Module 3

INPUT AND OUTPUT IN C

C programming has several in-built library functions to perform input and output tasks.

Two commonly used functions for I/O (Input/Output) are printf() and scanf().

The scanf() function reads formatted input from standard input (keyboard) whereas the printf() function sends formatted output to the standard output (screen).

Example #1: C Output

```
#include <stdio.h>      //This is needed to run printf() function.
int main()
{
printf("C Programming");  //displays the content inside
                          //quotation
return 0;
}
```

Output

```
C Programming
```

Working of this program

- All valid C program must contain the main() function. The code execution begins from the start of main() function.
- The printf() is a library function to send formatted output to the screen. The printf() function is declared in "stdio.h" header file.
- Here, stdio.h is a header file (standard input output header file) and #include is a preprocessor directive to paste the code from the header file when necessary. When the compiler encounters printf() function and doesn't find stdio.h header file, compiler shows error.
- The return 0; statement is the "Exit status" of the program. In simple terms, program ends.

Example #2: C Integer Output

```
#include <stdio.h>
int main()
{
   int testInteger = 5;
   printf("Number = %d", testInteger);
   return 0;
}
```

Output

```
Number = 5
```

Inside the quotation of printf() function, there is a format string "%d" (for integer). If the format string matches the argument (**testInteger** in this case), it is displayed on the screen.

Example #3: C Integer Input/Output

```
#include <stdio.h>
int main()
{
   int testInteger;
   printf("Enter an integer: ");
   scanf("%d",&testInteger);
   printf("Number = %d",testInteger);
   return 0;
}
```

Output

```
Enter an integer: 4
Number = 4
```

The scanf() function reads formatted input from the keyboard. When user enters an integer, it is stored in variable **testInteger**. Note the '&' sign before **testInteger**; **&testInteger** gets the address of **testInteger** and the value is stored in that address.

Example #3: C Floats Input/Output

```
#include <stdio.h>
int main()
{
   float f;
   printf("Enter a number: ");
// %f format string is used in case of floats
   scanf("%f",&f);
   printf("Value = %f", f);
   return 0;
}
```

Output

```
Enter a number: 23.45
Value = 23.450000
```

The format string "%f" is used to read and display formatted in case of floats.

Example #4: C Character I/O

```
#include <stdio.h>
int main()
{
   char chr;
   printf("Enter a character: ");
   scanf("%c",&chr);
   printf("You entered %c.",chr);
   return 0;
}
```

Output

```
Enter a character: g
You entered g.
```

Format string %c is used in case of character types.

Little bit on ASCII code

When a character is entered in the above program, the character itself is not stored. Instead, a numeric value(ASCII value) is stored.
And when we displayed that value using "%c" text format, the entered character is displayed.

Example #5: C ASCII Code

```
#include <stdio.h>
int main()
{
   char chr;
   printf("Enter a character: ");
   scanf("%c",&chr);

   // When %c text format is used, character is displayed in case
of character types
   printf("You entered %c.\n",chr);

   // When %d text format is used, integer is displayed in case of
character types
   printf("ASCII value of %c is %d.", chr, chr);
   return 0;
}
```

Output

```
Enter a character: g
You entered g.
ASCII value of g is 103.
```

The ASCII value of character 'g' is 103. When, 'g' is entered, 103 is stored in variable var1 instead of g.

You can display a character if you know ASCII code of that character. This is shown by following example.

Example #6: C ASCII Code

```
#include <stdio.h>
int main()
{
   int chr = 69;
   printf("Character having ASCII value 69 is %c.",chr);
   return 0;
}
```

Output

```
Character having ASCII value 69 is E.
```

More on Input/Output of floats and Integers

Integer and floats can be displayed in different formats in C programming.

