MARUDHAR KESARI JAIN COLLEGE for WOMEN

PG DEPARTMENT OF COMPUTER APPLICATIONS

CLASS : I-BCA

SUBJECT NAME: STRUCTURED PROGRAMMING LANGUAGE IN C

SUBJECT CODE: 23UFC14

SYLLABUS

UNIT 2

Decision Making and Branching: Decision making with if, simple IF, IFELSE nested IFELSE, ELSEIF ladder, switch, GOTO statement. **Decision Making and Looping**: While, Do-While, For, Jumps in loops.

DECISION MAKING AND BRANCHING

In C, decision-making statements are technology structures enabling programmers to make decisions based on specific conditions or criteria. In C, there are three primary decision-making statements that you can use:

- ✓ Simple If
- ✓ IF else
- ✓ Nested if else
- ✓ Else if ladder
- ✓ Switch and
- ✓ GOTO

Conditional operator statements

Each of these statements allows you to make decisions in different ways, depending on the complexity of your code and the specific conditions you need to evaluate.

Simple IF

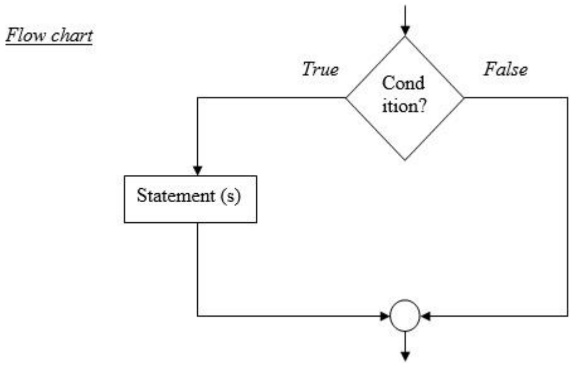
'if' keyword is used to execute a set of statements when the logical condition is true.

Syntax

The syntax is given below −

```
if (condition)
{
  Statement (s)
}
```

Working of 'simple if' statement



- The statement inside the if block are executed only when condition is true, otherwise not.
- If we want to execute only one statement when condition is true, then braces ({}) can be removed. In general, we should not omit the braces even if; there is a single statement to execute.
- When the condition is true the braces ({}) are required to execute more than one statement.

Example

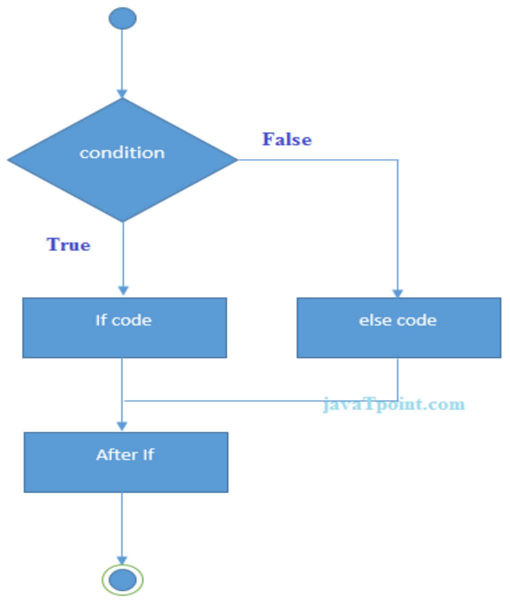
Given below is the C program to execute If conditional operators –

```
#include<stdio.h>
void main ()
{
  int a=4;
  printf("Enter the value of a: ");
  scanf("%d",&a);
  if(a%2==1){
    printf("a is odd number");
  }
}
```

```
Output

Run 1: Enter the value of a: 56

a is even number

Run2: Enter the value of a: 33
```

IF Else:

The if-else statement is used to perform two operations for a single condition. The if-else statement is an extension to the if statement , we can perform two different operations, i.e., one is for the correctness of that condition, and the other is for the incorrectness of the condition.



we must notice that if and else block cannot be executed simultaneously.

Using if-else statement is always preferable since it always invokes an otherwise case with every if condition.

