PROBLEM SOLVING IN C

UNIT III

Decision Control and Looping Statements: Conditional Branching Statements – Iterative Statements – Nested Loops – Break and Continue Statement – Goto Statement.

Arrays and Strings: Definition of Array, Declaration of Arrays, Types of Arrays – Operations on Arrays – Declaration of Strings, String handling functions

Introduction to Decision Control Statements :

- We generally follow a sequence in the program but sometimes we may want to execute only selected statements.
- This type of conditional processing is really useful to the programs.
- It allows the programmers to build the programs that determine which statements of the code should be executed and which are to be ignored.
- In any programming language, there is a need to perform different tasks based on the condition.

- For example, consider an online website, when you enter wrong id or password it displays error page and when you enter correct credentials then it displays welcome page. So there must be a logic in place that checks the condition (id and password) and if the condition returns true it performs a task (displaying welcome page) else it performs a different task(displaying error page).

Using decision control statements we can control the flow of program in such a way so that it executes certain statements based on the outcome of a condition (i.e. true or false). These are also called as ***conditional branching statements.***

Decision Control Statements (or) Conditional branching statements:

In C Programming language we have following decision control statements.

1. if statement

2. if-else statement
3. If – else if statement
4. Nested if statement
5. switch-case statement

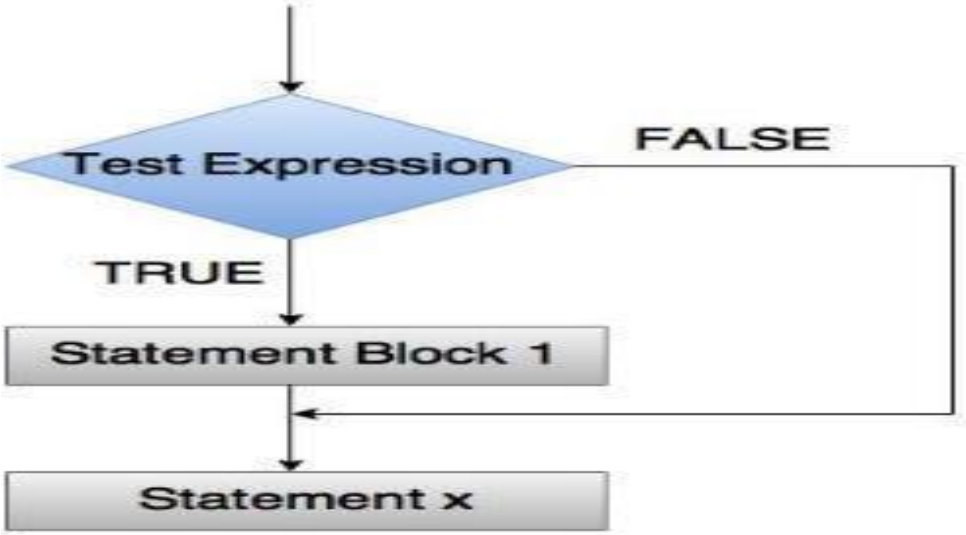
1. if statement

It is one of the most simple form of decision control statements which is frequently used in decision making.

Syntax:

```
if (test expression)
{
   statement 1;
   ….............
   statement n;
}
statement x;
```

FLOW CHART:



- The if structure may have one statement or n number of statements which are enclosed within the curly braces ({ }).

- Initially the test expression is evaluated.
- If this expression is true the statement of the if block is executed or all the statements will be skipped and statement x will be executed.

Example: Demonstrating an if statement

Write a program to print the highest number.

```
#include <stdio.h>
void main()
{
   int n;
   printf("Enter a number:");
   scanf("%d", &n);
   printf ("The entered number %d", n);
   if (n>100)
      printf ("You entered a higher number");
}
```

In the above program, a number is accepted as an input from the user and the output is given. The expression (n>100) is given. If the user gives a number higher than 100 then only the message will be printed else the statement will be skipped.

2. If-else statement :

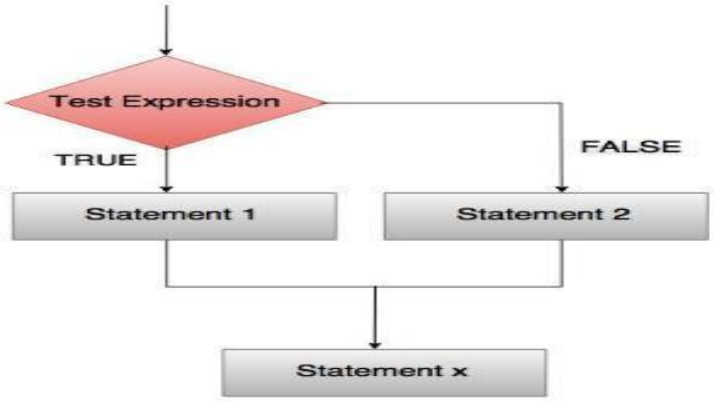
There are times where we would like to write that if the expression is true an action should be taken and if the expression is false there would be no action. We can also include an action for both the conditions.

In such cases, we use the if-else statement.

Syntax:

```
if (expression)
   Statement 1;
else
   Statement 2;
```

If the expression is true statement 1 will be executed and if the expression is false statement 2 will be executed.