Example #7: I/O of Floats and Integers

```
#include <stdio.h>
int main()
{

   int integer = 9876;
   float decimal = 987.6543;

   //  Prints the number right justified within 6 columns
   printf("4 digit integer right justified to 6 column: %6d\n",
integer);

   // Tries to print number right justified to 3 digits but the
number is not right adjusted because there are only 4 numbers
   printf("4 digit integer right justified to 3 column: %3d\n",
integer);

   // Rounds to two digit places
```

```
  printf("Floating point number rounded to 2 digits: %.2f\n",decimal);

  // Rounds to 0 digit places
  printf("Floating point number rounded to 0 digits: %.f\n",987.6543);

  // Prints the number in exponential notation(scientific notation)
  printf("Floating point number in exponential form: %e\n",987.6543);
  return 0;
}
```

Output

```
4 digit integer right justified to 6 column:   9876
4 digit integer right justified to 3 column: 9876
Floating point number rounded to 2 digits: 987.65
Floating point number rounded to 0 digits: 988
Floating point number in exponential form: 9.876543e+02
```

DECISION STATEMENTS

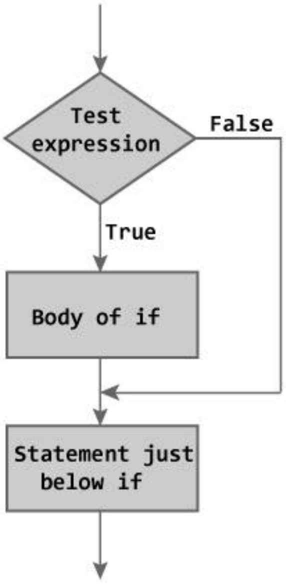
Decision making is used to specify the order in which statements are executed.

Simple if statement

```
    if (testExpression)
    {
      // statements
    }
```

The if statement evaluates the test expression inside the parenthesis.
If the test expression is evaluated to true (nonzero), statements inside the body of if is executed.
If the test expression is evaluated to false (0), statements inside the body of if is skipped from execution.

Flowchart of if statement



Example #1: C if statement

```
// Program to display a number if user enters negative number
// If user enters positive number, that number won't be displayed

#include <stdio.h>
int main()
{
   int number;

   printf("Enter an integer: ");
   scanf("%d", &number);

   // Test expression is true if number is less than 0
   if (number < 0)
   {
      printf("You entered %d.\n", number);
   }

   printf("The if statement is easy.");

   return 0;
}
```

Output 1

```
Enter an integer: -2
You entered -2.
The if statement is easy.
```

When user enters -2, the test expression (number < 0) becomes true. Hence, You entered -2 is displayed on the screen.

Output 2

```
Enter an integer: 5
The if statement in C programming is easy.
```

When user enters 5, the test expression (number < 0) becomes false and the statement inside the body of if is skipped.

if...else statement

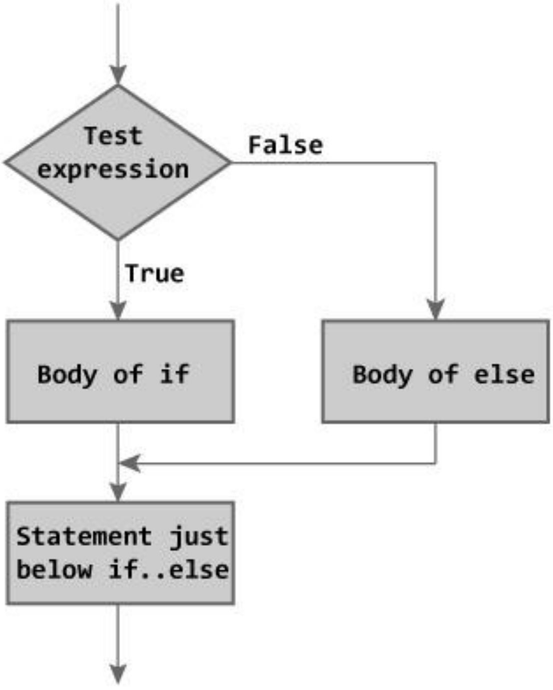
The if...else statement executes some code if the test expression is true (nonzero) and some other code if the test expression is false (0).

Syntax of if...else

```
if (testExpression) {
   // codes inside the body of if
}
else {
   // codes inside the body of else
}
```

If test expression is true, codes inside the body of if statement is executed and, codes inside the body of else statement is skipped. If test expression is false, codes inside the body of else statement is executed and, codes inside the body of if statement is skipped.