The syntax of the if-else statement is given below.

```
   if(expression){
//code to be executed if condition is true
}else{
//code to be executed if condition is false
}
```

Example:

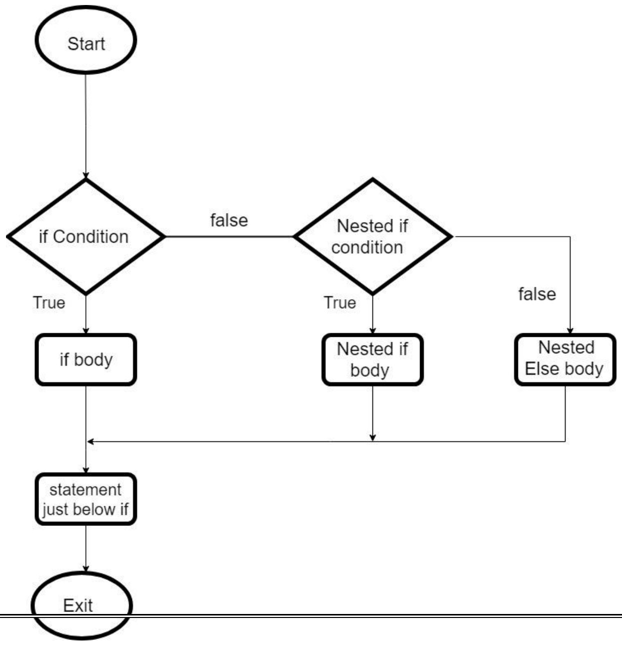
```
#include<stdio.h>
void main()
{
int number=0;
printf("enter a number:");
scanf("%d",&number);
if(number%2==0)
{
printf("%d is even number",number);
}
Else
{
printf("%d is odd number",number);
}
}
```

Output:

```
enter a number:4
4 is even number
enter a number:5
5 is odd number
```

Nested If else:

In more complex scenarios, **nested if-else statements** can be used to make more sophisticated decisions. This blog post will provide an in-depth explanation of nested if-else statements in C .



Syntax:

if (condition1)

```
{
   /* code to be executed if condition1 is true */
   if (condition2)
{
  /* code to be executed if condition2 is true */
}
else
{
 /* code to be executed if condition2 is false */
}
}
else
{
   /* code to be executed if condition1 is false */
}
```

Example:

```
            #include <stdio.h>
            void main()
            {
               int num;
        printf("Enter a number: ");
            scanf("%d", &num);
              if (num > 0)
            {
            printf("%d is positive.\n", num);
               }
            else
              {
                  if (num < 0)
            {
            printf("%d is negative.\n", num);
                  }
            else
                {
            printf("%d is zero.\n", num);
                  }
            }



        }
```

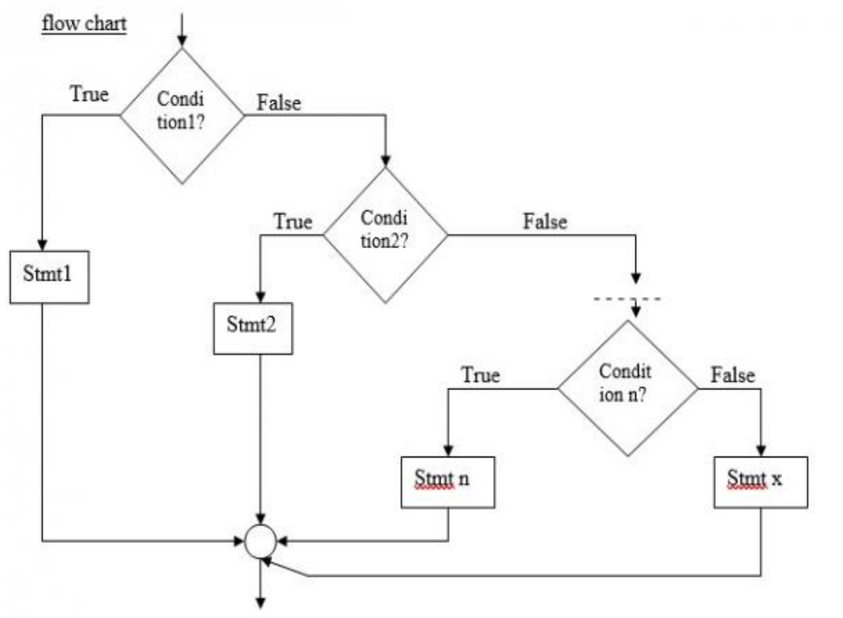
OUTPUT

```
Enter a number: 10
10 is positive.

Enter a number: -5
-5 is negative.

Enter a number: 0
0 is zero.
```

