Loop Control Structures

In a C program there may be situations where we may want to execute same block of statements many times. All programming languages offers loop control structure (also known as **looping**) allowing programmers to execute a statement or group of statements multiple times. In this chapter we will discuss the loop (repetitive) control structure provided by the C programming language. Here 'control structure' mean that the flow of execution may not be sequential and control may be transferred to any statement based on given conditions in our program.

In looping, the sequences of statements are executed until some exit condition is satisfied. The looping construct is composed of two parts : *body of loop and control statement.* Depending on the place of control statement in loop, it can be classified as **entry controlled** and **exit controlled** loop. In entry controlled loop the exit/control condition is checked before executions of statements inside loop body. In exit controlled loop the exit/control condition is checked after executions of loop body. This means that in case of exit controlled loop, body will be executed at least once before exiting from the loop. Further discussion related to the same is given during explanation of different types of loop.

Now let us discuss syntax and general rules related to following three basic loop control structure provided by C language :

- for
- while
- do…while

The For Loop

The *for* loop is normally used when we want to execute block of statements fixed number of times. To make *for* loop more dynamic, we can use the exit condition inside for loop. Let us try to understand the use of simple *for* loop statement by example14. 1. This example displays five " * " using simple *for* loop statement. Figure 14.1 gives the code listing and output of example 14.1.

11-1.c - SciTE

File Edit Search View Tools Options Language Buffers Help

11-1.c

```
/* Example 1: Program to illustrate simple for loop statement.*/
#include<stdio.h>

int main()
{
    int count;                              // declaration of variables
    for ( count = 0; count < 5 ; count++)  // for loop body repeats 5 times
        printf(" * ");
    return 0;
}
/*End of Program */
```

```
gcc 11-1.c
>gcc 11-1.c
>Exit code: 0
./a.out
>./a.out
 * * * * * >Exit code: 0
```

Figure 14.1 : Code listing and output of Example 14.1

Explanation

In the beginning of program one integer variable count is declared. Variable count is used as a counter variable in a for loop. Firstly variable count is initialized to zero. Then second expression (count < 5) is checked. When it is evaluated to true, the body of for loop is executed. In our case one " * " will be displayed on screen using printf statement. Then third for loop expression (count++) will be executed, incrementing value of count by 1. Again second expression (count<5) is evaluated and depending on its result, body of for loop will be executed. In our case five times body part will be executed displaying five stars " * * * * * " on the screen.

Syntax of For Loop

```
for ( expression1; expression2; expression3)
{
        Statement-block;
}
```

The header of *for* loop contains three expressions separated by semicolons in parenthesis. All these expressions are optional. Statement-block which is also known as body of *for* loop may contain simple statement or compound statement. Following figure 14.2 shows the flowchart of *for* loop.