Flowchart of if...else statement



Example #2: C if...else statement

```
// Program to check whether an integer entered by the user is odd or even

#include <stdio.h>
int main()
{
   int number;
   printf("Enter an integer: ");
   scanf("%d",&number);

   // True if remainder is 0
   if( number%2 == 0 )
      printf("%d is an even integer.",number);
   else
      printf("%d is an odd integer.",number);
   return 0;
}
```

Output

```
Enter an integer: 7
7 is an odd integer.
```

When user enters 7, the test expression (number%2 == 0) is evaluated to false. Hence, the statement inside the body of else statement printf("%d is an odd integer"); is executed and the statement inside the body of if is skipped.

Nested if...else statement (if...elseif....else Statement)

The if...else statement executes two different codes depending upon whether the test expression is true or false. Sometimes, a choice has to be made from more than 2 possibilities.

The nested if...else statement allows you to check for multiple test expressions and execute different codes for more than two conditions.

Syntax of nested if...else statement.

```
if (testExpression1)
{
  // statements to be executed if testExpression1 is true
}
else if(testExpression2)
{
  // statements to be executed if testExpression1 is false
and testExpression2 is true
}
else if (testExpression 3)
{
  // statements to be executed if testExpression1 and
testExpression2 is false and testExpression3 is true
}
.
.
else
{
  // statements to be executed if all test expressions are
false
}
```

Example #3: C Nested if...else statement

```
// Program to relate two integers using =, > or <

#include <stdio.h>
int main()
{
   int number1, number2;
   printf("Enter two integers: ");
   scanf("%d %d", &number1, &number2);

   //checks if two integers are equal.
   if(number1 == number2)
   {
      printf("Result: %d = %d",number1,number2);
   }
```

```
    //checks if number1 is greater than number2.
    else if (number1 > number2)
    {
        printf("Result: %d > %d", number1, number2);
    }

    // if both test expression is false
    else
    {
        printf("Result: %d < %d",number1, number2);
    }

    return 0;
}
```

Output

```
Enter two integers: 12
23
Result: 12 < 23
```

switch...case

The if..else..if ladder allows you to execute a block code among many alternatives. If you are checking on the value of a single variable in if...else...if, it is better to use switch statement.

The switch statement is often faster than nested if...else (not always). Also, the syntax of switch statement is cleaner and easy to understand.

