UNIT -3Syllabus:
Selection Statements – if and switch statements, Repetitive statements – while, for, do-while statements, C Programming examples, other statements related to looping – break, continue, goto, C Programming examples, Arrays- Basic concepts, one-dimensional arrays, two-dimensional arrays, multidimensional arrays, C Programming examples.

1. Selection Staements

1.1. if statement

The if statement checks whether the text expression inside parenthesis () is true or not. If the test expression is true, statement/s inside the body of if statement is executed but if test is false, statement/s inside body of if is ignored.

Syntax is,

```
if (test_expression)
{
        Staements to be executed if test expression is true;;
}
```

Flow of if statement

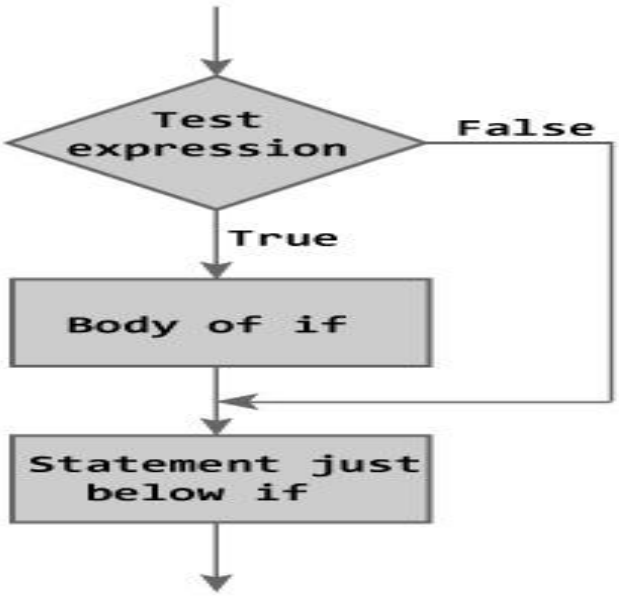


Figure: Flowchart of if Statement

Example :

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

```
intnum;;
printf("\n enter a number to check : ");
scanf("%d",&num);
if(num<0)  /* checking whether number is less than 0 or not. */
{
printf("\n Number =%d",num);
}
        /*If test condition is true, statement above will be executed, otherwise it will not
        be executed */
printf("The if statement in C programing is easy ");
getch();;
}
```

Output 1

```
Enter a number to check.
-2
Number = -2

The if statement in C programming is easy.
```

When user enters -2 then, the test expression (num<0) becomes true. Hence, Number = -2 is displayed in the screen.

Output 2

```
Enter a number to check.
5

The if statement in C programming is easy.
```

When the user enters 5 then, the test expression (num<0) becomes false. So, the statement/s inside body of if is skipped and only the statement below it is executed.

1.2. if – else statement

The if...else statement is used if the programmer wants to execute some statement/s when the test expression is true and execute some other statement/s if the test expression is false.

Syntax is,

```
if (test expression)
{
        statements to be executed if test expression is true;
}
else
{
        statements to be executed if test expression is false;
}
```

Flow chart of if-else statement

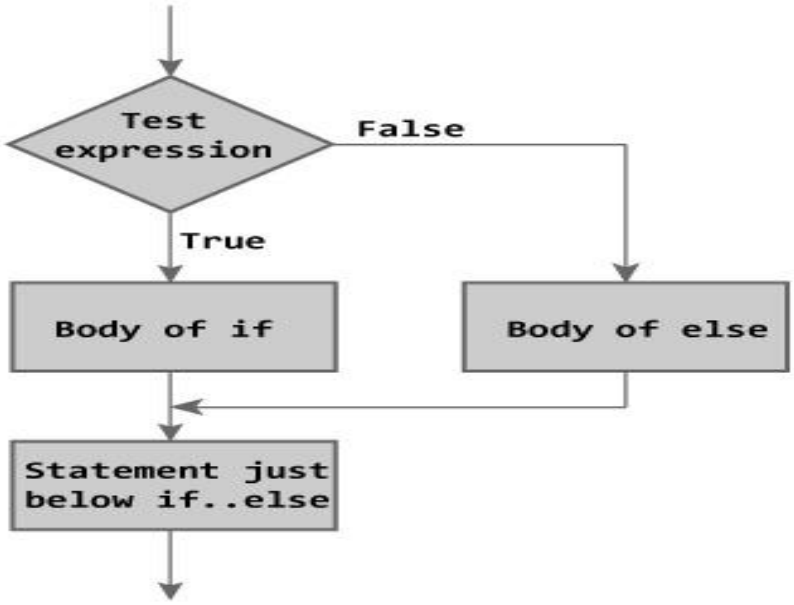


Figure: Flowchart of if...else Statement

Example:

Write a C program to check whether a number entered by user is even or odd

```
#include<stdio.h>
int main(){
intnum;
printf("Enter a number you want to check.\n");
scanf("%d",&num);
if((num%2)==0)//checking whether remainder is 0 or not.
printf("%d is even.",num);
else
printf("%d is odd.",num);
return0;
}
```

Output 1

```
Enter a number you want to check.
25
25 is odd.
```

Output 2

```
Enter a number you want to check.
2
2 is even.
```

1.3. Nested if –else statement (if – else if)

The nested if...else statement is used when program requires more than one test expression.