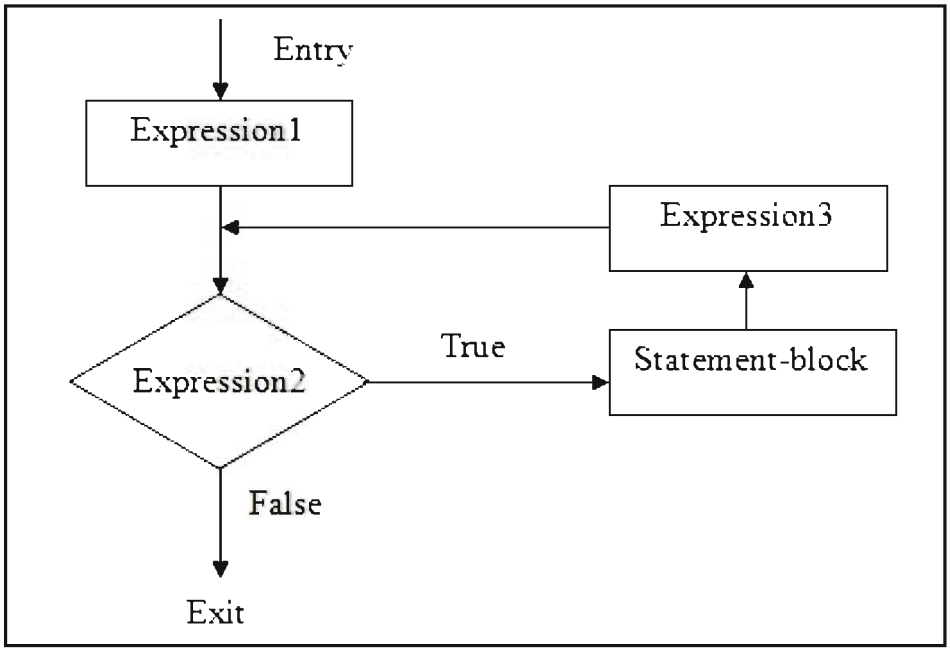

Figure 14.2 : Flowchart of for loop

As shown in figure14.2, the steps of executing for loop are as follows :

Step 1 : Evaluate expresison1. The expression1 will be evaluated only once when the for loop starts.

Step 2 : Evaluate expresison2. If its value is false then terminate the loop.

Step 3 : If expression2 is evaluated to true then execute the statements in the body of loop.

Step 4 : Evaluate expression3.

Step 5 : Go to Step 2.

Note following important points related to for loop :

- expression1 works for initializing value of counter variable that is used to control the number of times a loop is to be executed. This variable is known as control variable.
- expression2 works as test condition. A control variable is used for checking loop terminating criteria.
- expression3 works for incrementing or decrementing value of control variable.

Nested for loop

Nested for loop means using one for loop within another for loop. Let us try to understand this concept using example14.2. This example prints the following pattern for given number of lines. For example if number of lines are 4 then display output as follow:

1

1 2

1 2 3

1 2 3 4

Figure 14.3 gives the code listing of the example 14.2 while figure 14.4 shows its output.

11-2.c - SciTE

File Edit Search View Tools Options Language Buffers Help

11-2.c

```
/* Example 2: Program to illustrate the use nested for loop.*/
#include<stdio.h>

int main()
{
    int i, j, lines;                        // declaration of variables
    printf("Enter number of lines : ");     // message to user
    scanf("%d", &lines);                    // stores value in lines variable

    for ( i = 1; i <= lines ; i++)          // Outer loop, repeat number of lines time
    {
        for( j = 1 ; j <= i ; j++)    // inner loop for printing numbers from 1 to value of i
        {
                printf("%d ", j);           // print value of j
         }
        printf("\n");                       // move cursor to next line
    }
    return 0;
}
/* End of Program */
```

Figure 14.3 : Code listing of Example 14.2

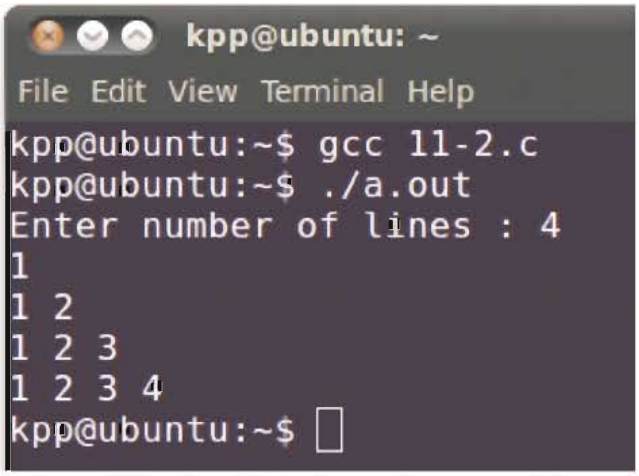

Figure 14.4 : Output of Example 14.2

Explanation

The program contains outer for loop with control variable i, which is executed 'lines' number of times by varying value of i 1 by 1 up to the value of lines. For each i in the outer loop, inner loop with control variable j is executed i times, and display numbers varying 1 by 1 up to i. After printing i numbers in line i, printf("\n") statement of outer loop will bring cursor on the next line. This enables the next line number to be printed in new line.