Syntax of switch...case

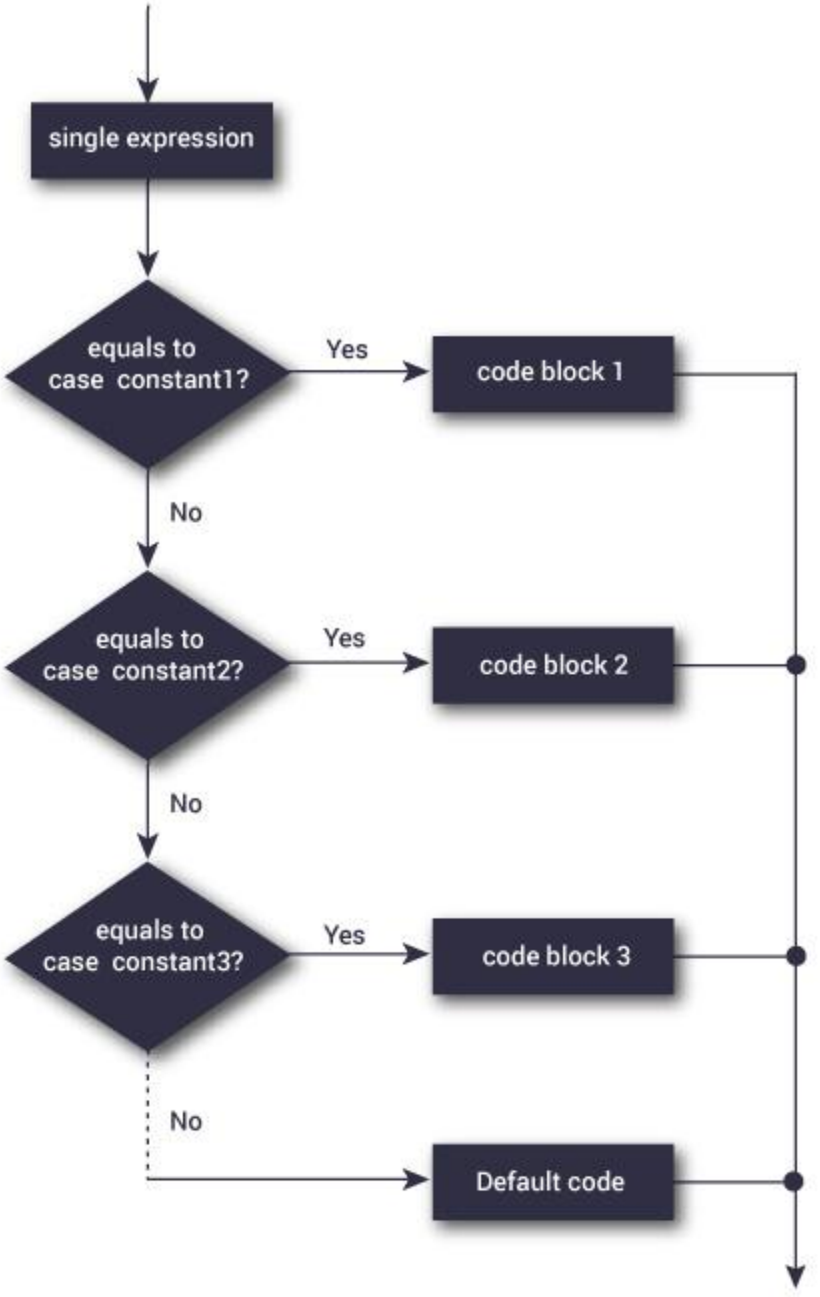
```
switch (n)
{
   case constant1:
      // code to be executed if n is equal to constant1;
      break;

   case constant2:
      // code to be executed if n is equal to constant2;
      break;
      .
      .
      .
   default:
      // code to be executed if n doesn't match any constant
}
```

When a case constant is found that matches the switch expression, control of the program passes to the block of code associated with that case.

In the above pseudo code, suppose the value of ***n*** is equal to ***constant2***. The compiler will execute the block of code associate with the case statement until the end of switch block, or until the break statement is encountered.

The **break** statement is used to prevent the code running into the next case.

switch Statement Flowchart



Example: switch Statement

```
// Program to create a simple calculator
// performs addition, subtraction, multiplication or division
depending the input from user

# include <stdio.h>

int main() {

   char operator;
   double firstNumber,secondNumber;

   printf("Enter an operator (+, -, *, /): ");
   scanf("%c", &operator);

   printf("Enter two operands: ");
   scanf("%lf %lf",&firstNumber, &secondNumber);

   switch(operator)
   {
      case '+':
         printf("%.1lf + %.1lf = %.1lf",firstNumber,
secondNumber, firstNumber+secondNumber);
         break;

      case '-':
         printf("%.1lf - %.1lf = %.1lf",firstNumber,
secondNumber, firstNumber-secondNumber);
         break;

      case '*':
         printf("%.1lf * %.1lf = %.1lf",firstNumber,
secondNumber, firstNumber*secondNumber);
         break;

      case '/':
         printf("%.1lf / %.1lf = %.1lf",firstNumber,
secondNumber, firstNumber/firstNumber);
```

```
            break;

        // operator is doesn't match any case constant (+, -, *, /)
        default:
            printf("Error! operator is not correct");
    }

    return 0;
}
```

Output

```
Enter an operator (+, -, *,): -
Enter two operands: 32.5
12.4
32.5 - 12.4 = 20.1
```

The - operator entered by the user is stored in **operator** variable. And, two operands 32.5 and 12.4 are stored in variables **firstNumber** and **secondNumber** respectively.

Then, control of the program jumps to
printf("%.1lf / %.1lf = %.1lf",firstNumber, secondNumber, firstNumber/firstNumber);

Finally, the **break** statement ends the switch statement.
If break statement is not used, all cases after the correct case is executed.

