Leaning Objective:

In this Module you will be learning the following:

- While loop
- For loop
- Do while loop

Introduction:

Loops are very basic and very useful programming facility that facilitates programmer to execute any block of code repeatedly and can be controlled as per conditions added by programmer. It saves writing code several times for same task.

Material:

C Language provides us different kind of looping statements that are:

- for loop
- while loop
- do-while loop

for Loop:

The syntax of for loop is:

```
for (initializationStatement; testExpression; updateStatement) {
        // codes
}
```

How for loop works?

The initialization statement is executed only once. Then, the test expression is evaluated. If the test expression is false (0), for loop is terminated. But if the test expression is true (nonzero), codes inside the body of for loop is executed and the update expression is updated. This process repeats until the test expression is false.

The for loop is commonly used when the number of iterations is known.

for loop Flowchart

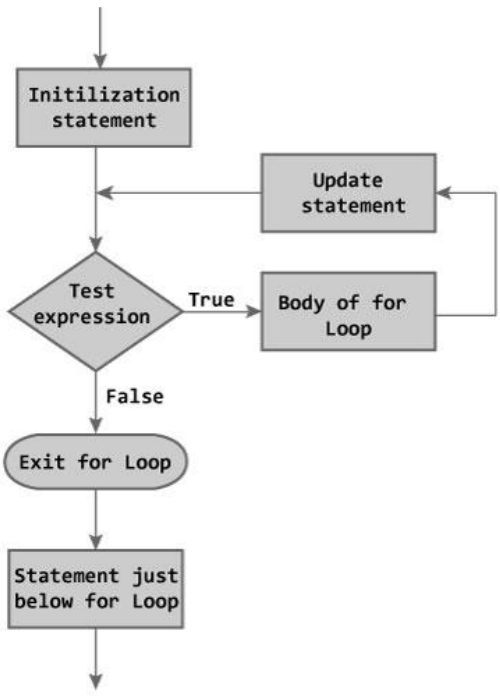


Figure: Flowchart of for Loop

Example: for loop

```
// Program to calculate the sum of first n natural numbers
// Positive integers 1,2,3...n are known as natural numbers
#include <stdio.h>
int main(){
   int num, count, sum = 0;
   printf("Enter a positive integer: ");
   scanf("%d", &num);
   // for loop terminates when n is less than count
   for(count = 1; count <= num; ++count)  {
      sum += count;
   }
   printf("Sum = %d", sum);
   return 0;
}
```

Output

```
Enter a positive integer: 10
Sum = 55
```

The value entered by the user is stored in variable num. Suppose, the user entered 10.

The count is initialized to 1 and the test expression is evaluated. Since, the test expression count <= num (1 less than or equal to 10) is true, the body of for loop is executed and the value of sum will equal to 1.

Then, the update statement ++count is executed and count will equal to 2. Again, the test expression is evaluated. Since, 2 is also less than 10, the test expression is evaluated to true and the body of for loop is executed. Now, the sum will equal 3.

This process goes on and the sum is calculated until the count reaches 11.

When the count is 11, the test expression is evaluated to 0 (false) as 11 is not less than or equal to 10. Therefore, the loop terminates and next, the total sum is printed.

while loop

The syntax of a while loop is:

```
while (testExpression) {
   //codes
}
```

where, testExpression checks the condition is true or false before each loop.

How while loop works?

The while loop evaluates the test expression.

If the test expression is true (nonzero), codes inside the body of while loop is evaluated. The test expression is evaluated again. The process goes on until the test expression is false.

When the test expression is false, the while loop is terminated.

