4.1 IF Statement

4.1.1 Simple if Statement

An **if** statement consists of a Boolean expression followed by one or more statements.

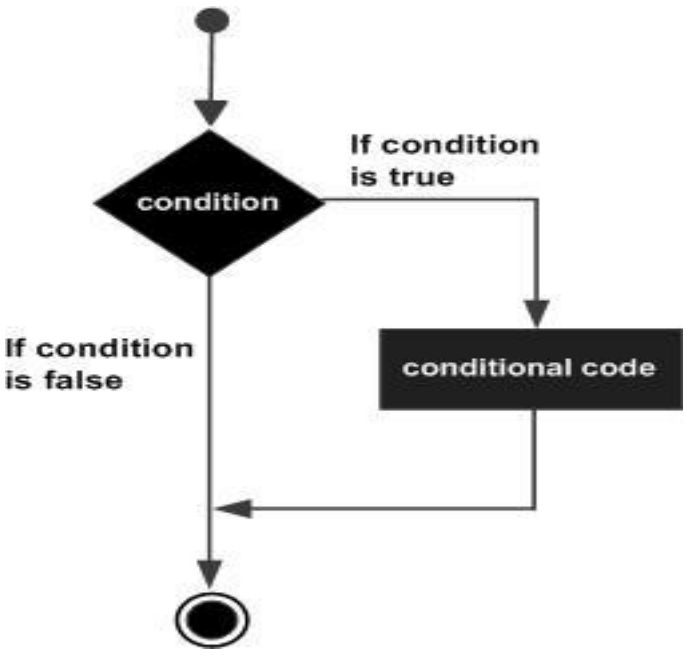
The syntax of an 'if' statement in C programming language is −

```
if(boolean_expression) {
  /* statement(s) will execute if the boolean expression is true */
}
```

If the Boolean expression evaluates to **true**, then the block of code inside the 'if' statement will be executed. If the Boolean expression evaluates to **false**, then the first set of code after the end of the 'if' statement (after the closing curly brace) will be executed.

C programming language assumes any **non-zero** and **non-null** values as **true** and if it is either **zero** or **null**, then it is assumed as **false** value.

Flow Diagram of if



Example

```
void main () {
            int a = 10;
            if( a < 20 ) {
            printf("a is less than 20\n" );
             }
  printf("value of a is : %d\n", a);
 }


Output :
a is less than 20
value of a is : 10
```

4.1.2 if…else statement

An **if statement** can be followed by an optional **else statement**, which executes when the Boolean expression is false.

The syntax of an 'if…else' statement in C programming language is −

```
if(boolean_expression) {
  /* statement(s) will execute if the boolean expression is true */
}
Else {
  /* statement(s) will execute if the boolean expression is false */
}
```

If the Boolean expression evaluates to **true**, then the block of code inside the 'if' statement will be executed. If the Boolean expression evaluates to **false**, then the block of code inside the 'else' statement will be executed.

Example

```
void main () {
          int a = 21;
          if( a < 20 ) {
                    printf("a is less than 20\n" );
           }
          Else {
                    printf("a is greater than or equal to 20\n" );

          }
  printf("value of a is : %d\n", a);
 }

Output :
a is greater than or equal to 20
value of a is : 10
```

4.1.3 Nested if statement

A nested if statements for used when we want to check for a secondary condition if the first condition executes as true. For this, we can have an if-else statement inside of another if-else statement.

The syntax of an 'if…else' statement in C programming language is −

```
          if(boolean_expression1) {
            /* statement(s) will execute if the boolean expression is true */
```

```
                if(boolean_expression2) {
                       /* statement(s) will execute if the boolean expression1 is true */
                }

        }
        Else {
          /* statement(s) will execute if the boolean expression1 is false */
        }
```

Example

```
void main () {

        int a = 11;

        if( a < 20 ) {

                if(a>10) {

                        printf("a is less than 20 and greater than 10 \n" );

                }

                Else {

                        printf("a is less than 10 \n" );

                }

         }

        Else {

                printf("a is greater than or equal to 20\n" );



        }

  printf("value of a is : %d\n", a);

 }


Output :

a is less than 20 and greater than 10

value of a is : 11
```

LOOPs

4.2. while loop

A **while** loop in C programming repeatedly executes a target statement as long as a given condition is true.