Loops

Loops are used in programming to repeat a specific block until some end condition is met. There are three loops in C programming:

1. for loop
2. while loop
3. do...while loop

for Loop

The syntax of for loop is:

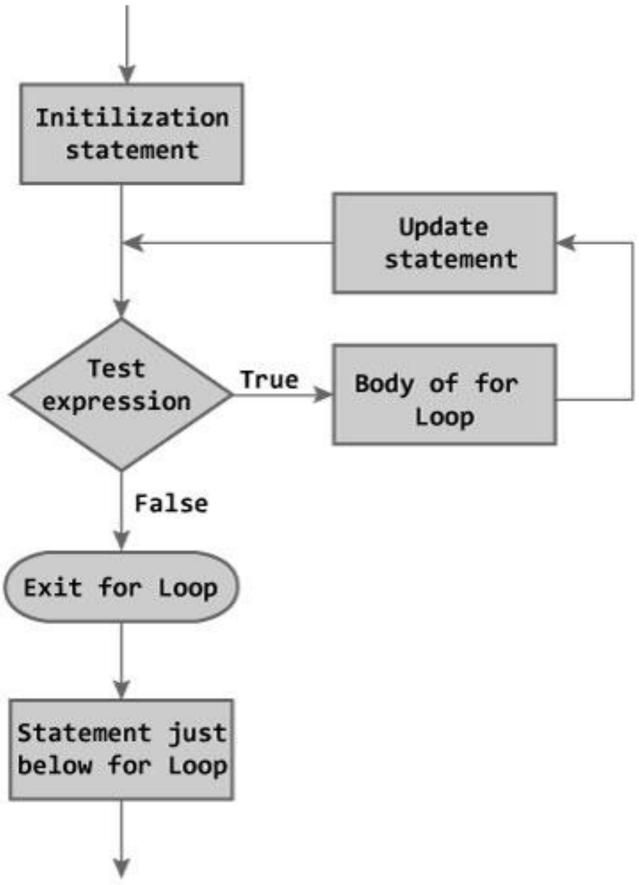
```
for (initializationStatement; testExpression; updateStatement)
{
    // codes
}
```

Working of for loop

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



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:

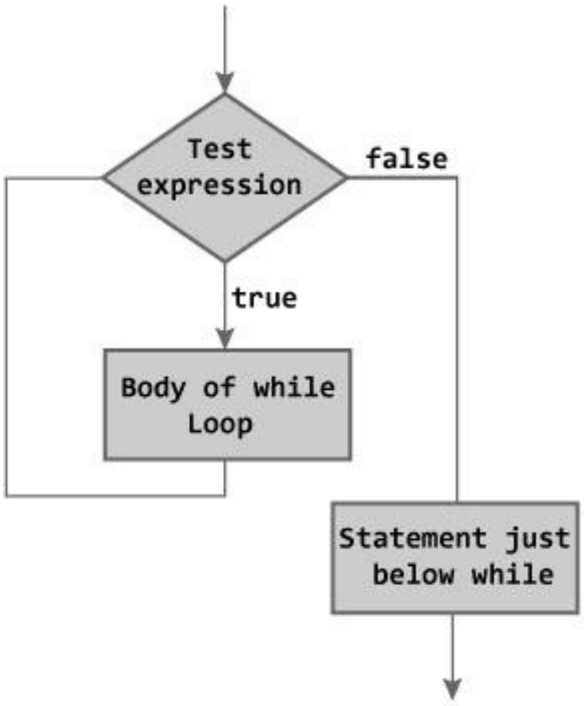
```
while (testExpression)
{
  //codes
}
```

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

Working of while loop

The while loop evaluates the test expression.
If the test expression is true (nonzero), codes inside the body of while loop are exectued. 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



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

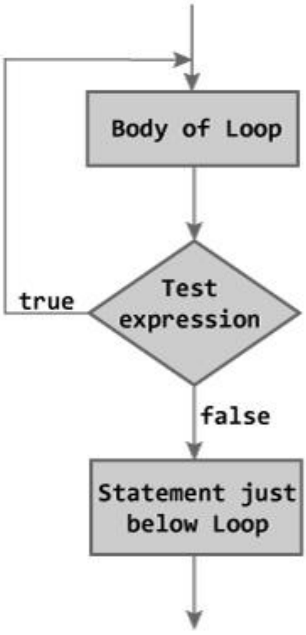
The do..while loop is similar to the while loop with one important difference. The body of do...while loop is executed once, before checking the test expression. Hence, the do...while loop is executed at least once.

do...while loop Syntax

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

Working of do...while loop

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.



