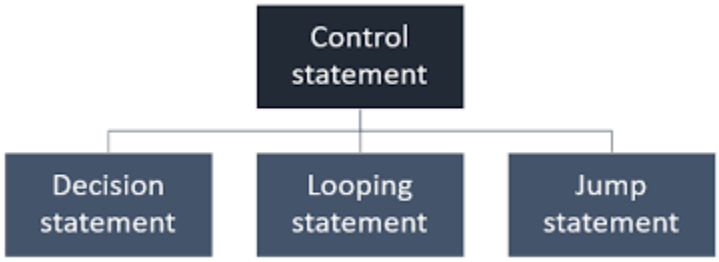
PROBLEM SOLVING USING C :MODULE 3

Unit III [4T + 10L]
Data input output functions - Simple C programs – Flow of Control - Decision making with IF statement, Simple IF statement, If-else statement, Nesting of If-else and else-if Ladder, Switch statement, Conditional operator, goto statement. Looping - While loop, Do-While, and For Loops,
Nesting of loops, jumps in loop, skipping of loops.

Introduction to Control Statements in C

- In C, the control flows from one instruction to the next instruction until now in all programs.
- This control flow from one command to the next is called sequential control flow. Nonetheless, in most C programs the programmer may want to skip instructions or repeat a set of instructions repeatedly when writing logic.
- This can be referred to as sequential control flow. The declarations in C let programmers make such decisions which are called decision-making or control declarations



Decision Making with Simple IF- statement

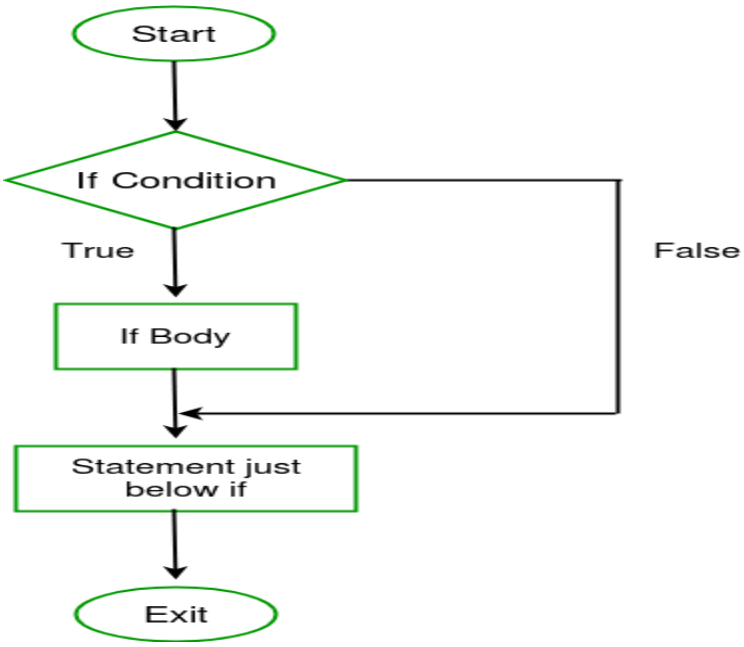
- if statement is the most simple decision making statement.
- It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.

Syntax:

```
if(condition)
{
   // Statements to execute if
   // condition is true
}
```

Here, **condition** after evaluation will be either true or false

Flow Chart:




```
// C program to illustrate If statement

#include <stdio.h>
```

```
void main() {
        int i = 10;
 if (i >15)
        {
        printf("i is less than 15");
        }
        printf("I am Not in if");
}
```

Output:

I am Not in if

Note : As the condition present in the if statement is false. So, the block below the if statement is not executed.