Flowchart of while loop

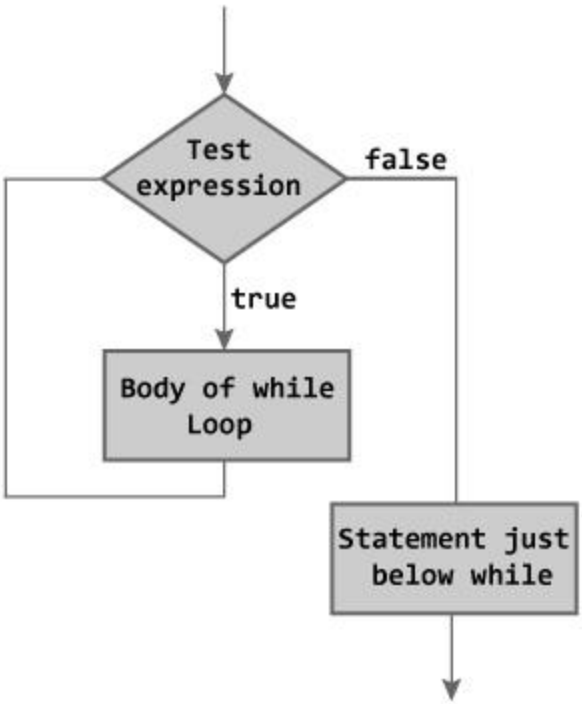


Figure: Flowchart of while Loop

Example #1: while loop

```
// Program to find factorial of a number
// For a positive integer n, factorial = 1*2*3...n
#include <stdio.h>
int main() {
   int number;
   long long factorial;
   printf("Enter an integer: ");
   scanf("%d",&number);
   factorial = 1;
   // loop terminates when number is less than or equal to 0
   while (number > 0)    {
      factorial *= number;  // factorial = factorial*number;
      --number;
   }
   printf("Factorial= %lld", factorial);
   return 0;
}
```

Output

```
Enter an integer: 5
Factorial = 120
```

do...while loop

The do..while loop is similar to the while loop with one important difference. The body of do...while loop is executed once, before checking the test expression. Hence, the do...while loop is executed at least once.

do...while loop Syntax

```
do {
  // codes
} while (testExpression);
```

How do...while loop works?

The code block (loop body) inside the braces is executed once. Then, the test expression is evaluated. If the test expression is true, the loop body is executed again. This process goes on until the test expression is evaluated to 0 (false). When the test expression is false (nonzero), the do...while loop is terminated.

flowchart of do...while loop

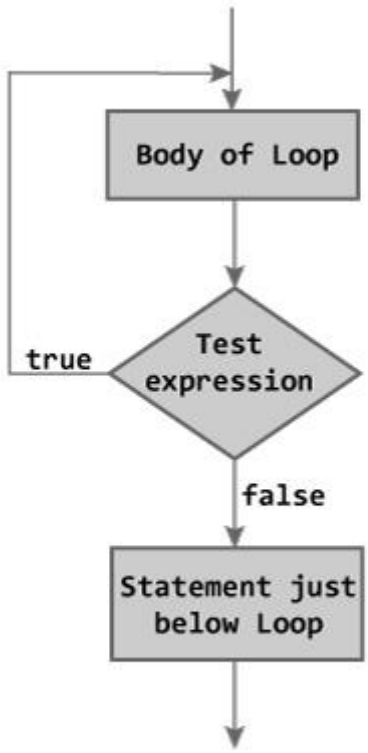


Figure: Flowchart of do...while Loop

Example #2: do...while loop

// Program to add numbers until user enters zero

```
#include <stdio.h>
int main(){
   double number, sum = 0;
   do {                                  // loop body is executed at least once
      printf("Enter a number: ");
      scanf("%lf", &number);
      sum += number;
   }  while(number != 0.0);
   printf("Sum = %.2lf",sum);
 }
```

Output

```
Enter a number: 1.5
Enter a number: 2.4
Enter a number: -3.4
Enter a number: 4.2
Enter a number: 0
Sum = 4.70
```

break and continue

It is sometimes desirable to skip some statements inside the loop or terminate the loop immediately without checking the test expression. In such cases, break and continue statements are used.

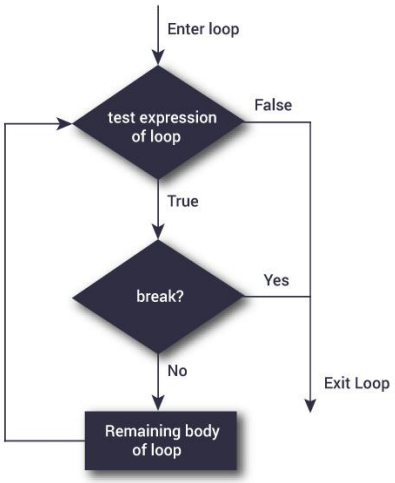
break Statement

The break statement terminates the loop (for, while and do...while loop) immediately when it is encountered. The break statement is used with decision making statement such as if...else.