Example #2: do...while loop

```
// Program to add numbers until user enters zero

#include <stdio.h>
int main()
{
   double number, sum = 0;

   // loop body is executed at least once
   do
   {
      printf("Enter a number: ");
      scanf("%lf", &number);
      sum += number;
   }
   while(number != 0.0);

   printf("Sum = %.2lf",sum);

   return 0;
}
```

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 Statement

It is sometimes desirable to skip some statements inside the loop or terminate the loop immediately without checking the test expression.

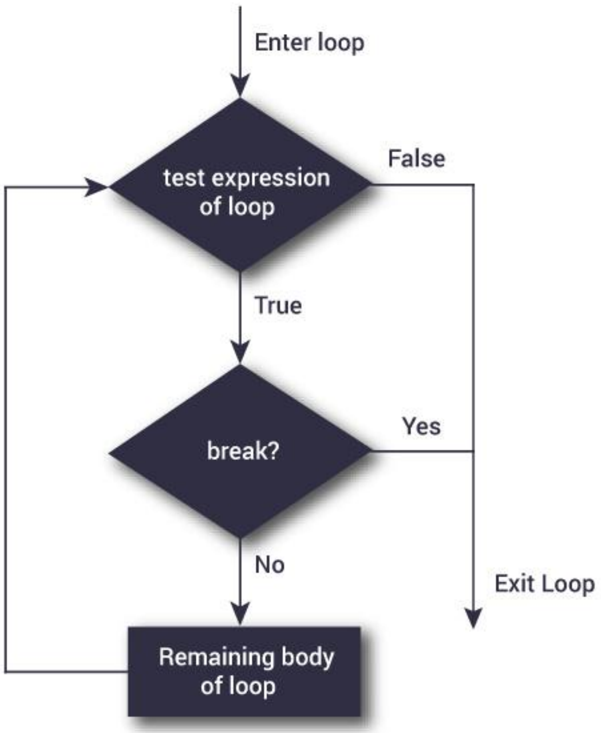
In such cases, break and continue statements are used.
The break statement terminates the loop (for, while and do...while loop) immediately when it is encountered. The break statement is used with decision making statement such as if...else.

Syntax of break statement

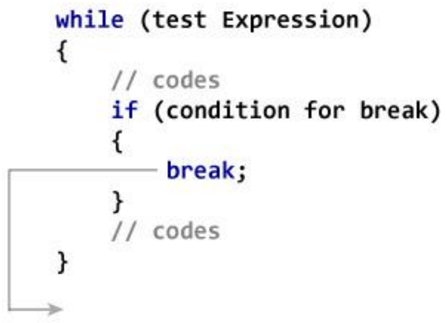
break;

The simple code above is the syntax for break statement.