Example: Program for leap year

```
#include <stdio.h>
void main()
{
    int yr;
    clrscr();
    printf("Enter a year:");
    scanf("%d", &yr);
    if ((yr%4 == 0) && ((yr%100 !=0) || (yr%400==0)))
       printf("It's a leap year...!!!!");
    else
       printf ("It's not a leap year....!!!");
}
```

Output:

Enter a year: 1997

It's not a leap year...!!!

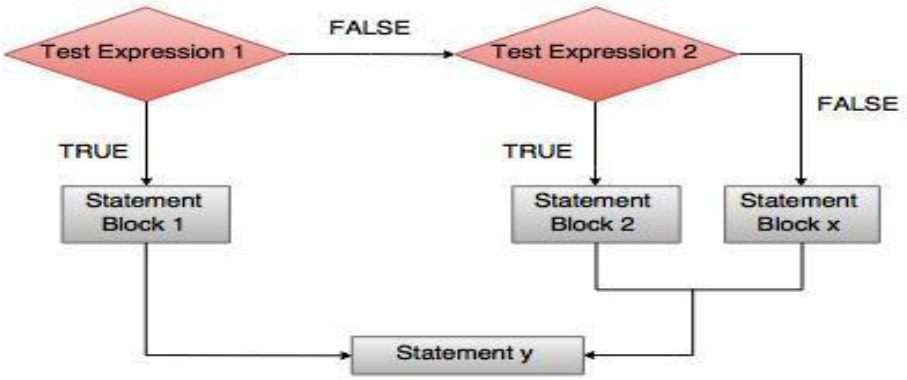
3. if-else-if statement

- This statement works in the normal way as the if statement.
- Its construct is known as the nested if statement.
- After the first if branch the program can have many other else-if branches depending on the expressions that need to be tested.

Syntax:

```
if (test expression 1)
{
   statement block 1;
}
else if (test expression 2)
{
   statement block 2;
}
.....
```

```
else
{
   statement block x;
}
statement y;
```



Example: Program to find largest of the three numbers by using && operator.

```
#include <stdio.h>
void main()
{
    int n1=10, n2=30, n3=75;
    if (n1>n2 && n1>n3)
       printf("%d is the largest number",n1);
    else if(n2>n1 && n2>n3)
       printf("%d is the largest number", n2);
    else
       printf("%d is the largest number", n3);
}
```

Output:
75 is the largest number.

4. C Nested If..else statement

When an if else statement is present inside the body of another “if” or “else” then this is called nested if else.

Syntax of Nested if else statement:

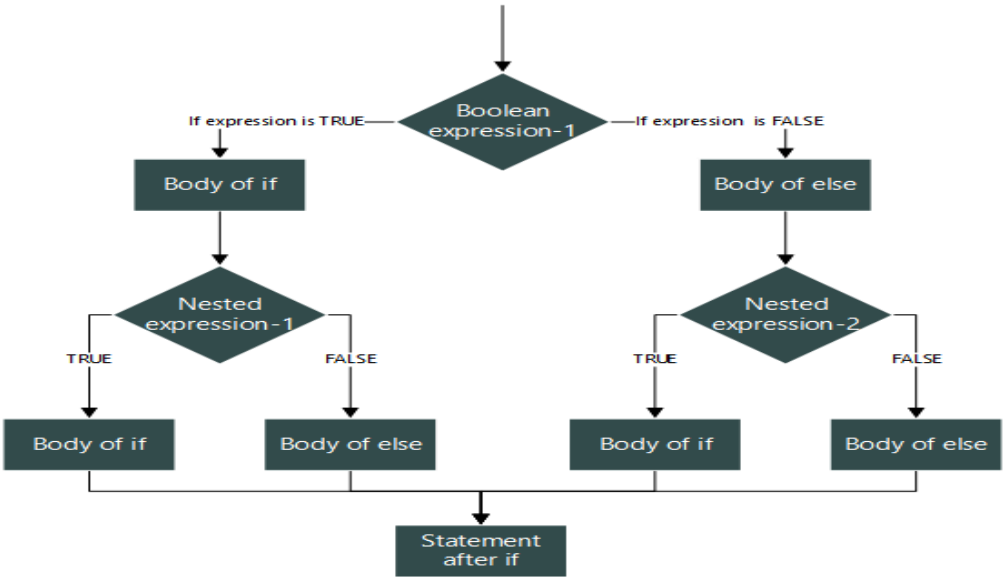
```
if(condition){
//Nested if else inside the body of "if"
if(condition2){
//Statements inside the body of nested "if"
```

```
}
else{
//Statements inside the body of nested "else"
}
}
else{
// Nested if else inside the body of "else"

        if(condition3){
//Statements inside the body of nested "if"
        }
        else{
//Statements inside the body of nested "else"
        }

}

Statement x;
```



Example of nested if..else

```
#include<stdio.h>
intmain()
{
int var1, var2;
printf("Input the value of var1:");
scanf("%d",&var1);
printf("Input the value of var2:");
scanf("%d",&var2);
if(var1 != var2)
{
        printf("var1 is not equal to var2\n");
        //Nested if else
        if(var1 > var2)
        {
```

```
                printf("var1 is greater than var2\n");
        }
        else
        {
                printf("var2 is greater than var1\n");
        }
}
else
{
        printf("var1 is equal to var2\n");
}
return0;
}
```

Output:

```
Input the value of var1:12
Input the value of var2:21
var1 isnot equal to var2
var2 is greater than var1
```

5. Switch case

This case statement is a multi-way decision statement which is a simpler version of the if-else block which evaluates only one variable.