The IF ….Else…Statement

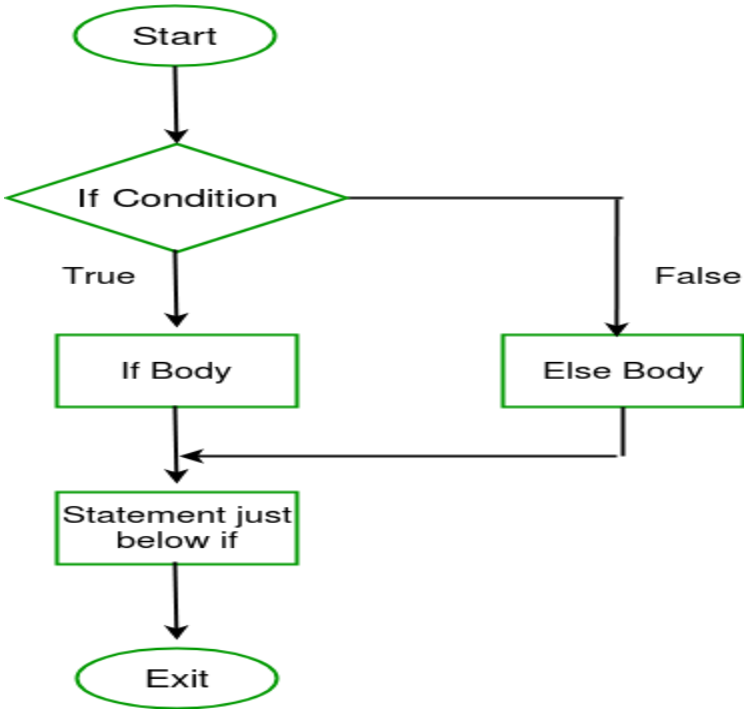
- The *if* statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't.
- But what if we want to do something else if the condition is false. Here comes the C *else* statement.
- We can use the *else* statement with *if* statement to execute a block of code when the condition is false.

Syntax:

```
if (condition)
{
    // Executes this block if
    // condition is true
}
else
{
    // Executes this block if
    // condition is false
}
```

The block of code following the *else* statement is executed as the condition present in the *if* statement is false.

Flowchart:




```
// C program to illustrate If –else statement
#include <stdio.h>

int main() {
   int i = 20;

   if (i < 15){
      printf("i is smaller than 15");}
   else
      printf("i is greater than 15");

}
```

Output i is greater than 15

Nested-if in C

- A nested if in C is an if statement that is the target of another if statement. Nested if statements means an if statement inside another if statement.
- Yes, C allows us to nested if statements within if statements, i.e, we can place an if statement inside another if statement.

Syntax:

```
if (condition1)
{
  // Executes when condition1 is true
  if (condition2)
  {
    // Executes when condition2 is true
  }
}
```

```
if (Condition1)
{
      if(Condition2)
      {
            Statement1;
       }
      else
      {
            Statement2;
      }
}
else
{
      if(Condition3)
      {
            Statement3;
      }
      else
      {
            Statement4;
      }
}
```

Flowchart

Example:

```
// C program to illustrate nested-if statement
    #include <stdio.h>
    int main() {
        int i = 10;
        if (i == 10)
        {
                // First if statement
                if (i < 15)
                printf("i is smaller than 15\n");
                // Nested - if statement
                // Will only be executed if statement above
                // is true
                if (i < 12)
```

```
                printf("i is smaller than 12 too\n");
        else
                printf("i is greater than 15");
    }   }
```

Output:

i is smaller than 15

i is smaller than 12 too

if-else-if ladder in C

- Here, a user can decide among multiple options. The C if statements are executed from the top down.
- As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the C else-if ladder is bypassed.
- If none of the conditions are true, then the final else statement will be executed.

Syntax:

```
if (condition)
   statement;
```

```
// C program to illustrate nested-if statement
#include <stdio.h>

int main() {
    int i = 20;

    if (i == 10)
        printf("i is 10");
    else if (i == 15)
```

```
    printf("i is 15");
  else if (i == 20)
    printf("i is 20");
  else
    printf("i is not present");
}
```

Output:

i is 20

Switch Statement in C

Switch case statements are a substitute for long if statements that compare a variable to several integral values

- The switch statement is a multiway branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression.
- Switch is a control statement that allows a value to change control of execution

Syntax:

```
switch (n)
{
    case 1: // code to be executed if n = 1;
        break;
    case 2: // code to be executed if n = 2;
        break;
    default: // code to be executed if n doesn't
match any cases
}
```