Use of Comma Operator in for Loop

In for loop we can initialize more than one parameters using comma operator. Similarly we may use comma operator to increment or decrement various variables in a for loop. Let us try to understand the use of comma operator using example14.3. Figure 14.5 gives the code listing of the example 14.3 while figure 14.6 shows its output.

11-3.c - SciTE

File Edit Search View Tools Options Language Buffers Help

11-3.c

```
/* Example 3: Program to illustrate the use comma operator in a for loop
   to print the addition table.*/

#include<stdio.h>
int main()
{
    int i, j, number;                          // declaration of variables
    printf("Enter number of lines in a table: "); // message to user
    scanf("%d", &number);                      // stores value in number variable
    for ( i=0, j=10; i < number; i++, j- -)
    {
            printf("%d  +  %d  =  %d\n", i, j, i + j );
    }
    return 0;
}
/* End of Program */
```

Figure 14.5 : Code listing of Example 14.3

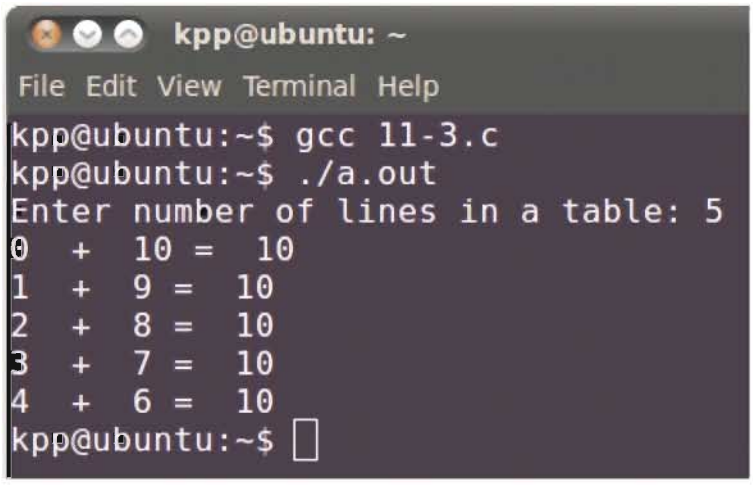

Figure 14.6 : Output of Example 14.3

Note that in example 14.3 we have initialized two variables i=0 and j=10. Also we are incrementing i with value one and decrementing j with value 1 in the same *for* loop. Based on the input given we will get output as shown in the example.

The While Loop

When number of iteration cannot be pre-determined and when loop terminating condition is to be tested before entering the loop, at that time use of while loop is more suitable. For example,

- Find sum of all numbers entered until user enters zero
- Display menu options and take proper action until user select exit option

Syntax of While Loop

```
while (test expression)
{
        statement-block;    /* body of while loop */
}
```

Flow chart of execution of while loop is shown in figure14.7.