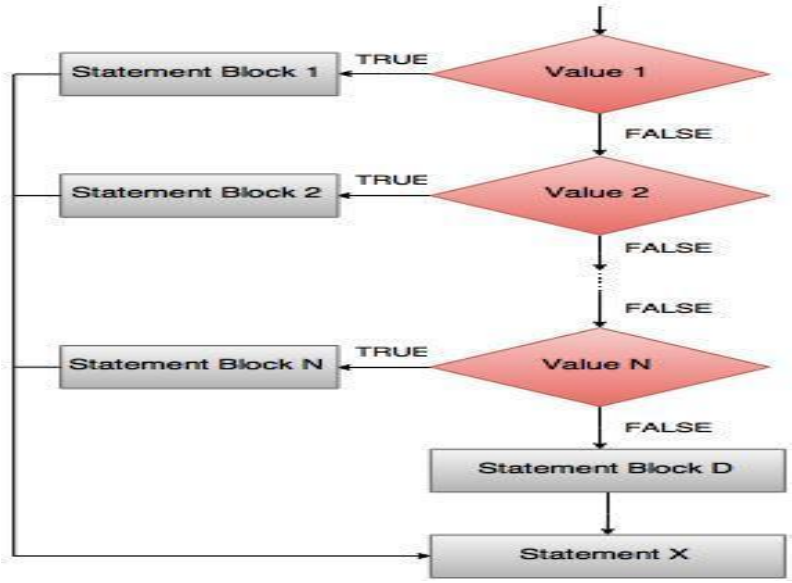
Syntax:

```
switch (expression)
{
    case expression 1: statement 1;
        statement 2;
        .....
        statement n;
        break;

    case expression 2: statement 1;
        statement 2;
        .....
        statement n;
        break;

    default: statement 1;
        statement 2;
        .....
        statement n;
}
```

All the expressions should have a result of an integer value or the character value.



Example: Write a program to check if the character entered is a vowel or not.

```
#include <stdio.h>
void main( )
{
    char ch;
    printf("Enter any character:");
    scanf("%c", &ch);
    switch(ch)
    {
       case 'A':
       case 'a':
       case 'E':
       case 'e':
       case 'I':
       case 'i':
       case 'O':
       case 'o':
       case 'U':
       case 'u':
          printf("%c is a vowel",ch);
          break;
      default:
          printf("%c is not a vowel", ch);
    }
}
```

Output:

Enter any character: c

c is not a vowel.

Iterative Statements:

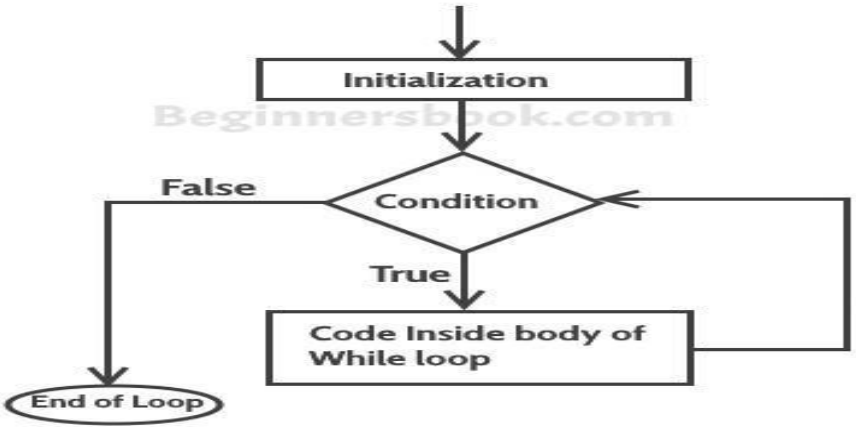
Iteration statements create loops in the program. In other words, it repeats the set of statements until the condition for termination is met. Iteration statements in C are ***for,*** ***while*** and ***do-while***.

1. while statement

The while loop in C is most fundamental loop statement. It repeats a statement or block while its controlling expression is true. Here is its general form:

```
while(condition){
// body of loop
}
```

The condition can be any boolean expression. The body of the loop will be executed as long as the conditional expression is true.




```
//example program to illustrate while looping
#include<stdio.h>

int main ()
{
   int n =10;
   while(n >0)
   {
      printf("tick %d", n);
```

```
        printf("\n");
        n--;
    }
}
```

The ouput of the program is:

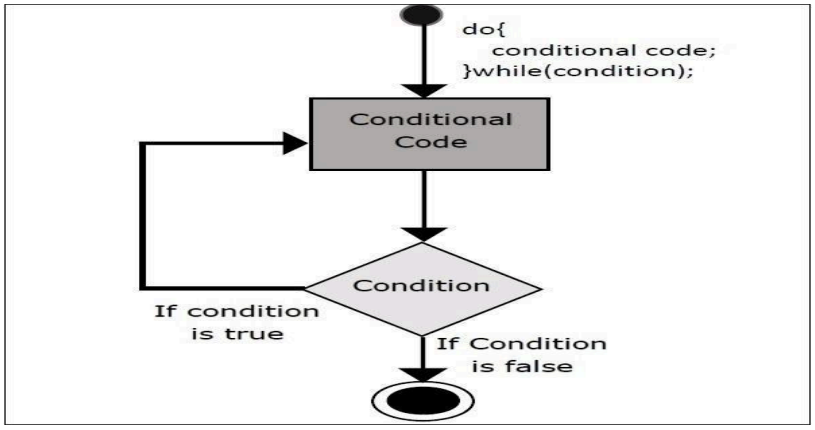
```
tick 10
tick 9
tick 8
tick 7
tick 6
tick 5
tick 4
tick 3
tick 2
tick 1
```

2. do-while statement

The *do-while loop always executes its body at least once*, because its conditional expression is at the bottom of the loop. Its general form is:

```
do{
// body of loop
}while(condition);
```

Each iteration of the do-while loop first executes the body of the loop and then evaluates the conditional expression. If this expression is true, the loop will repeat. Otherwise, the loop terminates. As with all of C's loops, condition must be a Boolean expression.



The program presented in previous while statement can be re-written using do-while as:

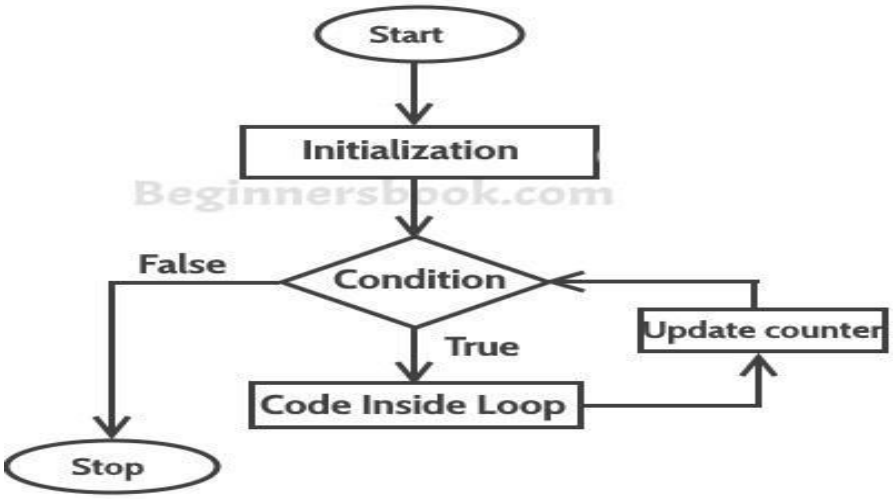
```
//example program to illustrate do-while looping
#include<stdio.h>

int main ()
{
   int n =10;
   do
   {
      printf("tick %d", n);
      printf("\n");
      n--;
   }while(n >0);
}
```

3. for loop :

Here is the general form of the traditional for statement:

```
for(initialization; condition;update_counter){
// body
}
```



The program from previous example can be re-written using for loop as:

```
//example program to illustrate for looping
#include<stdio.h>

int main ()
{
   int n;
   for(n =10; n>0; n--){
      printf("tick %d",n);
      printf("\n");
   }
}
```

The ouput of the program is same as output from program in while loop.

There can be more than one statement in initilaization and iteration section. They must be separated with comma.

Here, we've presented the example to illustrate this.

```
//example program to illustrate more than one statement using the comma
// for looping
#include<stdio.h>

int main ()
```

```
{
   int a, b;
   for(a =1, b =4; a < b; a++, b--)
   {
      printf("a = %d \n", a);
      printf("b = %d \n", b);
   }
}
```

The ouput of the program is:

```
a =1
b =4
a =2
b =3
```

Here, the initialization portion sets the values of both a and b. The two comma separated statements in the iteration portion are executed each time the loop repeats.

Nested Loops :

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.

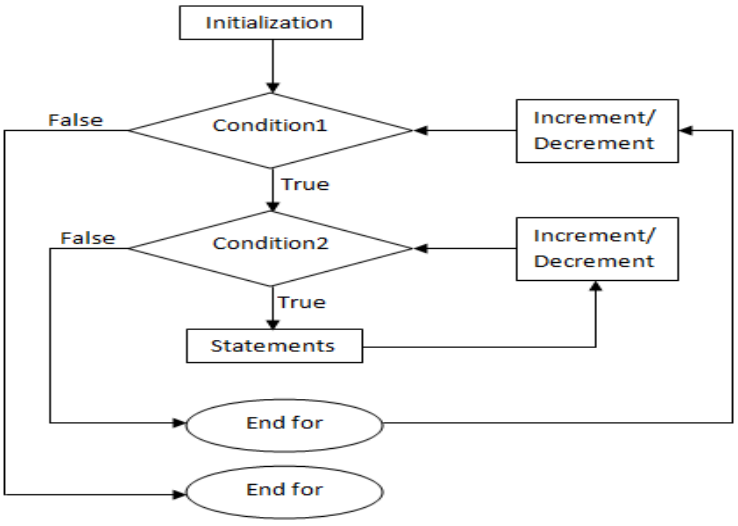


Fig: Flowchart for nested for loop

```
// nested loops
#include<stdio.h>

int main ()
{
   inti, j;
   for(i=0;i<8;i++)
```

```
    {
        for(j =i; j <8; j++)
            printf(".");
        printf("\n");
    }
}
```

Here, two for loops are nested. The number times inner loop iterates depends on the value of in outer loop.