Important I

1. The expression provided in the switch should result in a **constant value** otherwise it would not be valid.

Valid expressions for switch:

```
// Constant expressions allowed

switch(1+2+23)

switch(1*2+3%4)


// Variable expression are allowed provided

// they are assigned with fixed values

switch(a*b+c*d)

switch(a+b+c)
```

2. Duplicate case values are not allowed.

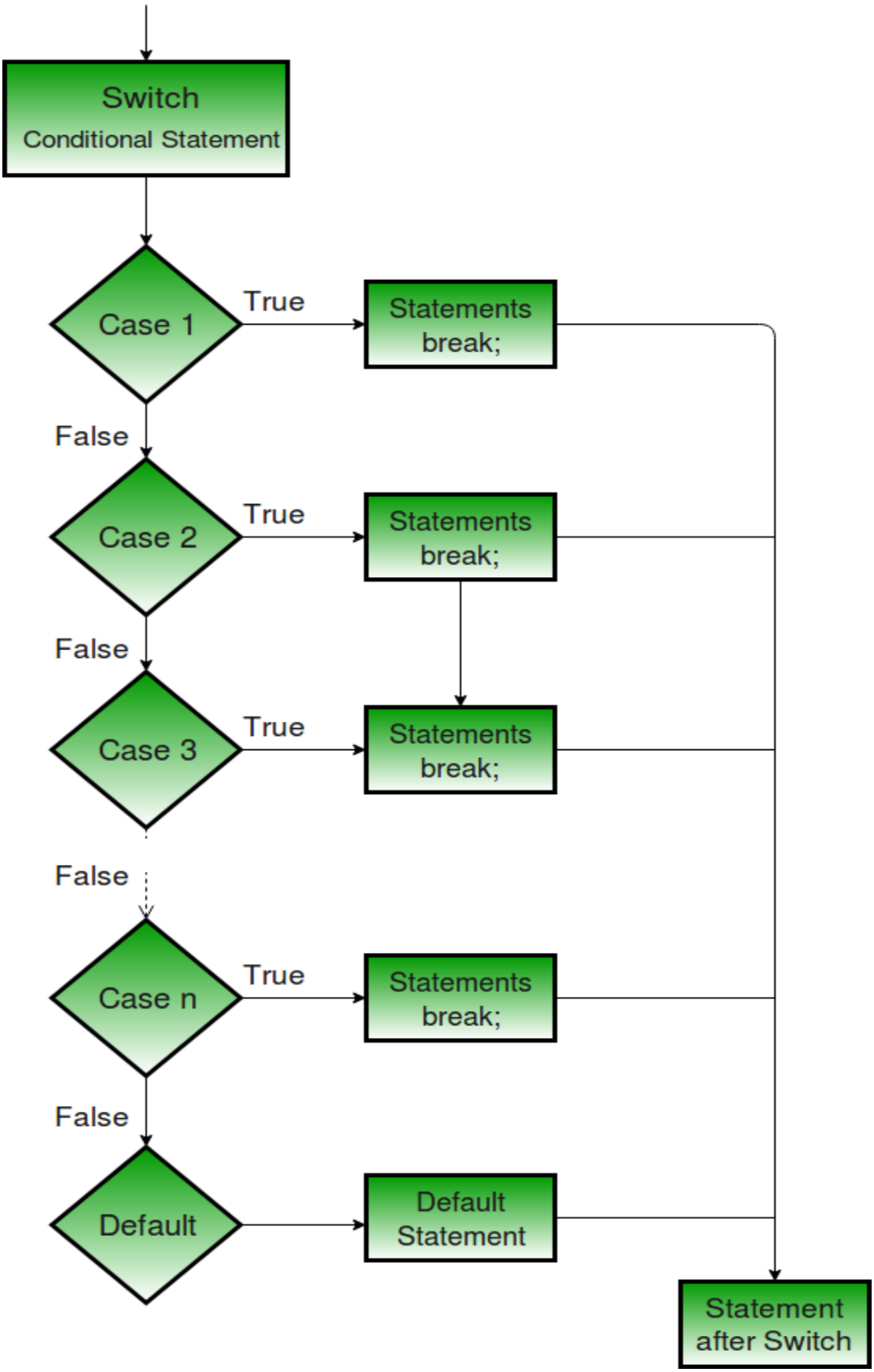
3. The default statement is optional.Even if the switch case statement do not have a default statement,
it would run without any problem.