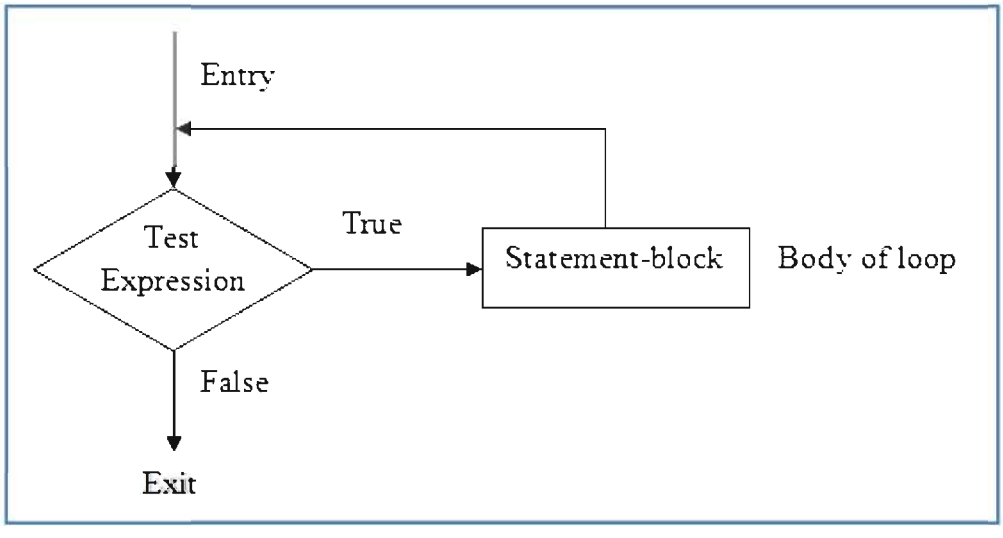

Figure14.7 : Flowchart of while loop

As seen in the figure14.7, it evaluates the test expression first. If test expression is evaluated to true, then body of loop containing statement-block is executed. Again program evaluates the test expression. This process is repeated until the test expression is evaluated to false. When test expression is evaluated to false then loop is terminated and control is passed to the next statement after the loop in a program. Body of loop may contain simple or compound statement.

Note that as we are checking condition at the entry point, it is known as *entry-controlled* loop. If test expression is evaluated to false, the body part will not be executed at all.

Let us try to understand the use of *while* loop using example 14.4. The program given in example14.4 finds sum of all numbers entered until user enter zero. Figure 14.8 gives the code listing of the example 14.4 while figure 14.9 shows its output.

11-4.c - SciTE

File Edit Search View Tools Options Language Buffers Help

11-4.c

```
/* Example 4: Program to illustrate use of while loop.*/
#include<stdio.h>

int main()
{
    int number, sum=0;                          // declaration of variables
    printf("Enter 0 to exit from program.\n");
    printf("Enter your number : ");             // message to user
    scanf("%d", &number);                       // stores value in number variable
    sum = number;

    while(number != 0)
    {
        printf("Enter your number : ");         // message to user
        scanf("%d", &number);                   // stores value in number variable
        sum = sum + number;
     }
    printf("The sum of entered numbers : %d \n", sum);
    return 0;
}
/*End of Program */
```

Figure 14.8 : Code listing of Example 14.4

kpp@ubuntu: ~

File Edit View Terminal Help

```
kpp@ubuntu:~$ gcc 11-4.c
kpp@ubuntu:~$ ./a.out
Enter 0 to exit from program.
Enter your number : 5
Enter your number : 10
Enter your number : 0
The sum of entered numbers : 15
kpp@ubuntu:~$
```

Figure 14.9 : Output of Example 14.4

Explanation

First statement after main function declares two variables required in the program. The second statement informs user about how to exit from the program. Third and fourth statement displays message and read a number from the user respectively. Fifth statement assigns the value of number variable to sum variable. In sixth statement, we check the test condition i.e. value stored in number variable is not equal to zero. When this test condition evaluates to true, statement number seven and eight displays message and read a number from the user respectively. The tenth number of statement will be adding the entered number into the sum variable. After this again test condition of *while* loop will be evaluated. When test condition is evaluated to false, the program executes the tenth statement displaying the sum of all the numbers entered by the user.

The do-while loop

We have seen that in *while* loop, the test expression is checked before executing body of loop. There are certain cases where we may want to execute body of loop before test expression. The do...*while* loop should be used when the test expression is to be checked after executing body of loop. As we are checking test condition at the end of loop, the do...*while* loop is a kind of *exit-controlled* loop. Note that in *do...while* loop, body of loop will be executed at least once.

Syntax of do...while loop

```
do
{
        statement-block;        /* body of loop */
}
while( test expression);
```

Flow chart of *do*...while loop is shown in figure14.10.