The output of the program is:

```
........
.......
......
.....
....
...
..
.
```

C break and continue statements :

Break :

The break statement ends the loop immediately when it is encountered. Its syntax is:

```
    break;
```

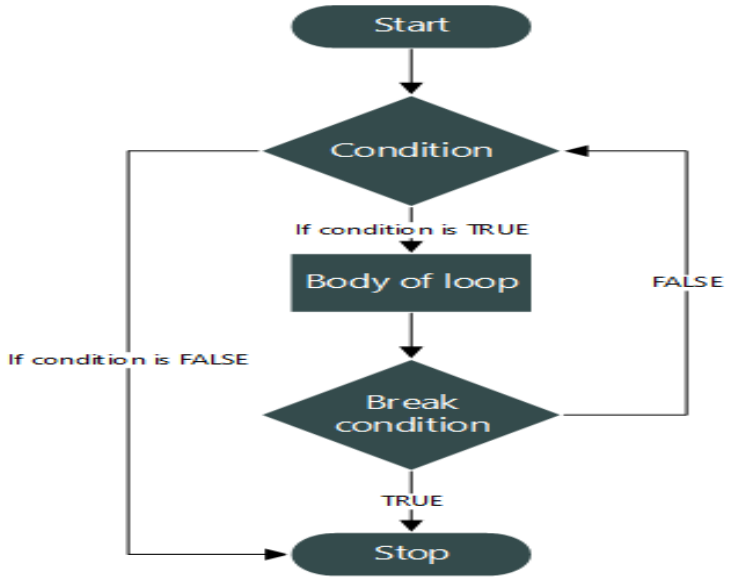
The break statement is almost always used with if...else statement inside the loop.

How break statement works?

```
while (testExpression) {
   // codes
   if (condition to break) {
      break;
   }
   // codes
}
```

```
do {
   // codes
   if (condition to break) {
      break;
   }
   // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
   // codes
   if (condition to break) {
      break;
   }
   // codes
}
```



Example 1: break statement

```
1.  // Program to calculate the sum of a maximum of 10 numbers
2.  // If a negative number is entered, the loop terminates
3.
4.  # include <stdio.h>
5.  intmain()
6.  {
7.  inti;
8.  double number, sum =0.0;
9.
10. for(i=1;i<=10;++i)
11. {
12. printf("Enter a n%d: ",i);
13. scanf("%lf",&number);
14.
15. // If the user enters a negative number, the loop ends
16. if(number <0.0)
17. {
18. break;
19. }
20.
21.         sum += number;// sum = sum + number;
22. }
23.
24. printf("Sum = %.2lf",sum);
25.
26. return0;
27. }
```

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 a maximum of 10 numbers. Why a maximum of 10 numbers? It's because if the user enters a negative number, the `break` statement is executed. This will end the `for` loop, and the `sum` is displayed.

In C, `break` is also used with the `switch` statement.

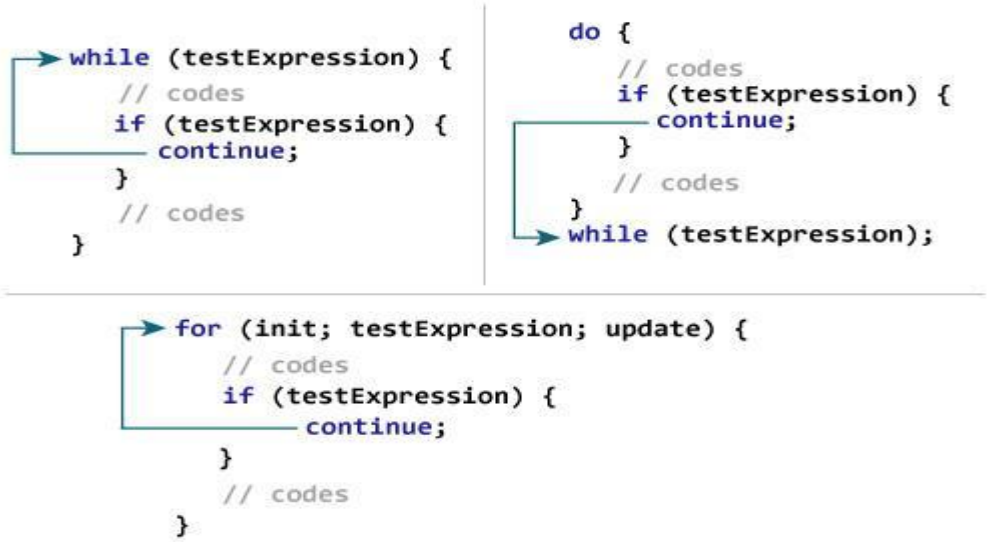
Continue :