Syntax is

```
if (test expression1)
{
 statement/s to be executed if test expression1 is true;
}
else if(test expression2)
{
 statement/s to be executed if test expression1 is false and 2 is
 true;
}
else if (test expression 3)
{
 statement/s to be executed if text expression1 and 2 are false and 3
 is true;
}
        .
        .
        .
else
{
 statements to be executed if all test expressions are false;
}
```

How nested if...else works?

The nested if...else statement has more than one test expression. If the first test expression is true, it executes the code inside the braces{ } just below it. But if the first test expression is false, it checks the second test expression. If the second test expression is true, it executes the statement/s inside the braces{ } just below it. This process continues. If all the test expression are false, code/s inside else is executed and the control of program jumps below the nested if...else

The ANSI standard specifies that 15 levels of nesting may be continued.

Example:

Write a C program to relate two integers entered by user using = or > or < sign.

```
#include<stdio.h>
int main(){
int numb1, numb2;
printf("Enter two integers to check\n");
scanf("%d %d",&numb1,&numb2);
if(numb1==numb2)//checking whether two integers are equal.
printf("Result: %d = %d",numb1,numb2);
else
if(numb1>numb2)//checking whether numb1 is greater than numb2.
printf("Result: %d > %d",numb1,numb2);
else
printf("Result: %d > %d",numb2,numb1);
return0;
}
```

Output 1

```
Enter two integers to check.
5
3
Result: 5 > 3
```

Output 2

```
Enter two integers to check.
-4
-4
Result: -4 = -4
```

1.4. switch Statement

Decision making are needed when, the program encounters the situation to choose a particular statement among many statements. If a programmer has to choose one block of statement among many alternatives, nested if...else can be used but, this makes programming logic complex. This type of problem can be handled in C programming using switch statement.

Syntax of switch statement is,

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

```
 case constant2:
 code/s to be executed if n equals to constant2;
 break;
   .
   .
   .
 default:
 code/s to be executed if n doesn't match to any cases;
}
```

The value of *n* is either an integer or a character in above syntax. If the value of *n* matches constant in case, the relevant codes are executed and control moves out of the switch statement. If the *n*doesn't matches any of the constant in case, then the default codes are executed and control moves out of switch statement.

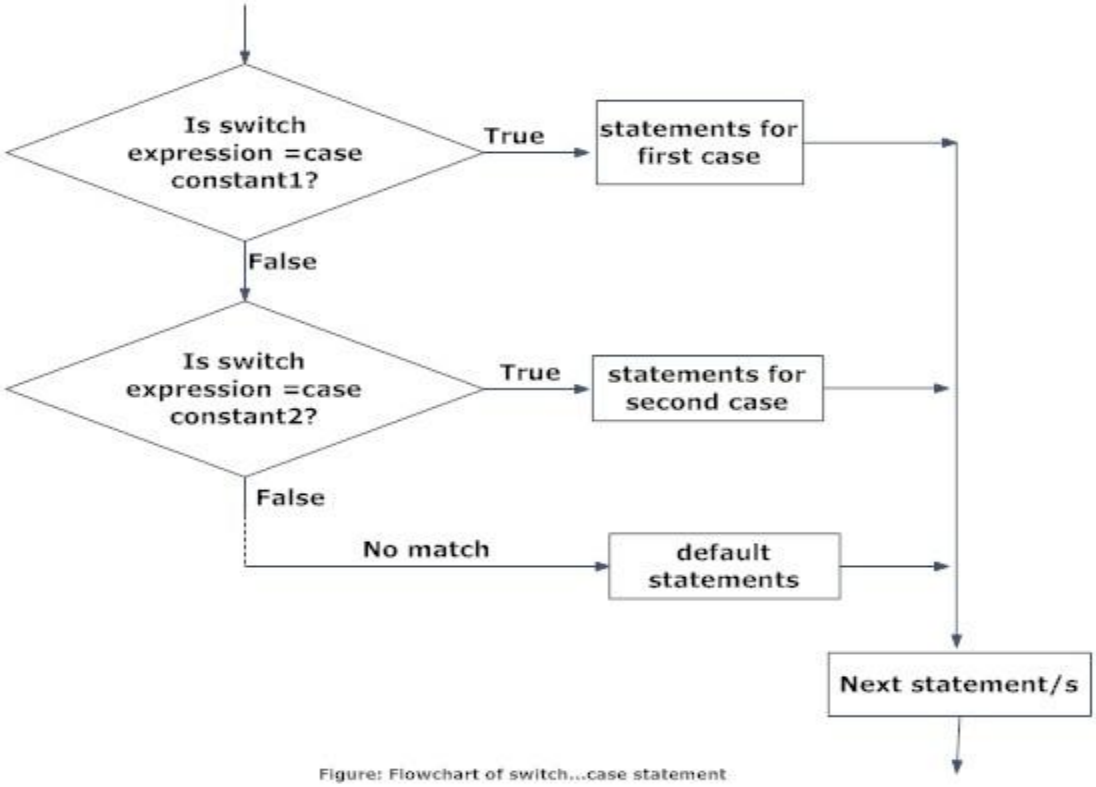


Figure: Flowchart of switch...case statement

Example:

Write a program that asks user an arithmetic operator('+','-','*' or '/') and two operands and perform the corresponding calculation on the operands.

```
/* C program to demonstrate the working of switch...case statement */
/* C Program to create a simple calculator for addition, subtraction,
multiplication and division */

# include <stdio.h>
```

```
int main(){
char o;
float num1,num2;
printf("Select an operator either + or - or * or / \n");
scanf("%c",&o);
printf("Enter two operands: ");
scanf("%f%f",&num1,&num2);
switch(o){
case'+':
printf("%.1f + %.1f = %.1f",num1, num2, num1+num2);
break;
case'-':
printf("%.1f - %.1f = %.1f",num1, num2, num1-num2);
break;
case'*':
printf("%.1f * %.1f = %.1f",num1, num2, num1*num2);
break;
case'/':
printf("%.1f / %.1f = %.1f",num1, num2, num1/num2);
break;
default:
/* If operator is other than +, -, * or /, error message is shown */
printf("Error! operator is not correct");
break;
}
return0;
}
```

Output

```
Enter operator either + or - or * or /
*
Enter two operands: 2.3
4.5
2.3 * 4.5 = 10.3
```

The break statement at the end of each case cause switch statement to exit. If break statement is not used, all statements below that case statement are also executed.

2. Repetitive statements (Iterative statements)

There may be a situation, when you need to execute a block of code several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.Programming languages provide various control structures that allow for more complicated execution paths.A loop statement or repetitive statement allows us to execute a statement or group of statements multiple times.

C programming language provides the following types of loop to handle looping requirements. Click the following links to check their detail.

Loop Type	Description
while loop	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
for loop	Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.
do...while loop	Like a while statement, except that it tests the condition at the end of the loop body
nested loops	You can use one or more loop inside any another while, for or do..while loop.

2.1. while loop

The while loop checks whether the test expression is true or not. If it is true, code/s inside the body of while loop is executed,that is, code/s inside the braces { } are executed. Then again the test expression is checked whether test expression is true or not. This process continues until the test expression becomes false.

Syntax of while loop,

```
while (test expression)
{
 statement/s to be executed.
}
```

Flow of while loop,

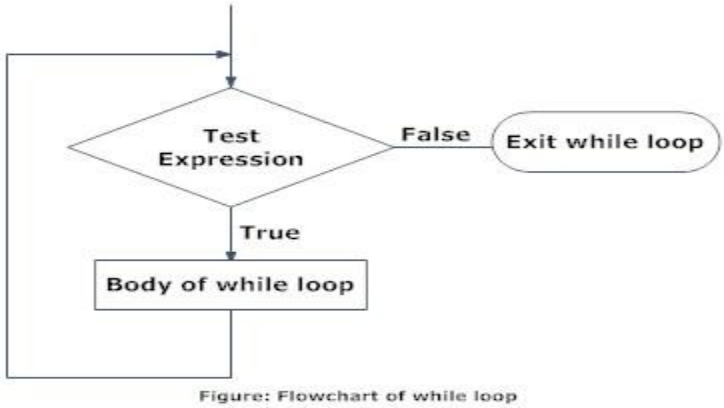


Figure: Flowchart of while loop

Example :

Write a C program to find the factorial of a number, where the number is entered by user. (Hints: factorial of n = 1*2*3*...*n

```
/*C program to demonstrate the working of while loop*/
#include<stdio.h>
int main(){
intnumber,factorial;
printf("Enter a number.\n");
scanf("%d",&number);
factorial=1;
while(number>0){/* while loop continues util test condition number>0 is true */
factorial=factorial*number;
--number;
}
printf("Factorial=%d",factorial);
return0;
}
```

Output

```
Enter a number.
5
Factorial=120
```

2.2. do-while loop

In C, do...while loop is very similar to while loop. Only difference between these two loops is that, in while loops, test expression is checked at first but, in do...while

loop code is executed at first then the condition is checked. So, the code are executed at least once in do...while loops.

Syntax of do-while loop is,

```
do
{
some code/s;
}while (test expression);
```

At first codes inside body of do is executed. Then, the test expression is checked. If it is true, code/s inside body of do are executed again and the process continues until test expression becomes false(zero).

Notice, there is semicolon in the end of while (); in do...while loop.