Syntax

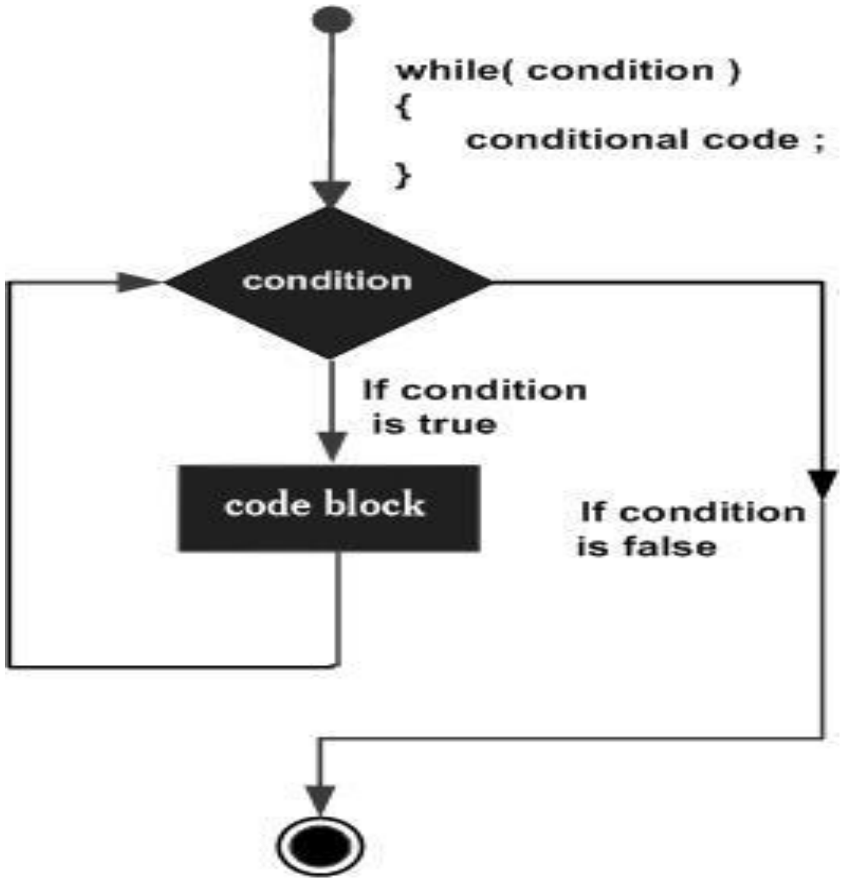
The syntax of a **while** loop in C programming language is −

```
while(condition) {
  statement(s);
}
```

Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any nonzero value. The loop iterates while the condition is true.

When the condition becomes false, the program control passes to the line immediately following the loop.

Flow Diagram



A while loop might not execute at all. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

Example

```
#include <stdio.h>

int main () {

  /* local variable definition */

   int a = 10;

   /* while loop execution */

   while( a < 20 ) {

      printf("value of a: %d\n", a);

      a++;

   }

   return 0;

}


OUTPUT :
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

4.2 do…while loop

Unlike **for** and **while** loops, which test the loop condition at the top of the loop, the **do...while** loop in C programming checks its condition at the bottom of the loop.

A **do...while** loop is similar to a while loop, except the fact that it is guaranteed to execute at least one time.

Syntax

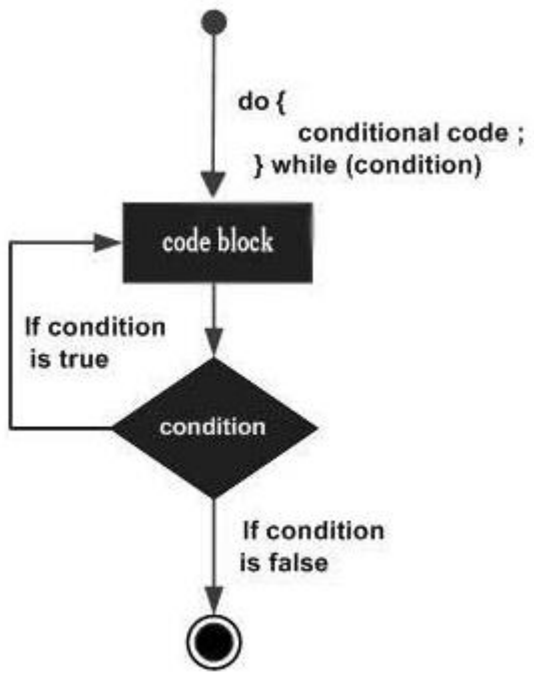
The syntax of a **do...while** loop in C programming language is −