4. The break statement is used inside the switch to terminate a statement sequence. When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.

5. The break statement is optional. If omitted, execution will continue on into the next case. The flow of control will fall through to subsequent cases until a break is reached.

6. Nesting of switch statements are allowed, which means you can have switch statements inside another switch. However nested switch statements should be avoided as it makes program more complex and less readable.

Flowchart:


Switch
Conditional Statement
Case 1
True
Statements
break;
False
Case 2
True
Statements
break;
False
Case 3
True
Statements
break;
False
Case n
True
Statements
break;
False
Default
Default
Statement
Statement
after Switch

Example:

```
// Following is a simple C program
// to demonstrate syntax of switch.
#include <stdio.h>
int main()
{
  int x ;
 printf(“enter a choice”);
 scanf(“%d”,&x);
  switch (x)
   {
      case 1: printf("Choice is 1");
              break;
      case 2: printf("Choice is 2");
               break;
      case 3: printf("Choice is 3");
              break;
      default: printf("Choice other than 1, 2 and 3");
               break;
   }
```

```
  return 0;
}
```

Output:

Enter

Choice is 2

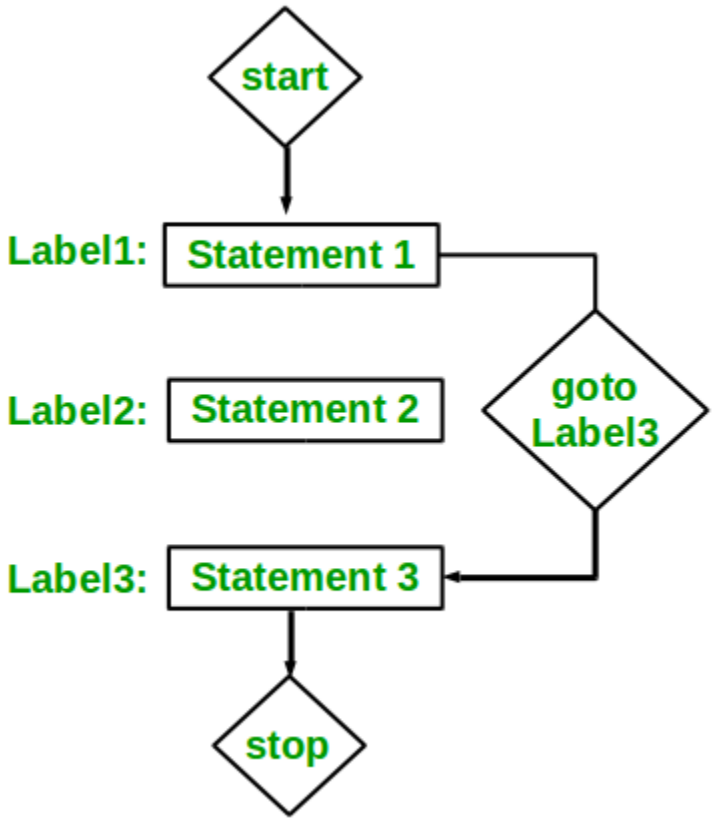
goto statement in C

The goto statement is a jump statement which is sometimes also referred to as unconditional jump statement. The goto statement can be used to jump from anywhere to anywhere within a function.

The goto statement is a jump statement which is sometimes also referred to as unconditional jump statement. The goto statement can be used to jump from anywhere to anywhere within a function.

```
Syntax1       |   Syntax2
----------------------------
goto label;   |    label:
.             |    .
.             |    .
.             |    .
label:        |    goto label;
```

In the above syntax, the first line tells the compiler to go to or jump to the statement marked as a label. Here label is a user-defined identifier which indicates the target statement. The statement immediately followed after 'label:' is the destination statement. The 'label:' can also appear before the 'goto label;' statement in the above syntax.