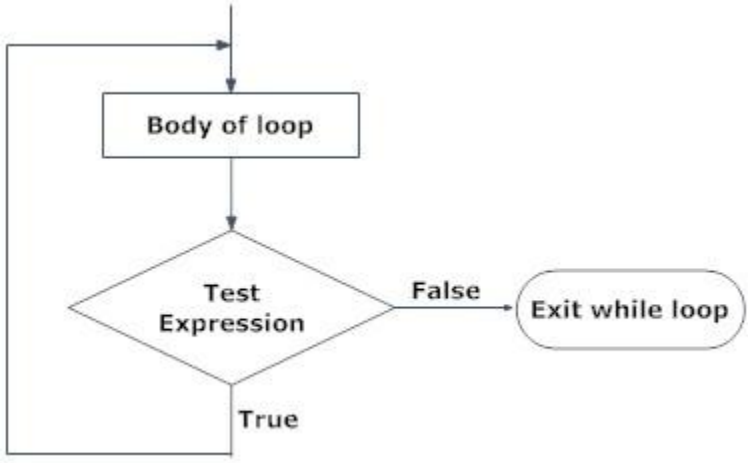


Figure: Flowchart of do...while loop

Example

Write a C program to add all the numbers entered by a user until user enters 0.

```
/*C program to demonstrate the working of do...while statement*/
#include<stdio.h>
int main(){
int sum=0,num;
do/* Codes inside the body of do...while loops are at least executed once. */
{
printf("Enter a number\n");
scanf("%d",&num);
sum+=num;
}
while(num!=0);
```

```
printf("sum=%d",sum);
return0;
}
```

Output

```
Enter a number
3
Enter a number
-2
Enter a number
0
sum=1
```

In this C program, user is asked a number and it is added with *sum*. Then, only the test condition in the do...while loop is checked. If the test condition is true,i.e, *num* is not equal to 0, the body of do...while loop is again executed until *num* equals to zero.

2.3. for loop

For loop is also used for the similar purpose like while & do-while. It has three statements. (i) initialization statements (ii) condition statements (iii) update statements.

The initialization statement is executed only once at the beginning of the for loop. Then the test expression is checked by the program. If the test expression is false, for loop is terminated. But if test expression is true then the code/s inside body of for loop is executed and then update expression is updated. This process repeats until test expression is false.

Syntax of for loop is,

```
for(initialization statement; test expression; update statement)
{
code/s to be executed;
}
```

Flow of for loop is,

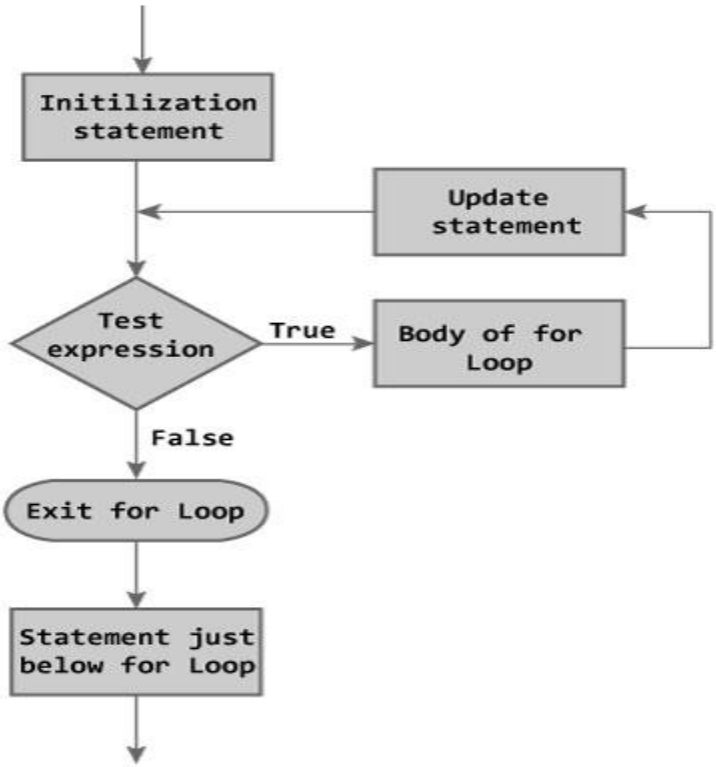


Figure: Flowchart of for Loop

Example,

Write a program to find the sum of first *n* natural numbers where n is entered by user. Note: 1,2,3... are called natural numbers.

```
#include<stdio.h>
int main(){
int n, count, sum=0;
printf("Enter the value of n.\n");
scanf("%d",&n);
for(count=1;count<=n;++count)//for loop terminates if count>n
{
sum+=count;/* this statement is equivalent to sum=sum+count */
}
printf("Sum=%d",sum);
return0;
}
```

Output

```
Enter the value of n.
19
Sum=190
```

In this program, the user is asked to enter the value of *n*. Suppose you entered 19 then, count is initialized to 1 at first. Then, the test expression in the for loop,i.e., (count<= n) becomes true. So, the code in the body of for loop is executed which makes *sum* to 1. Then, the expression ++count is executed and again the test expression is checked, which becomes true. Again, the body of for loop is executed which makes *sum* to 3 and this process continues. When count is 20, the test condition becomes false and the for loop is terminated.