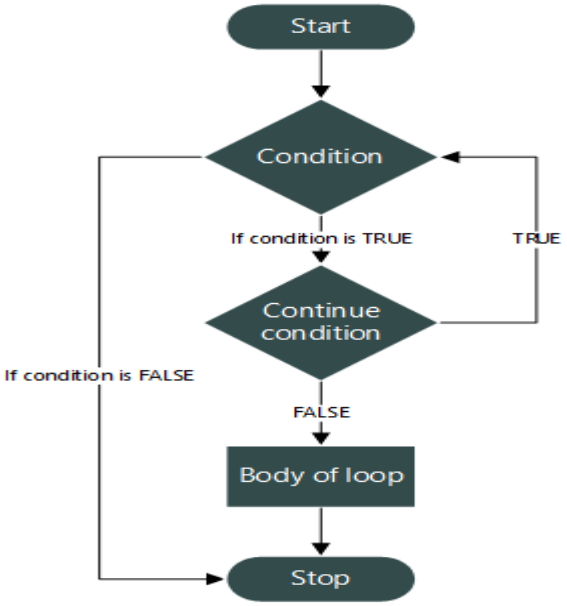
The `continue` statement skips the current iteration of the loop and continues with the next iteration. Its syntax is:

```
continue;
```

The continue statement is almost always used with the if...else statement.

How continue statement works?





Example 2: continue statement

```
1.  // Program to calculate the sum of a maximum of 10 numbers
2.  // Negative numbers are skipped from the calculation
3.
4.  # include <stdio.h>
5.  intmain()
```

```
{
inti;
double number, sum =0.0;

for(i=1;i<=10;++i)
{
printf("Enter a n%d: ",i);
scanf("%lf",&number);

if(number <0.0)
{
continue;
}

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

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

return0;
}
```

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 this program, when the user enters a positive number, the sum is calculated using `sum += number;` statement.

When the user enters a negative number, the `continue` statement is executed and it skips the negative number from the 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.



Example: goto Statement

```
1.
2.  // Program to calculate the sum and average of maximum of 5 numbers
3.  // If user enters negative number, the sum and average of previously entered positive
    numbers are displayed
4.
5.  # include <stdio.h>
6.
7.  intmain()
8.  {
9.
10. constintmaxInput=5;
11. int i;
12. double number, average, sum=0.0;
13.
14. for(i=1;i<=maxInput;++i)
15. {
16. printf("%d. Enter a number: ",i);
17. scanf("%lf",&number);
18.
19. // If user enters negative number, flow of program moves to label jump
20. if(number <0.0)
21. goto jump;
22.
23.       sum += number;// sum = sum+number;
```

```
24. }
25.
26.     jump:
27.
28.     average=sum/(i-1);
29. printf("Sum = %.2f\n", sum);
30. printf("Average = %.2f", average);
31.
32. return0;
33. }
```

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
Average = 5.53
```

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.

C Arrays - Introduction

An array is defined as the collection of similar type of data items stored at contiguous memory locations.

Arrays are the derived data type in C programming language which can store the primitive type of data such as int, char, double, float, etc.

It also has the capability to store the collection of derived data types, such as pointers, structure, etc.

The array is the simplest data structure where each data element can be randomly accessed by using its index number.

C array is beneficial if you have to store similar elements. For example, if we want to store the marks of a student in 6 subjects, then we don't need to define different variables for the marks in the different subject. Instead of that, we can define an array which can store the marks in each subject at the contiguous memory locations.

By using the array, we can access the elements easily. Only a few lines of code are required to access the elements of the array.

Declaration of C Array:

We can declare an array in the c language in the following way.

Syntax:

```
datatype  arrayname[arraysize];
```

Now, let us see the example to declare the array.

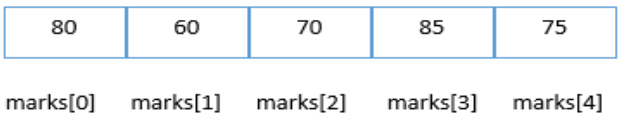
```
int  marks[5];
```

Here int is the *datatype*, marks is the *arrayname*, and 5 is the *arraysize*.

Initialization of C Array

The simplest way to initialize an array is by using the index of each element. We can initialize each element of the array by using the index. Consider the following example.

```
marks[0]=80;//initialization of array
marks[1]=60;
marks[2]=70;
marks[3]=85;
marks[4]=75;
```



Initialization of Array

C Array: Declaration with Initialization :

We can initialize the c array at the time of declaration. Let's see the code.

```
int marks[5]={20,30,40,50,60};
```

In such case, there is **no requirement to define the size**. So it may also be written as the following code.

```
int marks[]={20,30,40,50,60};
```

Accessing elements of the Array:

We can access any array element using array name and subscript/index written inside pair of square brackets [].

For Example:
Suppose we have an integer array of length 5 whose name is marks.

```
int marks[5] = {5,2,9,1,1};
```

Now we can access elements of array marks using subscript followed by array name.

- **marks[0]** = First element of array marks = 5
- **marks[1]** = Second element of array marks = 2
- **marks[2]** = Third element of array marks = 9
- **marks[3]** = Fourth element of array marks = 1
- **marks[4]** = Last element of array marks = 1

Remember array indexing starts from 0. Nth element in array is at index N-1.

Multi dimensional Arrays :

C programming language allows multidimensional arrays. Here is the general form of a multidimensional array declaration −

```
type name[size1][size2]...[sizeN];
```

For example, the following declaration creates a three dimensional integer array −

int threedim[5][10][4];

The simplest form of multidimensional array is the two-dimensional array.

A two-dimensional array is, in essence, a list of one-dimensional arrays.

To declare a two-dimensional integer array of size [x][y], you would write something as follows −

type arrayName [x][y] ;

Where **type** can be any valid C data type and **arrayName** will be a valid C identifier. A two-dimensional array can be considered as a table which will have x number of rows and y number of columns. A two-dimensional array **a**, which contains three rows and four columns can be shown as follows −

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Thus, every element in the array **a** is identified by an element name of the form **a[i][j]**, where 'a' is the name of the array, and 'i' and 'j' are the subscripts that uniquely identify each element in 'a'.

Initializing Two-Dimensional Arrays:

Multidimensional arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row has 4 columns.