```
do {
   statement(s);
} while( condition );
```

Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop executes once before the condition is tested.

If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop executes again. This process repeats until the given condition becomes false.

Flow Diagram



Example

```
#include <stdio.h>

int main () {

   /* local variable definition */

   int a = 10;

   /* do loop execution */

   do {

      printf("value of a: %d\n", a);

      a = a + 1;
   }while( a < 20 );

   return 0;

}
```

When the above code is compiled and executed, it produces the following result :

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

4.3 for loop

A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax

The syntax of a **for** loop in C programming language is :

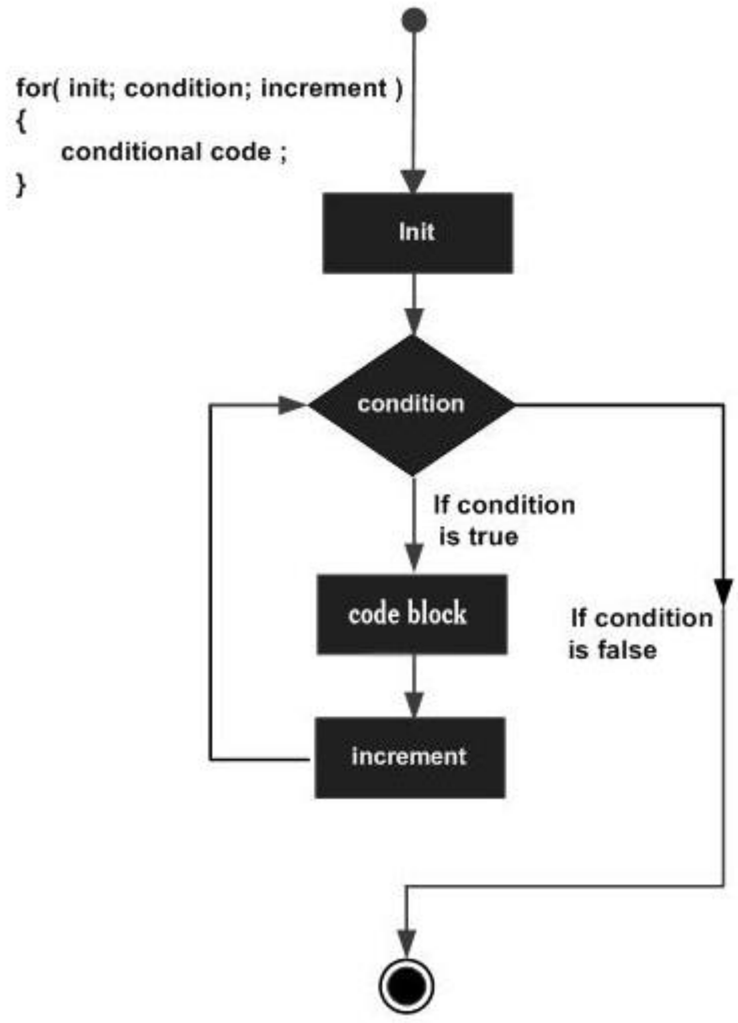
```
for ( init; condition; increment ) {
   statement(s);
}
```

Here is the flow of control in a 'for' loop −

- The **init** step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
- Next, the **condition** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and the flow of control jumps to the next statement just after the 'for' loop.
- After the body of the 'for' loop executes, the flow of control jumps back up to the **increment** statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.

- The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the 'for' loop terminates.

Flow Diagram



Example

```
#include <stdio.h>

 int main () {

   int a;

   /* for loop execution */

   for( a = 10; a < 20; a = a + 1 ){

      printf("value of a: %d\n", a);

   }
```

```
  return 0;