Below are some examples on how to use goto statement:
Examples:

```
#include<stdio.h>
 int main()
 {
   int n, i=1;

   printf("Enter a number: ");
   scanf("%d",&n);

   start:
   printf("%d\t",i);
   i++;
   if(i<n) goto start;
```

```
    return 0;


}
```

Disadvantages of using goto statement:

- The use of goto statement is highly discouraged as it makes the program logic very complex.
- use of goto makes the task of analyzing and verifying the correctness of programs (particularly those involving loops) very difficult.
- Use of goto can be simply avoided using break and continue statements.

LOOPING WITH C

- Looping Statements in C execute the sequence of statements many times until the stated condition becomes false.
- A loop in C consists of two parts, a body of a loop and a control statement.
- The control statement is a combination of some conditions that direct the body of the loop to execute until the specified condition becomesfalse.
- The purpose of the C loop is to repeat the same code a number of times.

Types of Loops in C

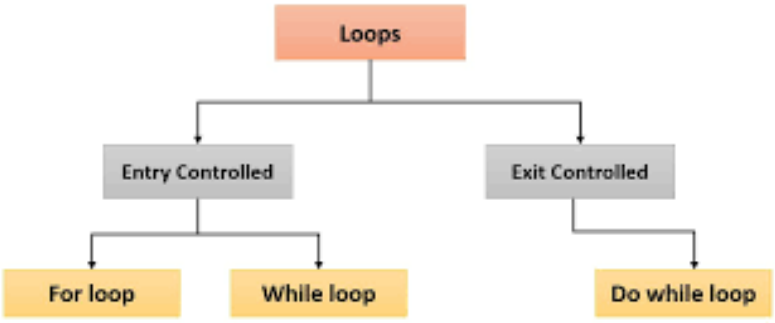
Depending upon the position of a control statement in a program, looping in C is classified into two types:

1. Entry controlled loop

2. Exit controlled loop

In an entry control loop in C, a condition is checked before executing the body of a loop. It is also called as a pre-checking loop.

In an exit controlled loop, a condition is checked after executing the body of a loop. It is also called as a post-checking loop.



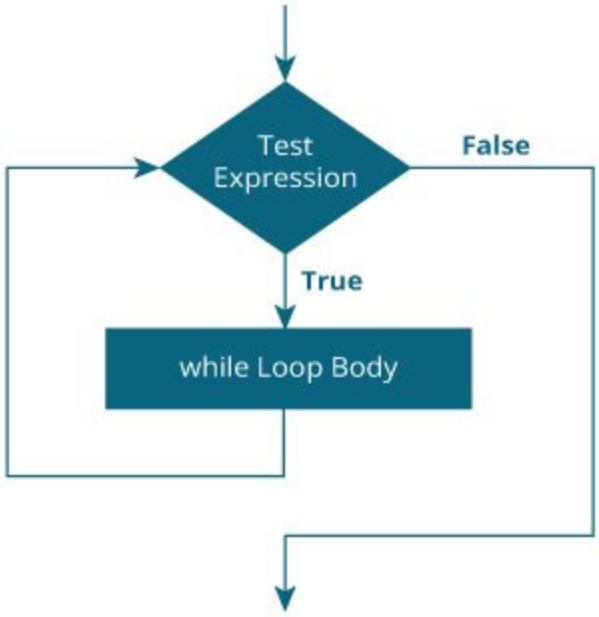
While Loop in C

- A while loop is the most straightforward looping structure.
- It is an entry-controlled loop. In while loop, a condition is evaluated before processing a body of the loop.
- If a condition is true then and only then the body of a loop is executed.
- After the body of a loop is executed then control again goes back at the beginning, and the condition is checked if it is true, the same process is executed until the condition becomes false.
- Once the condition becomes false, the control goes out of the loop.

