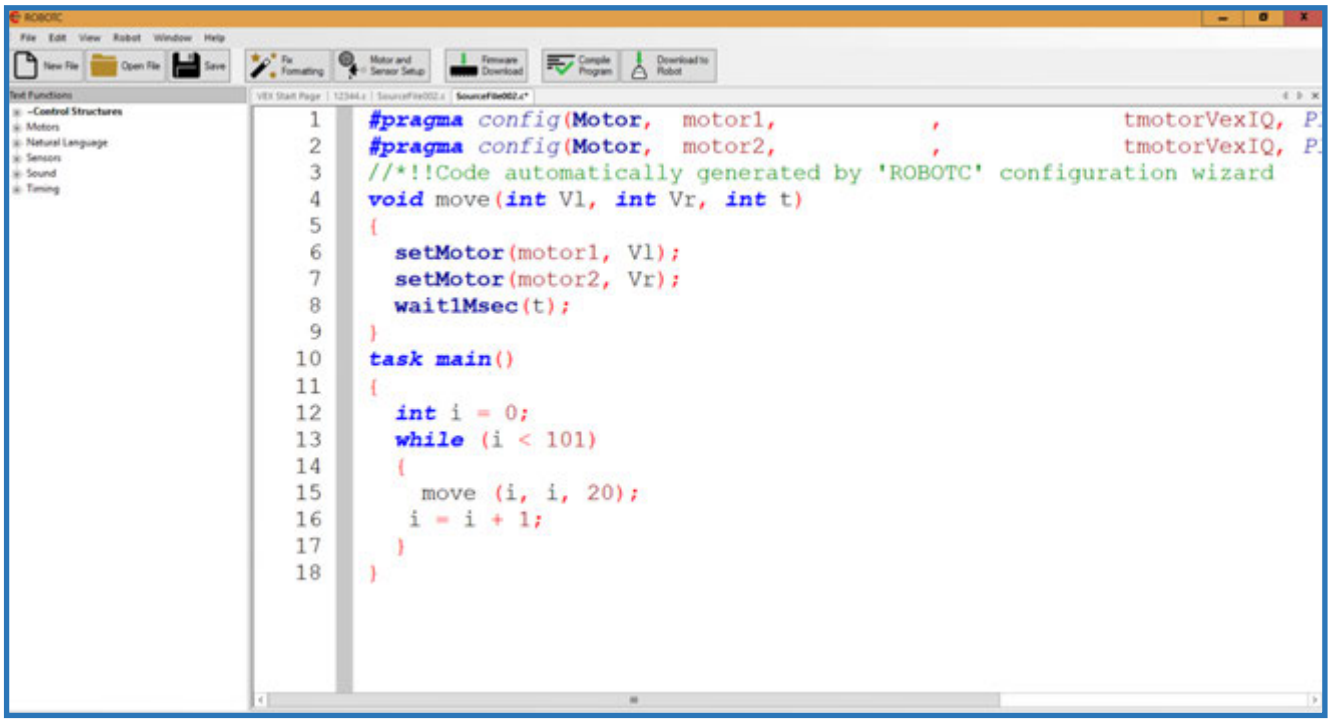
Программа же для всего шарфа будет выглядеть следующим образом:

```
количество рядов = 1;
количество петель = 1;
while(количество петель < 1001)
     {
     while(количество петель < 51)
          {
          Набирать петлю ();
          количество петель = количество петель + 1;
          }
Перейти к следующему ряду ();
количество рядов = количество рядов + 1;
```

Это так называемые **вложенные циклы**. Для того чтобы связать наш шарф в 1000 рядов по 50 петель каждый, в одной итерации внешнего цикла мы поместили 50 итераций внутреннего цикла.

Значит ли это, что при помощи циклов мы можем задать роботу любую задачу, имеющую многократно повторяющиеся действия?

Безусловно, да. Например, при помощи бесконечного цикла мы можем заставить робота плавно разгоняться.



```
1  #pragma config(Motor,  motor1,           ,             tmotorVexIQ, P
2  #pragma config(Motor,  motor2,           ,             tmotorVexIQ, P
3  //*!!Code automatically generated by 'ROBOTC' configuration wizard
4  void move(int Vl, int Vr, int t)
5  {
6    setMotor(motor1, Vl);
7    setMotor(motor2, Vr);
8    wait1Msec(t);
9  }
10 task main()
11 {
12   int i = 0;
13   while (i < 101)
14   {
15     move (i, i, 20);
16    i = i + 1;
17   }
18 }
```

Каждые 20 миллисекунд (третий аргумент функции `move` в 15 строке) робот на единицу увеличивает свою скорость i (первый и второй аргумент функции `move` в 15 строке).

Итак, если нам нужно, чтобы робот выполнял какое-либо действие многократно, мы задаем ему бесконечный цикл. Если же выполнение действия в какой-то момент нужно прервать либо ограничить каким-либо условием, мы применяем операторы сравнения. Счетчики ведут учет количества итераций и в условленный момент, когда оператор возвращает значение «Ложь», действие цикла прекращается.