Else if Ladder:

if else if ladder in C is used to test a series of conditions sequentially. Furthermore, if a condition is tested only when all previous if conditions in the if-else ladder are false. If any of the conditional expressions evaluate to be true, the appropriate code block will be executed, and the entire if-else ladder will be terminated.

Syntax



if (condition1)

stmt1;

else if (condition2)

stmt2;

- - - - -

- - - - -

else if (condition n)

stmtn;

else

stmt x;

Example:

```
#include<stdio.h>
void main ()
{
  inta,b,c,d;

  printf("Enter the values of a,b,c,d: ");
  scanf("%d%d%d%d",&a,&b,&c,&d);
  if(a>b && a>c && a>d)
  {
     printf("%d is the largest",a);
   }
else if(b>c && b>a && b>d)
  {
     printf("%d is the largest",b);

  }
else if(c>d && c>a && c>b)
 {
     printf("%d is the largest",c);
 }
else
{
     printf("%d is the largest",d);
}
```

```
Output:

Run 1:Enter the values of a,b,c,d: 2 4 6 8
8 is the largest
Run 2: Enter the values of a,b,c,d: 23 12 56
23
56 is the largest
```

}

Switch:

The switch statement in C is an alternate to if-else-if ladder statement which allows us to execute multiple operations for the different possible values of a single variable called switch variable. Here, We can define various statements in the multiple cases for the different values of a single variable.

Syntax:

```
switch(expression)
{
case value1:
 //code to be executed;
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
......

default:
 code to be executed if all cases are not matched;
}
```

Rules for switch statement in C language

1) The switch expression must be of an integer or character type.

2) The case value must be an integer or character constant.

3) The case value can be used only inside the switch statement.

4) The break statement in switch case is not must. It is optional. If there is no break statement found in the case, all the cases will be executed present after the matched case. It is known as fall through the state of C switch statement.

int x,y,z;

char a,b;

float f;