Note: Initial, test and update expressions are separated by semicolon(;).

3. break, continue and goto statements

3.1.break

In C programming, break is used in terminating the loop immediately after it is encountered. The break statement is used with conditional if statement.

Syntax is,

break;

The break statement can be used in terminating all three loops for, while and do...while loops.

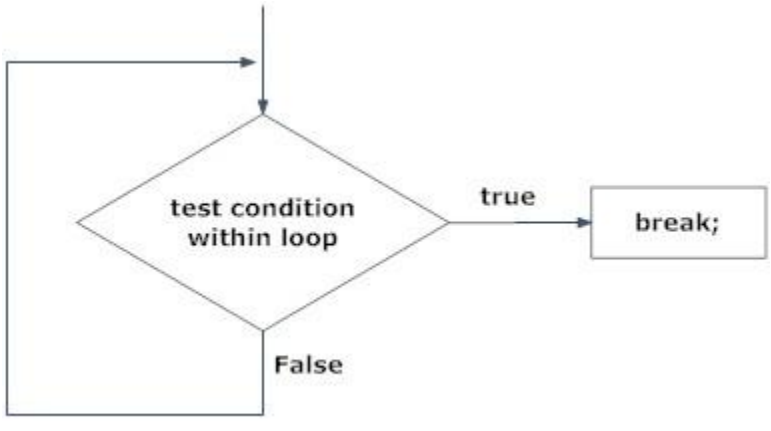
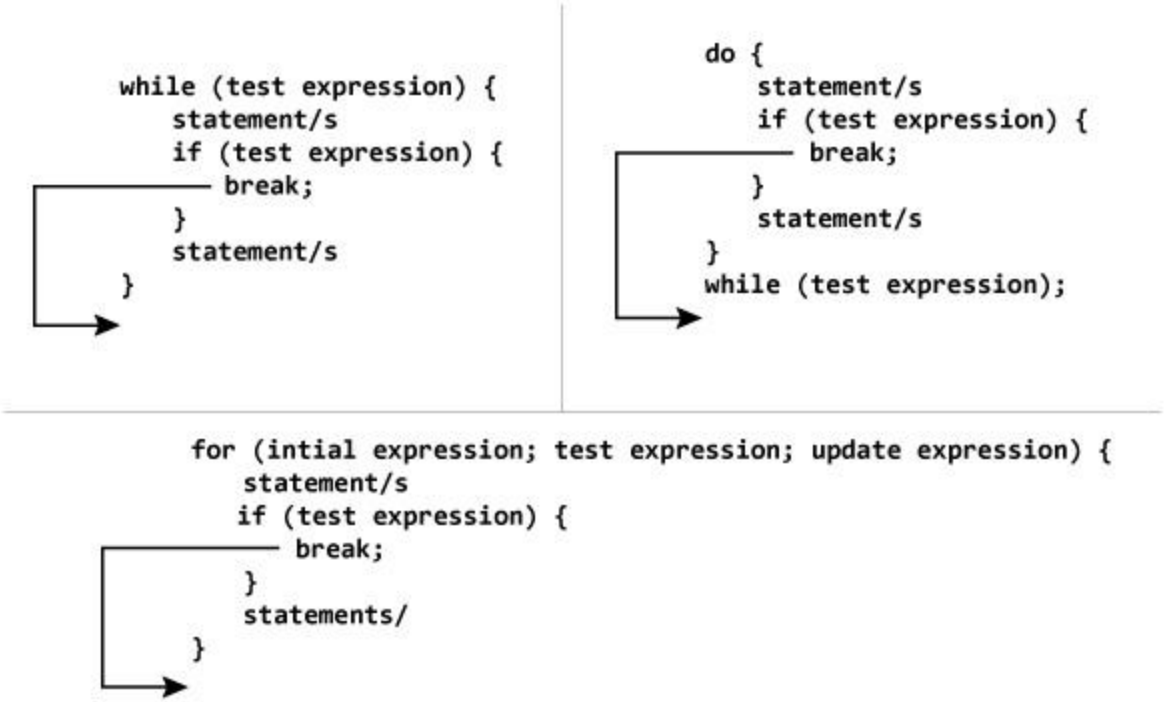


Figure: Flowchart of break statement

The figure below explains the working of break statement in all three type of loops.



NOTE: The break statment may also be used inside body of else statement.

Example:

Write a C program to find average of maximum of *n* positive numbers entered by user. But, if the input is negative, display the average(excluding the average of negative input) and end the program.

```
/* C program to demonstrate the working of break statement by terminating a loop, if user inputs negative number*/
# include <stdio.h>
int main(){
floatnum,average,sum;
inti,n;
printf("Maximum no. of inputs\n");
scanf("%d",&n);
for(i=1;i<=n;++i){
printf("Enter n%d: ",i);
scanf("%f",&num);
if(num<0.0)
break;//for loop breaks if num<0.0
sum=sum+num;
}
average=sum/(i-1);
```

```
printf("Average=%.2f",average);
return0;
}
```

Output

```
Maximum no. of inputs
4
Enter n1: 1.5
Enter n2: 12.5
Enter n3: 7.2
Enter n4: -1
Average=7.07
```

In this program, when the user inputs number less than zero, the loop is terminated using break statement with executing the statement below it i.e., without executing sum=sum+num.