}
```

When the above code is compiled and executed, it produces the following result −

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

➔nested loops in C

C programming allows to use one loop inside another loop. The following section shows a few examples to illustrate the concept.

Syntax

The syntax for a **nested for loop** statement in C is as follows −

```
for ( init; condition; increment ) {

  for ( init; condition; increment ) {
    statement(s);
  }
  statement(s);
}
```

The syntax for a **nested while loop** statement in C programming language is as follows −

```
while(condition) {

  while(condition) {
    statement(s);
  }
  statement(s);
}
```

The syntax for a **nested do...while loop** statement in C programming language is as follows −

```
do {
  statement(s);

  do {
    statement(s);
```

```
   }while( condition );

}while( condition );
```

A final note on loop nesting is that you can put any type of loop inside any other type of loop. For example, a 'for' loop can be inside a 'while' loop or vice versa.

Example

The following program uses a nested for loop to find the prime numbers from 2 to 100 −

```
#include <stdio.h>


int main () {


   /* local variable definition */
   int i, j;


   for(i = 2; i<100; i++) {


      for(j = 2; j <= (i/j); j++)
      if(!(i%j)) break; // if factor found, not prime
      if(j > (i/j)) printf("%d is prime\n", i);
   }


   return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
```

```
23 is prime
29 is prime
31 is prime
37 is prime
41 is prime
43 is prime
47 is prime
53 is prime
59 is prime
61 is prime
67 is prime
71 is prime
73 is prime
79 is prime
83 is prime
89 is prime
97 is prime
```

4.5 break and continue statement

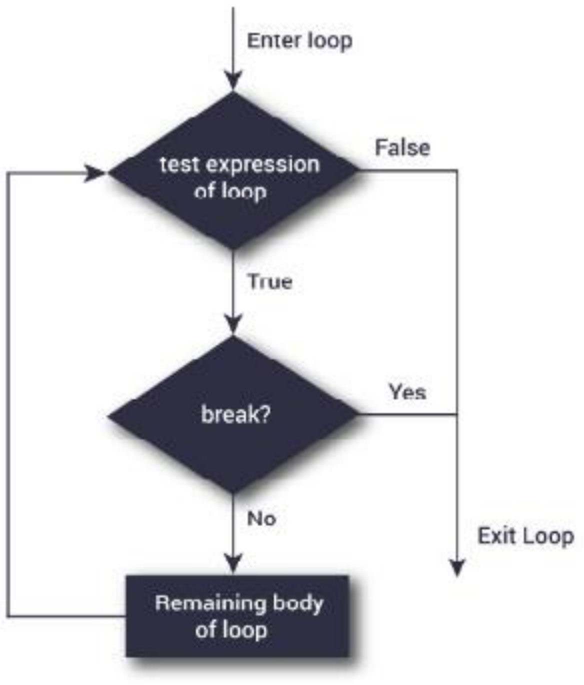
break Statement

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



Example of 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.

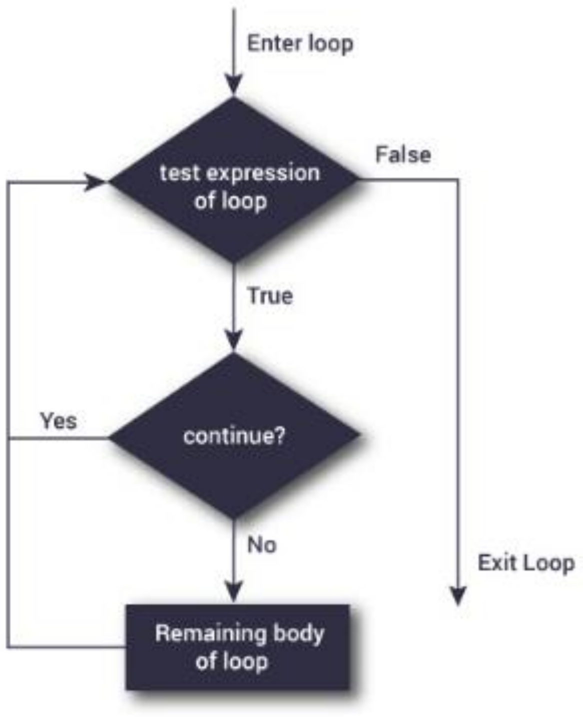
continue Statement

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



Example of 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.

4.6 switch statement

A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each **switch case**.

Syntax

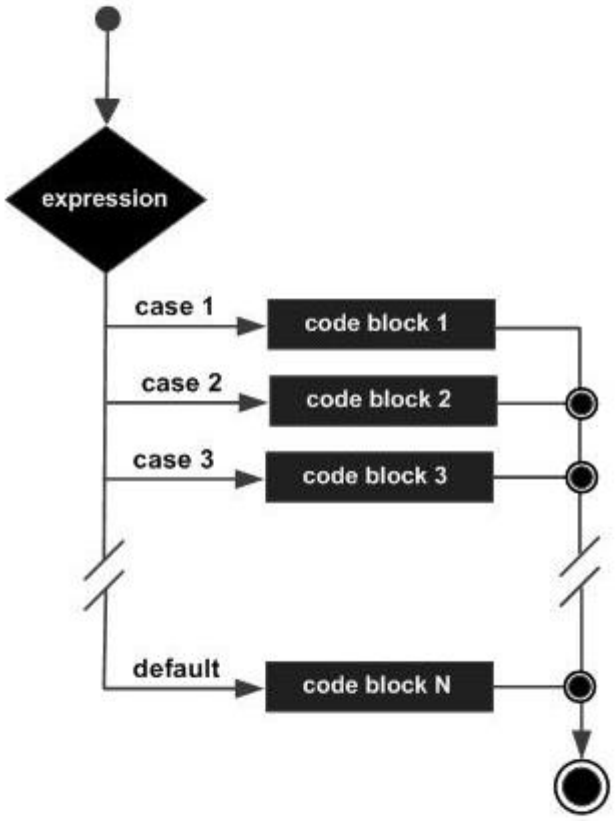
The syntax for a **switch** statement in C programming language is as follows −