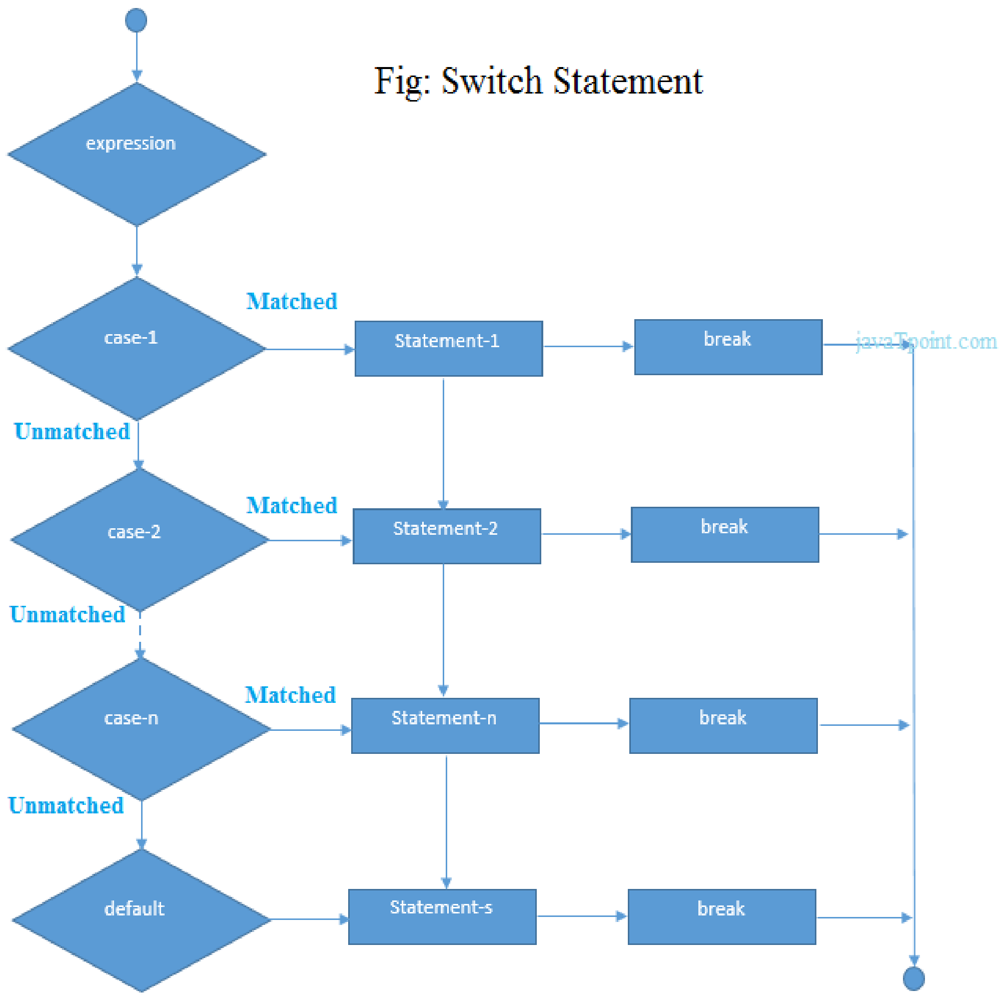


Fig: Switch Statement

Example:

#include<stdio.h>

int main()

{

int number=0;

printf("enter a number:");

```
scanf("%d",&number);
switch(number)
{
case 10:


printf("number is equals to 10");
break;
case 50:
printf("number is equal to 50");
break;
case 100:
printf("number is equal to 100");
break;
default:
printf("number is not equal to 10, 50 or 100");
}
return 0;
}
```

```
Output1:


enter a number:4
number is not equal to 10, 50 or 100
```

GOTO Statement:

- The goto statement is known as jump statement in C. As the name suggests, goto is used to transfer the program control to a predefined label.
- The goto statement can be used to repeat some part of the code for a particular condition. It can also be used to break the multiple loops which can't be done by using a single break statement.
- The goto statement can be used to jump from anywhere to anywhere within a function.

Syntax:

Syntax1 | Syntax2

```
goto label;  |    label:

label:         |    goto label;
```

Example:

```
#include <stdio.h>
int main()
{
  int num,i=1;
  printf("Enter the number whose table you want to print?");
  scanf("%d",&num);
  table:
  printf("%d x %d = %d\n",num,i,num*i);
  i++;
  if(i<=10)
  goto table;
}
```

Output:

```
Enter the number whose table you want to print?10
10 x 1 = 10
10 x 2 = 20
10 x 3 = 30
10 x 4 = 40
10 x 5 = 50
10 x 6 = 60
10 x 7 = 70
10 x 8 = 80
10 x 9 = 90
10 x 10 = 100
```

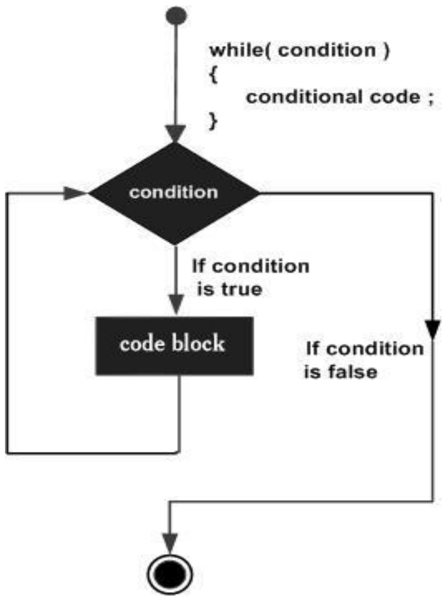
DECISION MAKING AND LOOPING

While Loop:

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

Syntax:

```
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.

Example:

```
#include <stdio.h>

void main ()