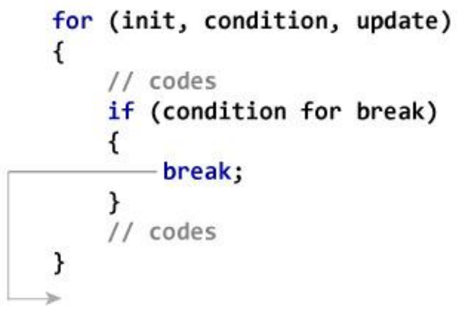
Flowchart of break statement



Working of break statement



```
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.

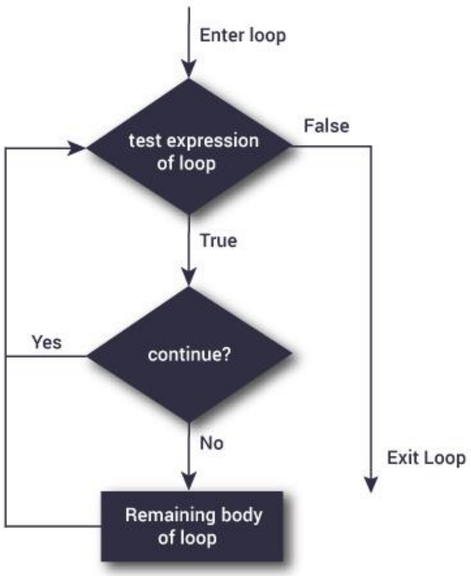
continue Statement

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

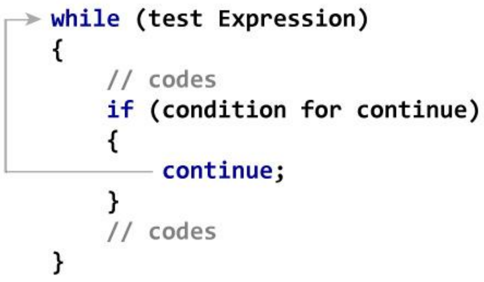
Syntax of continue Statement

continue;

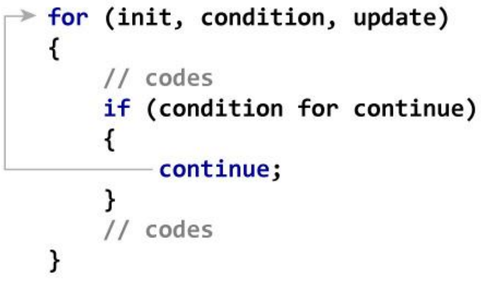
Flowchart of continue Statement



Working of continue statement



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



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

Example #2: continue statement

```
// Program to calculate sum of maximum of 10 numbers
// Negative numbers are skipped from calculation

# 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)
      {
         continue;
      }

      sum += number; // sum = sum + number;
   }

   printf("Sum = %.2lf",sum);

   return 0;
}
```

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
Enter a n6: -45.5
Enter a n7: 34.5
Enter a n8: -4.2
Enter a n9: -1000
Enter a n10: 12
Sum = 59.70
```

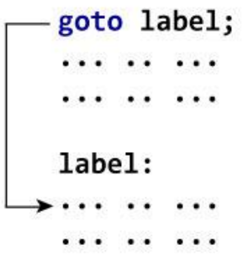
In the program, when the user enters positive number, the sum is calculated using sum += number; statement.
When the user enters negative number, the continue statement is executed and skips the negative number from calculation.

goto statement
The goto statement is used to alter the normal sequence of a C program.

Syntax of goto statement

```
    goto label;
    ... .. ...
    ... .. ...
    ... .. ...
    label:
    statement;
```

The label is an identifier. When goto statement is encountered, control of the program jumps to label: and starts executing the code.


```
goto label;
... .. ...
... .. ...

label:
... .. ...
... .. ...
```

Example: goto Statement

// Program to calculate the sum and average of maximum of 5 numbers
// If user enters negative number, the sum and average of previously entered positive number is displayed

```
# include <stdio.h>

int main()
{

   const int maxInput = 5;
   int i;
   double number, average, sum=0.0;

   for(i=1; i<=maxInput; ++i)
   {
      printf("%d. Enter a number: ", i);
      scanf("%lf",&number);

   // If user enters negative number, flow of program moves to
label jump
      if(number < 0.0)
         goto jump;

      sum += number; // sum = sum+number;
   }

   jump:

   average=sum/(i-1);
   printf("Sum = %.2f\n", sum);
   printf("Average = %.2f", average);

   return 0;
}
```

Output

```
1. Enter a number: 3
2. Enter a number: 4.3
3. Enter a number: 9.3
4. Enter a number: -2.9
Sum = 16.60
```

Reasons to avoid goto statement

The use of goto statement may lead to code that is buggy and hard to follow.

For example:

```
one:
for (i = 0; i < number; ++i)
{
   test += i;
   goto two;
}
two:
if (test > 5) {
  goto three;
}
... .. ...
```

Also, goto statement allows you to do bad stuff such as jump out of scope.

That being said, goto statement can be useful sometimes.
For example: to break from nested loops.

If you think the use of goto statement simplifies your program. By all means use it. The goal here is to create code that your fellow programmers can understand easily.