```
switch(expression) {

  case constant-expression  :
     statement(s);
     break; /* optional */
  case constant-expression  :
     statement(s);
     break; /* optional */

  /* you can have any number of case statements */
  default : /* Optional */
  statement(s);
}
```

The following rules apply to a **switch** statement −

- The **expression** used in a **switch** statement must have an integral or enumerated type, or be of a class type in which the class has a single conversion function to an integral or enumerated type.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The **constant-expression** for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a **break** statement is reached.
- When a **break** statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a **break**. If no **break** appears, the flow of control will *fall through* to subsequent cases until a break is reached.
- A **switch** statement can have an optional **default** case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No **break** is needed in the default case.

Flow Diagram



Example

```
#include <stdio.h>

int main () {

   /* local variable definition */

   char grade = 'B';

   switch(grade) {

      case 'A' :

         printf("Excellent!\n" );

         break;

      case 'B' :

      case 'C' :

         printf("Well done\n" );
```

```
        break;

      case 'D' :

        printf("You passed\n" );

        break;

      case 'F' :

        printf("Better try again\n" );

        break;

      default :

        printf("Invalid grade\n" );

   }

   printf("Your grade is  %c\n", grade );

   return 0;

}
```

When the above code is compiled and executed, it produces the following result −

```
Well done
Your grade is B
```

==> nested switch statements

It is possible to have a switch as a part of the statement sequence of an outer switch. Even if the case constants of the inner and outer switch contain common values, no conflicts will arise.

Syntax

The syntax for a **nested switch** statement is as follows −

```
switch(ch1) {

   case 'A':
      printf("This A is part of outer switch" );

      switch(ch2) {
         case 'A':
            printf("This A is part of inner switch" );
            break;
         case 'B': /* case code */
      }

      break;
   case 'B': /* case code */
}
```

Example

```
#include <stdio.h>

int main () {

   /* local variable definition */

   int a = 100;

   int b = 200;

   switch(a) {


      case 100:

         printf("This is part of outer switch\n", a );

        switch(b) {

            case 200:

               printf("This is part of inner switch\n", a );

         }

   }

   printf("Exact value of a is : %d\n", a );

   printf("Exact value of b is : %d\n", b );

   return 0;

}
```

When the above code is compiled and executed, it produces the following result −

```
This is part of outer switch
This is part of inner switch
Exact value of a is : 100
Exact value of b is : 200
```