{

  /* local variable definition */

int a = 10;
```

```
   /* while loop execution */
while( a < 20 )
   {
printf("value of a: %d\n", a);
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
```

DO WHILE

- The **do…while in C** is a loop statement used to repeat some part of the code till the given condition is fulfilled.
- It is a form of an **exit-controlled or post-tested loop** where the test condition is checked after executing the body of the loop.
- Due to this, the statements in the do…while loop will always be executed at least once no matter what the condition is.

Syntax of do…while Loop in C

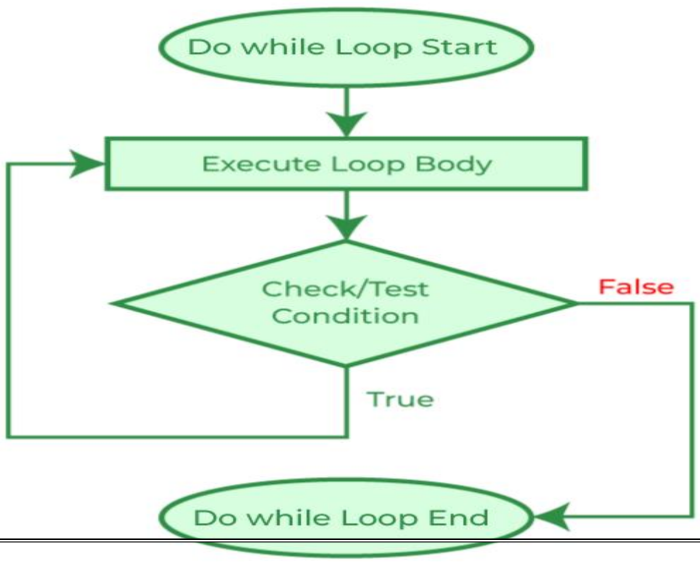
```
 do {

 // body of do-while loop

} while (condition);
```

How does the do…while Loop works?

The working of the do…while loop is explained below:

1. When the program control first comes to the do…while loop, **the body of the loop is executed first and then the test condition/expression is checked**, unlike other loops where the test condition is checked first. Due to this property, the do…while loop is also called exit controlled or post-tested loop.
2. When the test condition is evaluated as **true**, the **program control goes to the start** of the loop and the body is executed once more.
3. The above process repeats till the test condition is true.
4. When the test condition is evaluated as **false, the program controls move on to the next statements** after the do…while loop.

FLOW CHART



PROGRAM

```
#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;

}
```

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
```

while Loop	**do…while Loop**
The test condition is checked **before the loop body is executed.**	The test condition is checked **after executing the body.**
When the condition is false, the **body is not executed** not even once.	The body of the **do…while loop is executed at least once** even when the condition is false.
It is a type of **pre-tested or entry-controlled loop.**	It is a type of **post-tested or exit-controlled loop.**
Semicolon is not required.	Semicolon is required at the end.

for Loop in C

The **for loop** in C Language provides a functionality/feature to repeat a set of statements a defined number of times. The for loop is in itself a form of an **entry-controlled loop**.

The for loop contains the initialization, condition, and updating statements as part of its syntax. It is mainly used to traverse arrays, vectors, and other data structures.

Syntax of for Loop

```
for(initialization; check/test expression; updation)

{

    // body consisting of multiple statements

}
```

Structure of for Loop

The for loop follows a very structured approach where it begins with initializing a condition then checks the condition and in the end executes conditional statements followed by an updation of values.

1. **Initialization**
2. **Check/Test Condition**
3. **Body**
4. **Updation**

program

```
#include <stdio.h>

 int main()

{

   inti;

   // for loop without curly braces

   for (i = 1; i<= 10; i++)
```

```
    printf("%d ", i);

    printf("\nThis statement executes after for loop end!!!!"); // Statement print only once


    return 0;

}
```