Syntax of While Loop in C:

```
while (condition) {
        statements;
}
```

Flow chart



Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int num=1;   //initializing the variable
        while(num<=10)    //while loop with condition
        {
                printf("%d\n",num);
                num++;                  //incrementing operation
        }
}
```

Output:

1

2

3

4

5

6

7

8

9

10

Do-While loop in C

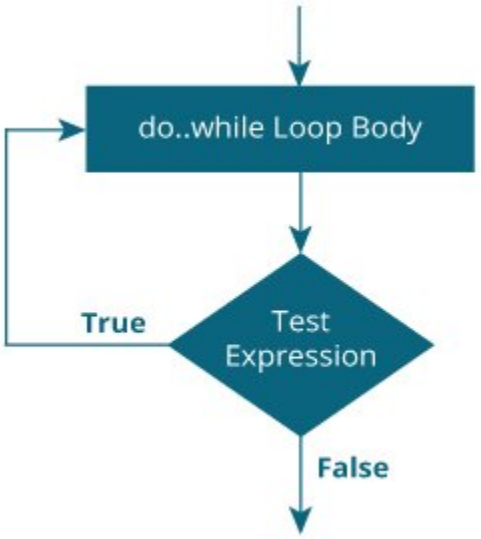
- A do...while loop in C is similar to the while loop except that the condition is always executed after the body of a loop. It is also called an exit-controlled loop.
- In the do-while loop, the body of a loop is always executed at least once. After the body is executed, then it checks the condition.
- If the condition is true, then it will again execute the body of a loop otherwise control is transferred out of the loop.
- Similar to the while loop, once the control goes out of the loop the statements which are immediately after the loop is executed.
- The critical difference between the while and do-while loop is that in while loop the while is written at the beginning. In do-while loop, the while condition is written at the end and terminates with a semi-colon (;)

Syntax:

```
do {

 //statements
} while (expression);
```

Flowchart:



Example :

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int num=1;      //initializing the variable
        do      //do-while loop
        {
```

```
                printf("%d\n",2*num);
                num++;          //incrementing operation
        }while(num<=10);

}
```

Output:

```
2
4
6
8
10
12
14
16
18
20
```

For Loop in C

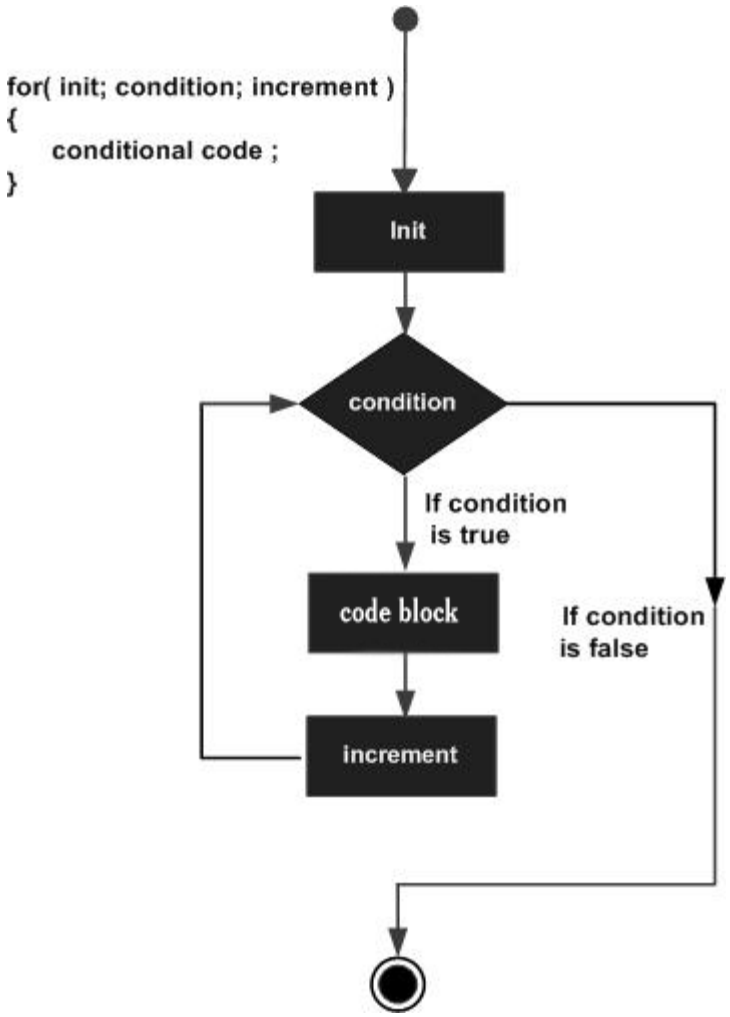
- A for loop is a repetition control structure which allows us to write a loop that is executed a specific number of times.
- The loop enables us to perform n number of steps together in one line.