In C, break statements are also used in switch...case statement. You will study it in C switch… case statement chapter.

3.2. continue Statement

It is sometimes desirable to skip some statements inside the loop. In such cases, continue statements are used.

Syntax is,

continue;

Just like break, continue is also used with conditional if statement.

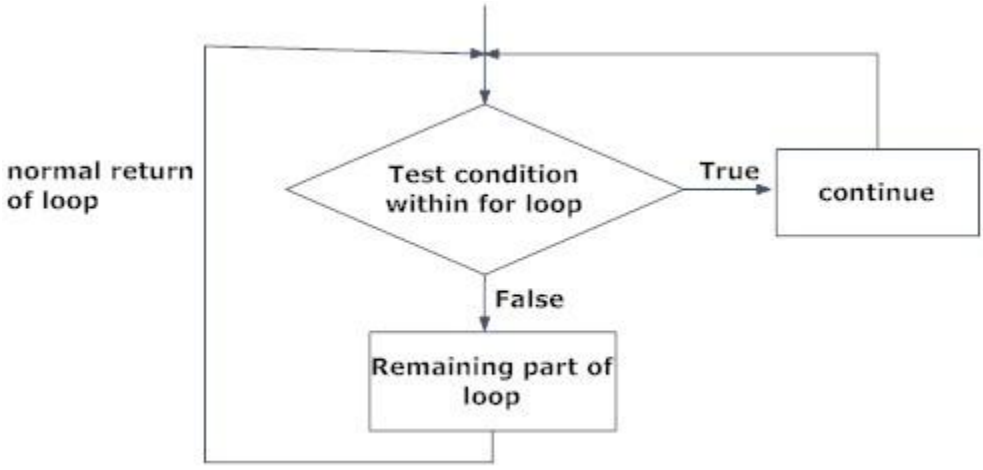
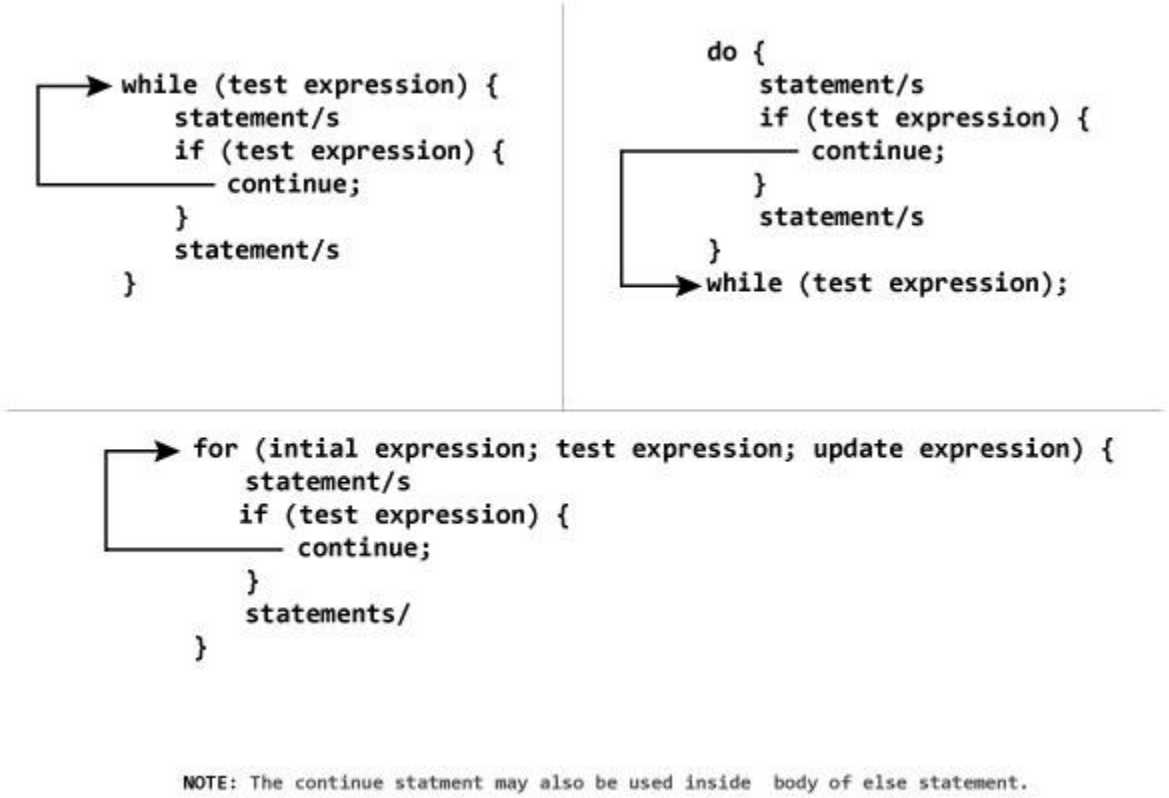


Fig: Flowchart of continue statement

For better understanding of how continue statements works in C programming. Analyze the figure below which bypasses some code/s inside loops using continue statement.