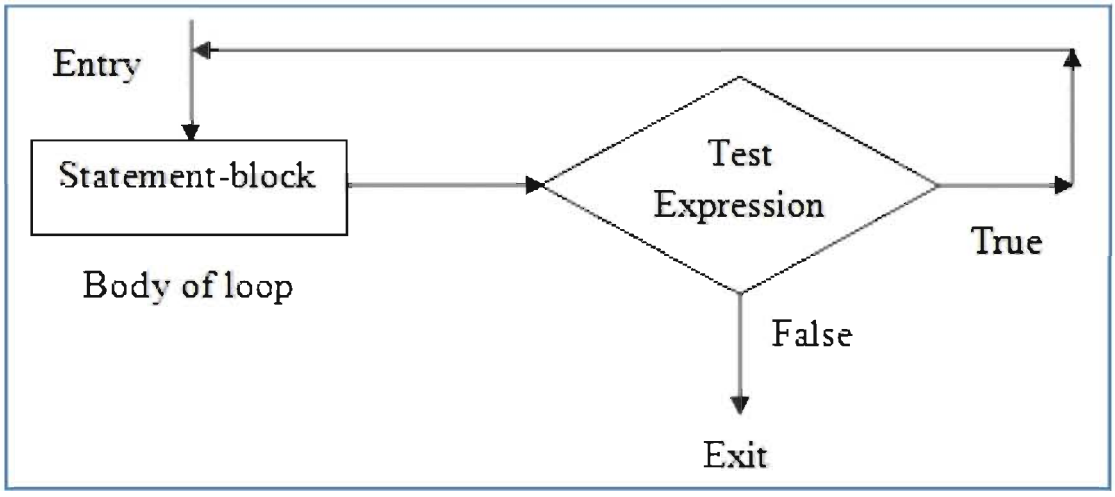

Figure 14.10 : Flowchart of do...while loop

As seen in the figure14.10 the body of loop containing statement-block is executed first. Then it evaluates the test expression. If the test expression is evaluated to true then body of loop containing statement-block is executed again. After executing statement-block, once again text expression is evaluated. This process is repeated until the test expression is evaluated to false. When test expression is evaluated to false then loop is terminated and control is passed to the next statement after the loop in a program. Body of loop may contain simple or compound statement.

Let us try to understand the use of do..*while* loop using example 14.5. Program given in the example14.5 will find sum of all numbers entered by the user, until user enters zero using do…*while* loop. Figure 14.11 gives the code listing of the example 14.5 while figure 14.12 shows its output.

11-5.c - SciTE

File Edit Search View Tools Options Language Buffers Help

11-5.c

```
/* Example 5: Program to illustrate use of do...while loop.*/
#include<stdio.h>

int main()
{
    int number, sum=0;                    // declaration of variables
    printf("Enter 0 to exit from program.\n");
    do
    {
        printf("Enter your number : ");   // message to user
        scanf("%d", &number);             // stores value in number variable
        sum = sum + number;
     } while(number != 0);

    printf("The sum of entered numbers : %d \n", sum);
    return 0;
}
/*End of Program */
```

Figure 14.11 : Code listing of Example 14.5

kpp@ubuntu: ~

File Edit View Terminal Help

```
kpp@ubuntu:~$ gcc 11-5.c
kpp@ubuntu:~$ ./a.out
Enter 0 to exit from program.
Enter your number : 3
Enter your number : 4
Enter your number : 0
The sum of entered numbers : 7
kpp@ubuntu:~$
```

Figure 14.12 : Output of Example 14.5

Explanation

First statement after main function declares two variables required in the program and initializes one of them. Second statement displays the message about how to exit from the program. At statement three the *do...while* loop starts. Statement fourth and fifth displays message and read a number from user respectively. Statement sixth adds number entered by user into sum variable. Statement seven evaluates the test condition. Statement fourth to sixth is executed repeatedly every time when test expression is evaluated to true. When test expression is evaluated to false, the program executes the eighth statement displaying the sum of all the numbers entered by the user.

Nested Loops

Nesting of different types of loops is possible by using one loop control structure within another, irrespective of the type of loop control structures used. For example, in *for* loop, *while* or *do...while* loop can be used. For example, display all prime numbers between 1 and 100. Loop for numbers can be written using *for* loop. Within this loop, while loop can be used to determine whether number is prime or not.

Selecting a Loop

So far in this chapter, we have discussed three looping construct for, while and do..while of C programming language. To select the more appropriate loop for our program, we may use the following steps :

- For a given problem, identify whether entry-controlled or exit-controlled loop is required.
- For entry controlled loop, we may use *for* or *while* loop construct. Use of for loop is advisable for counter based exit condition.
- For exit controlled loop, we have choice of *do..while* construct.