```
inta[3][4]={
{0,1,2,3},/*  initializers for row indexed by 0 */
{4,5,6,7},/*  initializers for row indexed by 1 */
{8,9,10,11}/*  initializers for row indexed by 2 */
};
```

The nested braces, which indicate the intended row, are optional. The following initialization is equivalent to the previous example −

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

Accessing Two-Dimensional Array Elements:

An element in a two-dimensional array is accessed by using the subscripts, i.e., row index and column index of the array. For example −

```
int val = a[2][3];
```

The above statement will take the 4th element from the 3rd row of the array. You can verify it in the above figure. Let us check the following program where we have used a nested loop to handle a two-dimensional array –

```
#include<stdio.h>

int main (){

/* an array with 5 rows and 2 columns*/
inta[5][2]={{0,0},{1,2},{2,4},{3,6},{4,8}};
inti, j;

/* output each array element's value */
for(i=0;i<5;i++){

for( j=0; j <2;j++){
printf("a[%d][%d] = %d\n",i,j, a[i][j]);
}
}

return0;
}
```

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

```
a[0][0]: 0
a[0][1]: 0
a[1][0]: 1
a[1][1]: 2
a[2][0]: 2
a[2][1]: 4
a[3][0]: 3
a[3][1]: 6
```

a[4][0]: 4
a[4][1]: 8

As explained above, we can have arrays with any number of dimensions, although it is likely that most of the arrayswe create will be of one or two dimensions.

Operations on Array

Following operations can be performed on arrays:

1. Traversing
2. Searching
3. Insertion
4. Deletion
5. Sorting
6. Merging

1. Traversing: It is used to access each data item exactly once so that it can be processed.
E.g.
We have linear array A as below:

1	2	3	4	5
10	20	30	40	50

Here we will start from beginning and will go till last element and during this process we will access value of each element exactly once as below:

A [1] = 10

A [2] = 20

A [3] = 30

A [4] = 40

A [5] = 50

2. Searching: It is used to find out the location of the data item if it exists in the given collection of data items.

E.g.

We have linear array A as below:

1	2	3	4	5

15	50	35	20	25

Suppose item to be searched is 20. We will start from beginning and will compare 20 with each element. This process will continue until element is found or array is finished. Here:

1) Compare 20 with 15
20 # 15, go to next element.

2) Compare 20 with 50
20 # 50, go to next element.

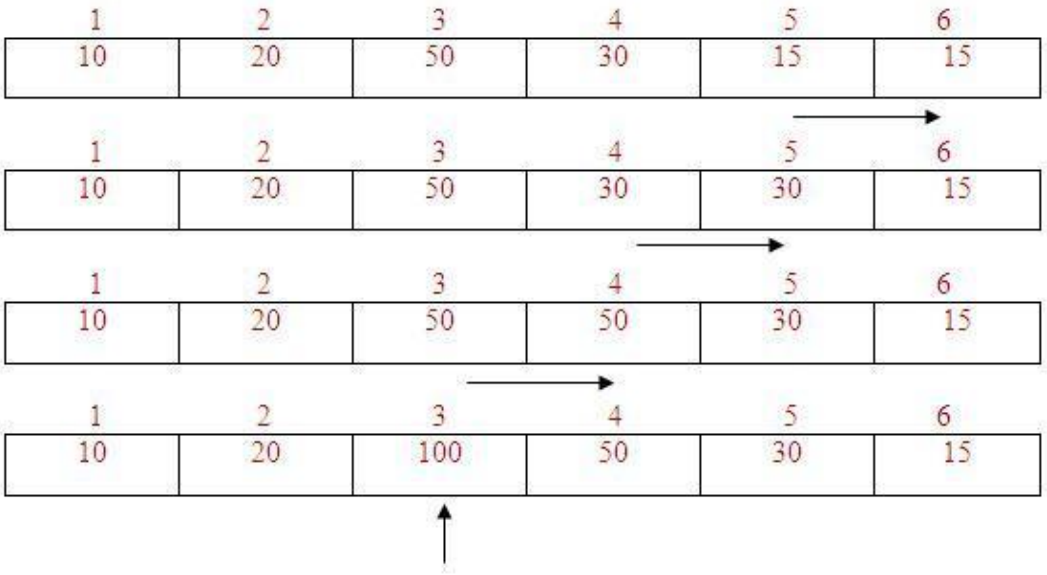
3) Compare 20 with 35
20 #35, go to next element.

4) Compare 20 with 20
20 = 20, so 20 is found and its location is 4.

3. Insertion: It is used to add a new data item in the given collection of data items.
E.g.
We have linear array A as below:

1	2	3	4	5
10	20	50	30	15

New element to be inserted is 100 and location for insertion is 3. So shift the elements from 5th location to 3rd location downwards by 1 place. And then insert 100 at 3rd location. It is shown below:



After deletion the array will be:

1	2	3	4	5	6
10	20	40	25	60	

5. Sorting: It is used to arrange the data items in some order i.e. in ascending or descending order in case of numerical data and in dictionary order in case of alphanumeric data.

E.g.

We have linear array A as below:

1	2	3	4	5
10	50	40	20	30

After arranging the elements in increasing order by using a sorting technique, the array will be:

1	2	3	4	5

10	20	30	40	50

6. Merging: It is used to combine the data items of two sorted files into single file in the sorted form

We have sorted linear array A as below:

1	2	3	4	5	6
10	40	50	80	95	100

And sorted linear array B as below:

1	2	3	4
20	35	45	90

After merging merged array C is as below:

1	2	3	4	5	6	7	8	9	10
10	20	35	40	45	50	80	90	95	100

Strings in C

Strings are defined as an array of characters. The difference between a character array and a string is the string is terminated with a special character '\0'.

Declaration of strings: Declaring a string is as simple as declaring a one dimensional array. Below is the basic syntax for declaring a string.