NOTE: The continue statment may also be used inside body of else statement.

Example:

Write a C program to find the product of 4 integers entered by a user. If user enters 0 skip it.

```
//program to demonstrate the working of continue statement in C programming
# include <stdio.h>
int main(){
inti,num,product;
for(i=1,product=1;i<=4;++i){
printf("Enter num%d:",i);
scanf("%d",&num);
if(num==0)
continue;/ *In this program, when num equals to zero, it skips the statement product*=num and continue the loop. */
product*=num;
}
printf("product=%d",product);
return0;
```

```
}
```

Output

```
Enter num1:3
Enter num2:0
Enter num3:-5
Enter num4:2
product=-30
```

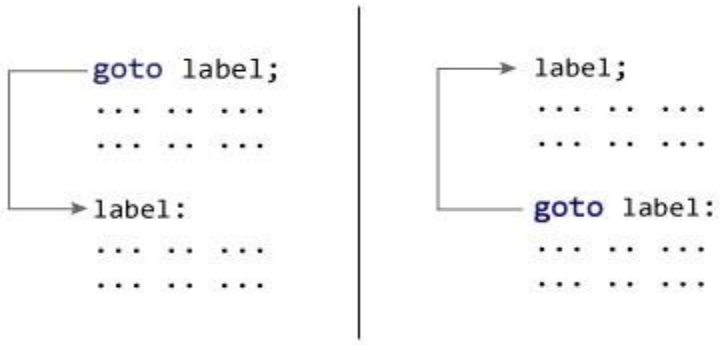
3.3.goto statement

In C programming, goto statement is used for altering the normal sequence of program execution by transferring control to some other part of the program.

Syntax of goto statement is,

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

In this syntax, label is an identifier. When, the control of program reaches to goto statement, the control of the program will jump to the label: and executes the code below it.



Example

```
/*  C program to demonstrate the working of goto statement. */
/* This program calculates the average of numbers entered by user. */
/* If user enters negative number, it ignores that number and
calculates the average of number entered before it.*/
# include <stdio.h>
```

```
int main(){
floatnum,average,sum;
inti,n;
printf("Maximum no. of inputs: ");
scanf("%d",&n);
for(i=1;i<=n;++i){
printf("Enter n%d: ",i);
scanf("%f",&num);
if(num<0.0)
goto jump;            /* control of the program moves to label jump */
sum=sum+num;
}
jump:
average=sum/(i-1);
printf("Average: %.2f",average);
return 0;
}
```

Output

```
Maximum no. of inputs: 4

Enter n1: 1.5
Enter n2: 12.5
Enter n3: 7.2
Enter n4: -1
Average: 7.07
```

Though goto statement is included in ANSI standard of C, use of goto statement should be reduced as much as possible in a program.

Reasons to avoid goto statement

Though, using goto statement give power to jump to any part of program, using goto statement makes the logic of the program complex and tangled. In modern programming, goto statement is considered a harmful construct and a bad programming practice.

The goto statement can be replaced in most of C program with the use of break and continue statements. In fact, any program in C programming can be perfectly written without the use of goto statement. All programmer should try to avoid goto statement as possible as they can.

4. Arrays

In C programming, one of the frequently arising problem is to handle similar types of data. For example: If the user want to store marks of 100 students. This can be done by creating 100 variable individually but, this process is rather tedious and impracticable. These type of problem can be handled in C programming using arrays.

An array is a sequence of data item of homogeneous value(same type).

Arrays are of two types:

1. One-dimensional arrays
2. Multi-dimensional arrays

4.1.Delaration of arrays,

data_typearray_name[array_size];

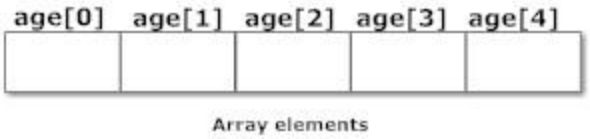
For example:
int age[5];

Here, the name of array is *age*. The size of array is 5,that is, there are 5 items(elements) of array *age*. All element in an array are of the same type (int, in this case).

Array elements

Size of array defines the number of elements in an array. Each element of array can be accessed and used by user according to the need of program. For example:

int age[5];



Array elements

Note that, the first element is numbered 0 and so on.

Here, the size of array *age* is 5 times the size of int because there are 5 elements.

Suppose, the starting address of age[0] is 2120d and the size of int be 4 bytes. Then, the next address (address of a[1]) will be 2124d, address of a[2] will be 2128d and so on.