Also while using any loops in our program following three important points are to be considered carefully to avoid miscellaneous behavior of a loop:

- Initialization of loop counter
- To test exit condition to come out from the loop
- Incrementing or decrementing loop counter

Skipping a Part of Loop

During execution of a program, sometimes we may want to skip some of the statements or iterations of the loop. This is possible in C language by using following statements :

(i) break statement

(ii) continue statement

The break Statement

Sometimes during execution of any loop, we may want to terminate loop instantly without even execution of other statements of loop. This is possible using the break statement. The break statement in C programming language has following usages :

- When the break statement is encountered inside the loop, the loop is immediately terminated and program control executes the next statement following the loop.
- The break statement can also be used to terminate a case in the *switch* statement.

In case of nested loop, the *break* statement will stop the execution of currently executing loop and starts execution of next statement after the block of code.

Example14.6 shows how the *break* statement works in a program. Figure 14.13 gives the code listing as well as output of example 14.6.

11-6.c * SciTE

File Edit Search View Tools Options Language Buffers Help

11-6.c *

```
/* Example 6: Program to illustrate use of the break statement. */
#include <stdio.h>

int main ()
{
      int number = 1;
      while( number < 10 )
      {
            printf("Value of number is %d\n", number);
            number = number + 1;
            if( number > 5)
            {
              break;     // statement will terminate the loop.
            }
      }
      return 0;
}
/* End of Program*/
```

```
gcc 11-6.c
>gcc 11-6.c
>Exit code: 0
./a.out
>./a.out
Value of number is 1
Value of number is 2
Value of number is 3
Value of number is 4
Value of number is 5
>Exit code: 0
```

Figure 14.13 : Code listing and output of Example 14.6

Explanation

First statement after main function initializes an integer variable. The test condition in *while* statement is written to execute loop nine times. The printf statement inside loop will print the value of number variable. After printing value of number variable it is incremented by 1. Next, *if* statement will be executed. But during execution of program when *if* statements is evaluated to true, the *break* statement executes; causing termination of while loop. Hence in our program, printf statement inside the while loop will be executed only five times.

The continue Statement

The continue statement works to some extent like the break statement. Instead of forcing termination of loop, however, continue statement forces the next iteration of the loop to take place, skipping any code in between.

For example, in for loop, continue statement causes the conditional test and increment portions of the loop to execute. For the while and do...while loops, continue statement causes the program control passes to the conditional tests.

Let us see example14.7 which shows how continue statement works in a program. Figure 14.14 gives the code listing and output of example 14.7.