Syntax of break statement

```
break;
```

Flowchart of break statement



How break statement works?

```
while (test Expression) {
    // codes
    if (condition for break)
    {
        break;
    }
    // codes
}
```

```
for (init, condition, update){
    // codes
    if (condition for break)
    {
        break;
    }
    // codes
}
```

Example #1: break statement

// Program to calculate the sum of maximum of 10 numbers

// Calculates sum until user enters positive number

```
# include <stdio.h>
int main(){
   int i;
   double number, sum = 0.0;
   for(i=1; i <= 10; ++i)   {
      printf("Enter a n%d: ",i);
      scanf("%lf",&number);
      // If user enters negative number, loop is terminated
      if(number < 0.0) {
         break;
      }
      sum += number; // sum = sum + number;
   }
   printf("Sum = %.2lf",sum);
   return 0;
}
```

Output

```
Enter a n1: 2.4
Enter a n2: 4.5
Enter a n3: 3.4
Enter a n4: -3
Sum = 10.30
```

This program calculates the sum of maximum of 10 numbers. It's because, when the user enters negative number, the break statement is executed and loop is terminated.

In C programming, break statement is also used with switch...case statement.

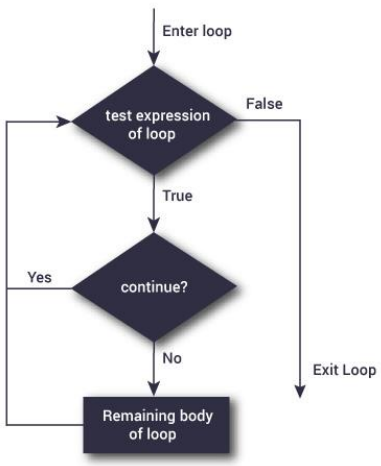
continue Statement

The continue statement skips some statements inside the loop. The continue statement is used with decision making statement such as if...else.

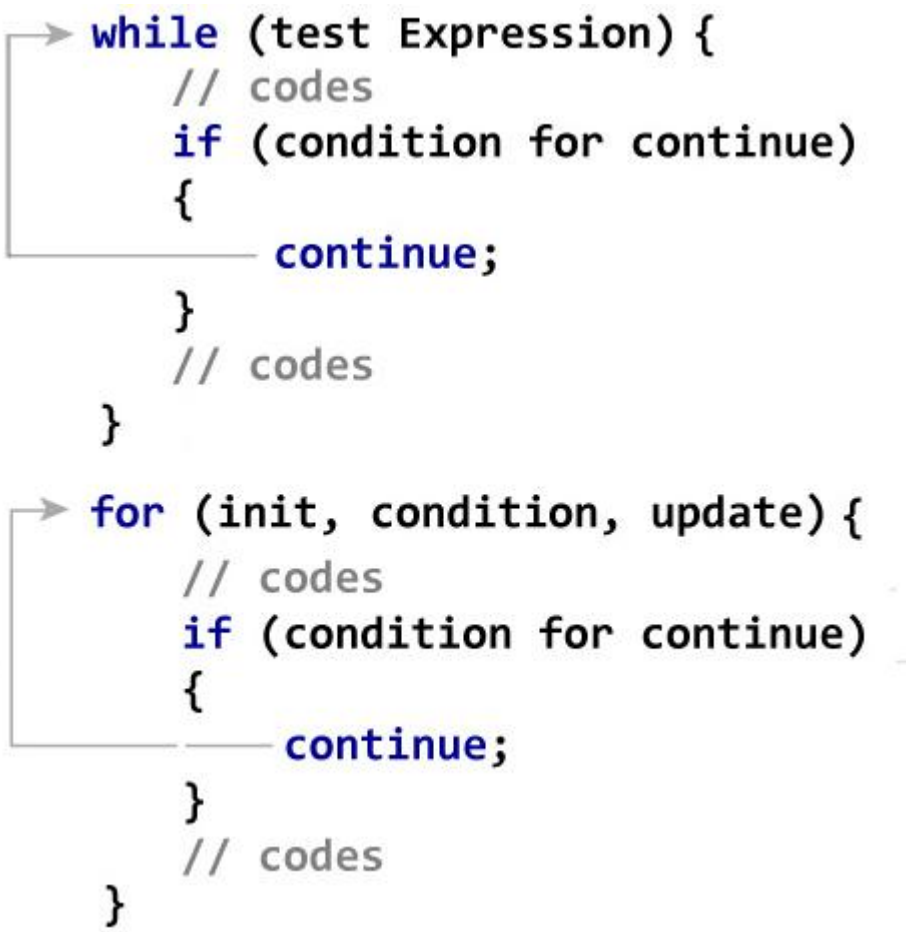
Syntax of continue Statement

```
continue;
```