OUTPUT

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

This statement executes after for loop end!!!!
```

JUMPS INLOOP

Jump Statement in C is used in C programming to transfer the program control from one part of the code to another.

Types of Jump Statements in C

There are 4 types of jump statements in C:

1. break
2. continue
3. goto
4. return

1. break in C

The break statement exits or terminates the loop or switch statement based on a certain condition, without executing the remaining code.

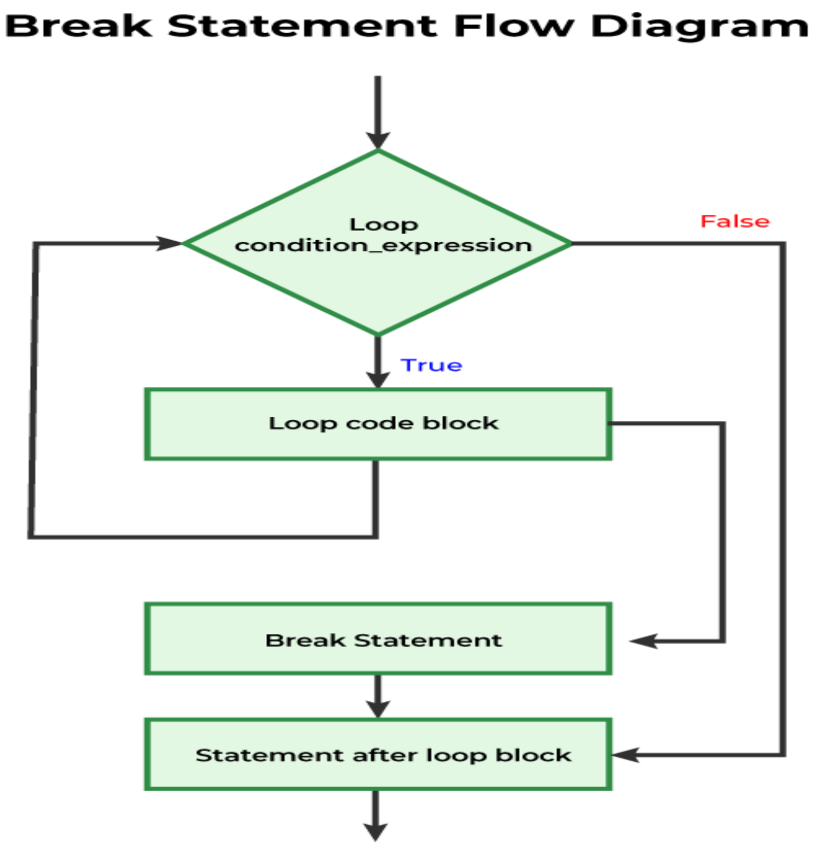
Syntax of break in C

{

condition

break;

}


Break Statement Flow Diagram
Loop condition_expression
False
True
Loop code block
Break Statement
Statement after loop block

Uses of break in C

The break statement is used in C for the following purposes:

- To come out of the loop.
- To come out from the nested loops.
- To come out of the switch case.

PROGRAM

```
#include <stdio.h>
int main()
{
        inti;
        // for loop
        for (i = 1; i<= 10; i++) {

            // when i = 6, the loop should end
    if (i == 6)
    {
               break;
            }
            printf("%d ", i);
   }
        printf("Loop exited.\n");
   return 0;
    }
```

OUTPUT

```
1 2 3 4 5 Loop exited.
```

Explanation:

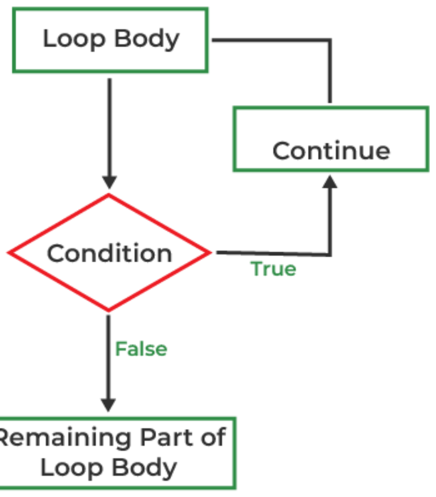
- Loop Execution Starts and goes normally till i = 5.
- When i = 6, the condition for the break statement becomes true and the program control immediately exits the loop.

2. Continue in C

- The continue statement in C is used to skip the remaining code after the continue statement within a loop and jump to the next iteration of the loop.
- When the continue statement is encountered, the loop control immediately jumps to the next iteration, by skipping the lines of code written after it within the loop body.

Syntax of continue in C

continue;



Example of continue Statement