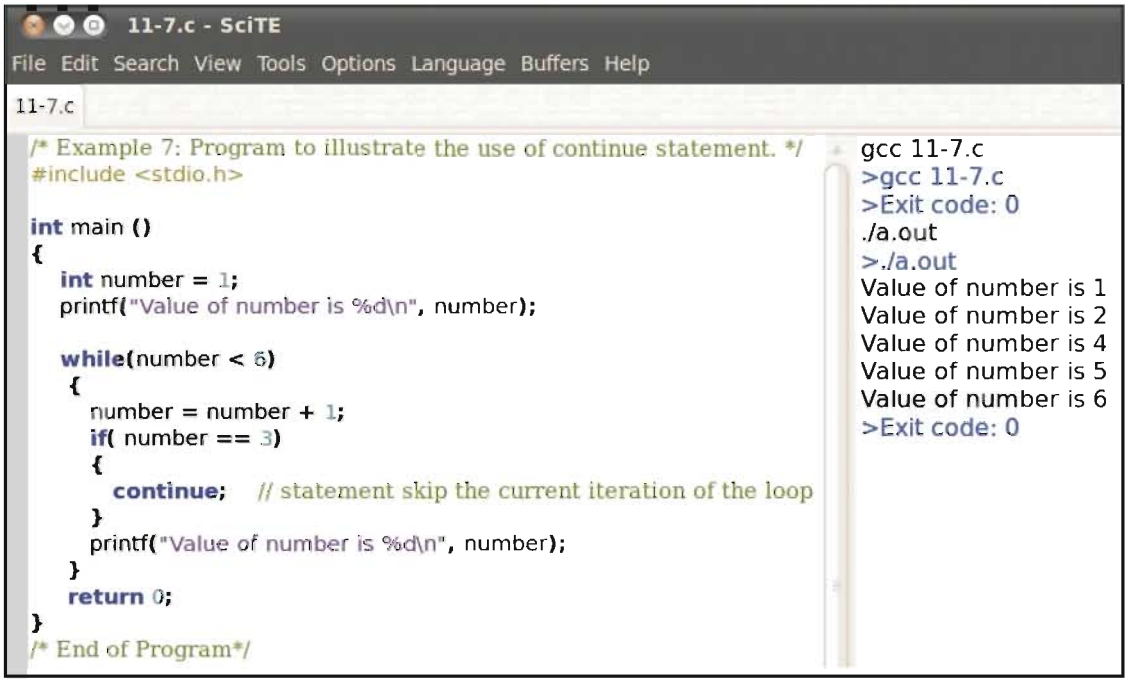

Figure 14.14 : Code listing and output of Example 14.7

Explanation

First statement declares and initializes a number variable required in the program. Second statement prints value of number variable. The test condition in while statement is written to execute loop five times. Inside while loop, the number is incremented by value 1. After this the if statement checks the test condition. When it is evaluated to true, the continue statement executes, which terminates the current iteration of while loop. Hence value of number 3 will not be printed. Program executes next iteration of loop causing program to print value of number variable 4, 5 and 6.

The Infinite Loop

Any loop in a program becomes infinite loop if it runs forever and program control never comes out of it. The loop becomes infinite due to non availability of exit condition in the logic of loop. Observe the program code logic of example 14.8 shown in figure 14.15.

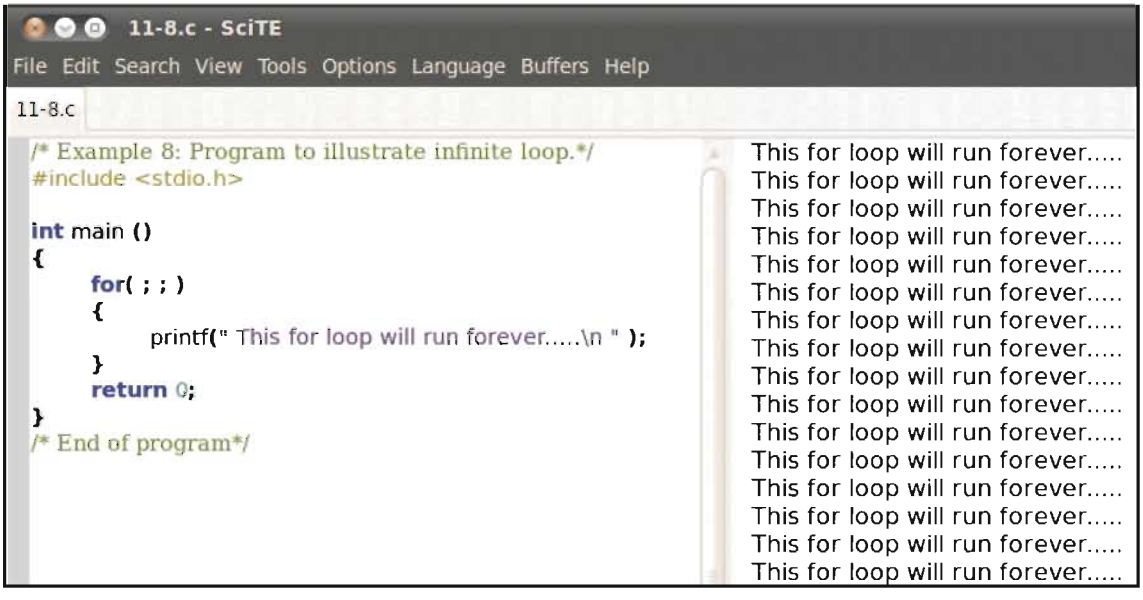

Figure 14.15 : Code listing and output of Example 14.8

Explanation

When we compile and execute this code it will runs forever. In C program *for* loop all three expressions are optional. When the conditional expression is absent inside *for* loop, it is assumed to be true. We may have initialization expression and increment expression inside for loop, but C program runs forever when we use for(; ;) construct. Note that you may use CTRL + C keys to terminate your infinite loop.