Flowchart of continue Statement



How continue statement works?



Example #2: continue statement

// Program to calculate sum of maximum of 10 numbers

// Negative numbers are skipped from calculation

```
# include <stdio.h>
void main(){
   int i;
   double number, sum = 0.0;
   for(i=1; i <= 5; ++i)    {
      printf("Enter a n%d: ",i);
      scanf("%lf",&number);
      if(number < 0.0) {    // If user enters negative number, loop is terminated
         continue;
      }
      sum += number; // sum = sum + number;
   }
   printf("Sum = %.2lf",sum);
}
```

Output

```
Enter a n1: 1.1
Enter a n2: 2.2
Enter a n3: 5.5
Enter a n4: 4.4
Enter a n5: -3.4
Sum = 13.20
```

Problem Set:

1. Write a C program to print all natural numbers from 1 to n.
2. Write a C program to print all even numbers between 1 to 100.
3. Write a C program to print table of any number.
4. Write a C program to enter any number and calculate sum of all natural numbers between 1 to n.
5. Write a C program to find first and last digit of any number.
6. Write a C program to calculate sum and count of digits of any number.
7. Write a C program to enter any number and print its reverse.
8. Write a C program to enter any number and check whether the number is palindrome or not.
9. Write a C program to find HCF (GCD) and LCM of two numbers.
10. Write a C program to check whether a number is Prime number or not.
11. Write a C program to check whether a number is Armstrong number or not.
12. Write a C program to check whether a number is Perfect number or not.
13. Write a C program to check whether a number is Strong number or not.
14. Write a C program to print all Prime numbers between 1 to n and find their sum also.
15. Write a C program to print all Armstrong numbers between 1 to n.
16. Write a C program to print all Perfect numbers between 1 to n.
17. Write a C program to print all Strong numbers between 1 to n.
18. Write a C program to enter any number and print its prime factors.
19. Write a C program to print Fibonacci series up to n terms.
20. Write a C program to convert Decimal to Binary number system.
21. Write a C program to convert one number system to another number system.
22. Write a C program to print the following patterns

```
1          1          1          1          1          1
2 3        2 4        1 2 1      2 1        1 2        2 1
4 5 6      1 3 5      1 2 3 2 1  1 2 3      1 2 3      3 2 1
7 8 9 10   2 4 6 8    1 2 3 4 3 2 1  4 3 2 1    1 2 3 4 4 3 2 1
```

```
1
121
12321
1234321
123454321
1234321
12321
121
1
```

23. Write a program in C to find the sum of the following series?
    a) 1 + 1/2 + 1/3 + 1/4 + 1/5 ... 1/n
    b) 1/2 - 2/3 + 3/4 - 4/5 + 5/6 - ..... n
    c) (1) + (1+2) + (1+2+3) + (1+2+3+4) + ... + (1+2+3+4+...+n)
    d) x - ((x^3)/3!) + ((x^5)/5!) –……..
    e) 1 + (x/1!) + ((x^2)/2!) + ((x^3)/3!)+…….