```
#include <stdio.h>

int main()

{

for (int j=0; j<=8; j++)

   {

if (j==4)

      {
            /* The continue statement is encountered when
             * the value of j is equal to 4.
             */
          continue;
       }

       /* This print statement would not execute for the
          * loop iteration where j ==4  because in that case
          * this statement would be skipped.
          */
printf("%d ", j);
   }
return 0;
}
```

Output:

```
0 1 2 3 5 6 7 8
```

EXPLANITATION

- ✓ Value 4 is missing in the output, why?
- ✓ When the value of variable j is 4, the program encountered a continue statement, which makes the control to jump at the beginning of the for loop for next iteration, skipping the statements for current iteration (that’s the reason printf didn’t execute when j is equal to 4).

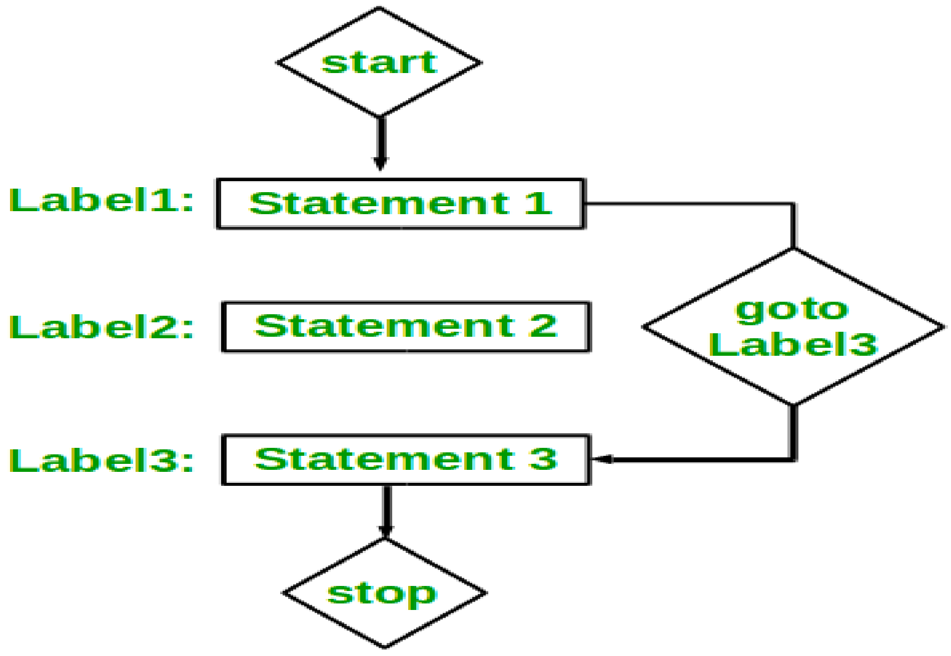
3. Goto Statement in C

The goto statement is used to jump to a specific point from anywhere in a function. It is used to transfer the program control to a labeled statement within the same function.

Syntax of goto Statement

gotolabel;
..
label:
//code

Flowchart of goto Statement



PROGRAM

```
#include <stdio.h>

int main()

{

int sum=0;

for(inti = 0; i<=10; i++){

            sum = sum+i;

            if(i==5){

            goto addition;

            }

   }

addition:

printf("%d", sum);

return 0;

}
```

Output:

```
15
```

Explanation: In this example, we have a label addition and when the value of i (inside loop) is equal to 5 then we are jumping to this label using goto. This is reason the sum is displaying the sum of numbers till 5 even though the loop is set to run from 0 to 10.

4. Return Statement in C

The return statement in C is used to terminate the execution of a function and return a value to the caller. It is commonly used to provide a result back to the calling code.

```
returnexpression;
```

```
int total (){
  inta,b,c;
  a=10;
  b=20;
  c=a+b;
  return c; //it returns the c value i.e. prints the result
}
```

Output

When the above program is executed, it produces the following result −

```
30
```