Summary

In this chapter we had learnt that how C languages offers **loop control structure** allowing programmers to execute a statement or group of statements multiple times. We had learnt three basic loop control structure (for, while and do...while) provided by the C language. We had discussed that how entry controlled and exit controlled loops are useful in our programs. We had also learnt that how nesting of various loops can be performed. At the end of chapter we had learnt that how during execution of a program; we can skip some of the statements or iterations of the loop using break and continue statements.

EXERCISE

1. Explain the syntax of for loop construct with suitable examples.
2. What do you mean by looping statement ? State use of looping statement in a C program.
3. What is the difference between while and do-while loop ?
4. Describe how to skip part or a loop using break and continue statement.
5. What do you mean by entry controlled and exit controlled loop ? Give suitable example for the same.
6. Fill in blanks :
   (a) Depending on the place of control statement in loop, it can be classified as _____ controlled and _____ controlled loop.
   (b) The for loop is example of _____ controlled loop.
   (c) The do-while loop is example of _____ controlled loop.
   (d) In exit controlled loop, body will be executed _____ time before exiting from the loop.
   (e) The _____ statement causes immediate exit from the loop control structure.
   (f) The _____ statement skips the subsequent statements of the loop and continues the next iteration of the loop.
7. State True or False :
   (a) Within for loop do...while loop cannot be used.
   (b) Execution of break statement inside the loop immediately terminates the execution of program.

(c) Initialization, test condition and increment parts are optional in a for loop header.

(d) The break statement cannot be used to terminate a case in the switch statement.

(e) Initialization, test condition and increment parts are separated by a commas.

(f) For every while clause, there must be a do.

(g) For every do clause, there must be a corresponding while.

8. Choose the correct option from the following :

(1) How many times printf statement is executed in the following program segment ?

```
int num1 = 3, num2 = 6;
while(num1 < num2) {
    printf(" Hello Students…");
    num1 = num1 + 1;
}
```

(a) 1 (b) 2 (c) 3 (d) 6

(2) What is the output of following program segment ?

```
int a = 5, b = 10;
do
{
    a = a + 1;
}while( a < = b);
printf(" %d", a);
```

(a) 10 (b) 9 (c) 11 (d) 15

(3) What is the output of following program segment ?

```
int main( ){
int i;
for(i = 0; i < 10; i++);
printf("%d", i);
}
```

(a) 0123456789 (b) 9 (c) 10 (d) Error

(4) What is the value of a number variable after execution of following code ?

```
int number = 1;
while(number < 5){
      number = number + 1;
      if( number = = 3) {
         continue;
      }
}
```

(a) 1 (b) 3 (c) 5 (d) None of these

LABORATORY EXERCISE

1. Change the following for loop into its equivalent while loop

(a)
```
for(number=0; number <5; number++) {
printf("%d", number);
}
```

(b)
```
for(number=5; number > 0; number- -) {
      printf("%d", number);
}
```

2. Write a program to demonstrate use of the compound statement within a while loop.
3. Write a program to find the sum of first 100 integers using for loop.
4. Demonstrate the use of while loop to find the sum of first 100 even integers.
5. Write a program to print following pattern for a given number of lines. For example, if number of lines entered are 4 then out is as under:

```
*
* *
* * *
* * * *
* * *
* *
*
```

6. Write a for loop program for printing following series of numbers on screen.

(a) 1, 3, 5, 7, 9, 11 (b) 1, 2, 4, 8, 16, 32, 64

(c) 10, 8, 6, 4, 2, 0 (d) -10, -8, -6, -4, -2, 0

7. Write a program to display ASCII value of uppercase and lowercase letters.
8. Write a program to count and display the number of characters entered by user until user enters 'Z';