4.2.Initialization of one-dimensional array

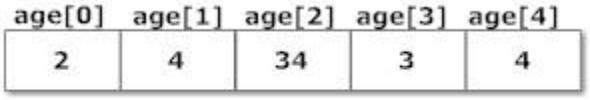
Arrays can be initialized at declaration time in this source code as:

```
int age[5]={2,4,34,3,4};
```

It is not necessary to define the size of arrays during initialization.

```
int age[]={2,4,34,3,4};
```

In this case, the compiler determines the size of array by calculating the number of elements of an array.



Initialization of one-dimensional array

4.3.Accessing array elements

In C programming, arrays can be accessed and treated like variables in C.

For example:

```
scanf("%d",&age[2]);

/* statement to insert value in the third element of array age[]. */

scanf("%d",&age[i]);
/* Statement to insert value in (i+1)th element of array age[]. */
/* Because, the first element of array is age[0], second is age[1], ith is age[i-1] and (i+1)th is age[i]. */

printf("%d",age[0]);
/* statement to print first element of an array. */

printf("%d",age[i]);
/* statement to print (i+1)th element of an array. */
```

Example :

```
/* C program to find the sum marks of n students using arrays */
#include <stdio.h>
int main(){
int marks[10],i,n,sum=0;
```

```
printf("Enter number of students: ");
scanf("%d",&n);
for(i=0;i<n;++i){
printf("Enter marks of student%d: ",i+1);
scanf("%d",&marks[i]);
sum+=marks[i];
    }
printf("Sum= %d",sum);
return 0;
}
```

Output

```
Enter number of students: 3
Enter marks of student1: 12
Enter marks of student2: 31
Enter marks of student3: 2
sum=45
```

Important thing to remember in C arrays

Suppose, you declared the array of 10 students. For example: arr[10]. You can use array members from arr[0] to arr[9]. But, what if you want to use element arr[10], arr[13] etc. Compiler may not show error using these elements but, may cause fatal error during program execution.

4.4. Multi-dimensional arrays

C programming language allows programmer to create arrays of arrays known as multidimensional arrays. For example:

```
float a[2][6];
```

Here, *a* is an array of two dimension, which is an example of multidimensional array.

For better understanding of multidimensional arrays, array elements of above example can be thinked of as below:

	col 1	col 2	col 3	col 4	col 5	col 6
row 1	a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]	a[0][5]
row 2	a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]	a[1][5]

Figure: Multidimensional Arrays

Initialization of multi-dimensional arrays

In C, multidimensional arrays can be initialized in different number of ways.

```
int c[2][3]={{1,3,0}, {-1,5,9}};
            OR
int c[][3]={{1,3,0}, {-1,5,9}};
            OR
int c[2][3]={1,3,0,-1,5,9};
```

Initialization Of three-dimensional Array

```
doublecprogram[3][2][4]={
{{-0.1, 0.22, 0.3, 4.3}, {2.3, 4.7, -0.9, 2}},
 {{0.9, 3.6, 4.5, 4}, {1.2, 2.4, 0.22, -1}},
 {{8.2, 3.12, 34.2, 0.1}, {2.1, 3.2, 4.3, -2.0}}
};
```

Suppose there is a multidimensional array arr[i][j][k][m]. Then this array can hold i*j*k*m numbers of data.

Similarly, the array of any dimension can be initialized in C programming.

Example

Write a C program to find sum of two matrix of order 2*2 using multidimensional arrays where, elements of matrix are entered by user.

```
#include <stdio.h>
int main(){
float a[2][2], b[2][2], c[2][2];
inti,j;
printf("Enter the elements of 1st matrix\n");
/* Reading two dimensional Array with the help of two for loop. If there was an array
of 'n' dimension, 'n' numbers of loops are needed for inserting data to array.*/
for(i=0;i<2;++i)
for(j=0;j<2;++j){
printf("Enter a%d%d: ",i+1,j+1);
scanf("%f",&a[i][j]);
      }
printf("Enter the elements of 2nd matrix\n");
for(i=0;i<2;++i)
for(j=0;j<2;++j){
```

```
printf("Enter b%d%d: ",i+1,j+1);
scanf("%f",&b[i][j]);
      }
for(i=0;i<2;++i)
for(j=0;j<2;++j){
/* Writing the elements of multidimensional array using loop. */
      c[i][j]=a[i][j]+b[i][j];  /* Sum of corresponding elements of two arrays. */
      }
printf("\nSum Of Matrix:");
for(i=0;i<2;++i)
for(j=0;j<2;++j){
printf("%.1f\t",c[i][j]);
if(j==1)            /* To display matrix sum in order. */
printf("\n");
     }
return 0;
}
```

Ouput

```
Enter the elements of 1st matrix
Enter a11: 2;
Enter a12: 0.5;
Enter a21: -1.1;
Enter a22: 2;
Enter the elements of 2nd matrix
Enter b11: 0.2;
Enter b12: 0;
Enter b21: 0.23;
Enter b22: 23;

Sum Of Matrix:
2.2     0.5
-0.9    25.0
```