It have 3 sections : initialization expr; test expr; incre/decre

```
for ( init; condition; incre/decre ) {
   statement(s);
}
```

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.



Example

Live Demo

```
#include <stdio.h>

void main () {

   int a;

   /* for loop execution */
   for( a = 10; a < 20; a++ ){
      printf("value of a: %d\n", a);
   }


}
```

```
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
```

Jump Statements

- Jump Statement makes the control jump to another section of the program unconditionally when encountered. It is usually used to terminate the loop or switch-case instantly. It is also used to escape the execution of a section of the program.

there are 4 jump statements offered by C Programming Language:

- ***Break***
- ***Continue***
- ***Goto***
- ***Return***

1.Break

break statement is used to terminate the execution of the rest of the block where it is present and takes the control out of the block to the next statement.

- It is mostly used in loops and switch-case to bypass the rest of the statement and take the control to the end of the loop. The use of the **break** keyword in switch-case has been explained in the previous tutorial Switch – Control Statement.
- Another point to be taken into consideration is that the **break** statement when used in nested loops only terminates the inner loop where it is used and not any of the outer loops.
- Let's implement an example to understand how *break* statement works in C language.

```
#include <stdio.h>

void main() {

int i;

for (i = 1; i <= 15; i++) {

printf("%d\n", i);

if (i == 10)

break;

}
}
```

```
Output:-
1
2
3
4
5
6
7
8
9
10
```

- In this program, we see that as soon as the condition *if(i==10)* becomes true the control flows out of the loop and the program ends.

2. continue

The `continue` jump statement like any other jump statement interrupts or changes the flow of control during the execution of a program. **Continue** is mostly used in loops.

Rather than terminating the loop it stops the execution of the statements underneath and takes control to the next iteration.

Similar to a break statement, in the case of a nested loop, the continue passes the control to the next iteration of the inner loop where it is present and not to any of the outer loops.

Let's implement an example to understand how *continue* statement works in C language.

```
#include <stdio.h>
int main() {
int i, j;
for (i = 1; i < 3; i++) {
for (j = 1; j < 5; j++) {
if (j == 2)
continue;
printf("%d\n", j);
}
}
return 0;
}
```

```
Output:-
1
3
4
1
3
4
```

In this program, we see that the *printf()* instruction for the condition **j=2** is skipped each time during the execution because of **continue**. We also see that only the condition **j=2** gets affected by the **continue**. The outer loop runs without any disruption in its iteration.

3. Return

`Return` jump statement is usually used at the end of a function to end or terminate it with or without a value

- . It takes the control from the calling function back to the main function(main function itself can also have a **return**).

- An important point to be taken into consideration is that **return** can only be used in functions that is declared with a **return** type such as *int, float, double, char,* etc.
- The functions declared with void type does not return any value. Also, the function returns the value that belongs to the same data type as it is declared. Here is a simple example to show you how the **return** statement works.
- Let's implement an example to understand *return* statement in C language.

```
#include <stdio.h>

char func(int ascii) {

return ((char) ascii);

}

int main() {

int ascii;

char ch;

printf("Enter any ascii value in decimal: \n");

scanf("%d", & ascii);

ch = func(ascii);

printf("The character is : %c", ch);

return 0;

}
```

```
Output:-
Enter any ascii value in decimal:
110
The character is : n
```

In this program, we have two functions that have a **return** type but only one function is returning a value[*func*()] and the other is just used to terminate the function[*main*()].

The function *func*() is returning the character value of the given number(here **110**). We also see that the return type of *func*() is char because it is returning a character value.

The return in *main*() function returns zero because it is necessary to have a **return** value here because `main` has been given the return type int.