```
char   str_name[size];
```

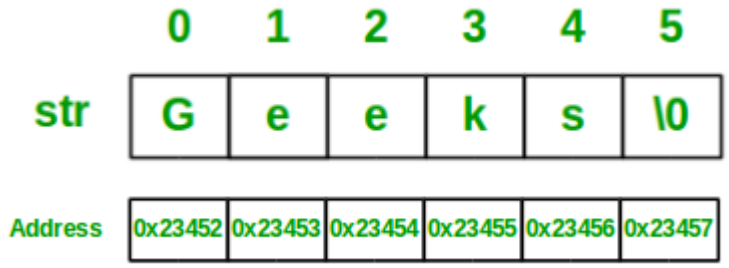
In the above syntax str_name is any name given to the string variable and size is used define the length of the string, i.e the number of characters strings will store. Please keep in mind

that there is an extra terminating character which is the Null character (‘\0’) used to indicate termination of string which differs strings from normal character arrays.

Initializing a String: A string can be initialized in different ways. We will explain this with the help of an example. Below is an example to declare a string with name as str and initialize it with “Geeks”.

Char str[]= “Geeks”;

Below is the memory representation of a string “Geeks”.




```
// C program to illustrate strings
#include<stdio.h>
int main()
{
   // declare and initialize string
   char str[] = "Geeks";
   // print string
   printf("%s",str);

    return0;
}
```

Output:Geeks

We can see in the above program that strings can be printed using a normal printf statements just like we print any other variable. Unlike arrays we do not need to print a string, character by character. The C language does not provide an inbuilt data type for strings but it has an access specifier “**%s**” which can be used to directly print and read strings.

String Handling Functions

1) strlen( ) Function :

strlen( ) function is used to find the length of a character string.

Example: **int n;**

char st[20] = “Bangalore”;

n = strlen(st);

• This will return the length of the string 9 which is assigned to an integer variable n.

• Note that the null character "**\0**" available at the end of a string is not counted.

2) strcpy( ) Function :

strcpy( ) function copies contents of one string into another string. Syntax for strcpy function is given below.

Syntax: **char * strcpy (char * destination, const char * source);**

Example:

strcpy(str1, str2) – It copies contents of str2 into str1.

strcpy(str2, str1) – It copies contents of str1 into str2.

If destination string length is less than source string, entire source string value won't be copied into destination string.

For example, consider destination string length is 20 and source string length is 30. Then, only 20 characters from source string will be copied into destination string and remaining 10 characters won't be copied and will be truncated.

Example : **char city[15];**

strcpy(city, "BANGALORE") ;

This will assign the string "BANGALORE" to the character variable city.

3) strcat( ) Function :

strcat( ) function in C language concatenates two given strings. It concatenates source string at the end of destination string. Syntax for **strcat( )** function is given below.

Syntax : char * strcat (char * destination, const char * source);

Example :

strcat(str2, str1); - str1 is concatenated at the end of str2.

strcat(str1, str2); - str2 is concatenated at the end of str1.

• As you know, each string in C is ended up with null character ('**\0**').

• In **strcat( )** operation, null character of destination string is overwritten by source string's first character and null character is added at the end of new destination string which is created after **strcat( )** operation.

4) Strncat() function :

strncat( ) function in C language concatenates (appends) portion of one string at the end of another string.

Syntax : char * strncat (char * destination, const char * source, size_tnum);

Example :

strncat (str2, str1, 3); – First 3 characters of str1 is concatenated at the end of str2.

strncat (str1, str2, 3); - First 3 characters of str2 is concatenated at the end of str1.

As you know, each string in C is ended up with null character ('**\0**').

In **strncat( )** operation, null character of destination string is overwritten by source string's first character and null character is added at the end of new destination string which is created after **strncat( )** operation.

```
#include<stdio.h>
#include<string.h>

intmain()
{
charsource[]=""  ftl" ;
chartarget[]= "welcome to" ;

printf( "\n Source string =%s", source );
printf( "\n Target string =%s", target );

strncat( target, source,3);

printf( ""\n Target string after strncat()=%s", target );
}
```

Output :

```
Source string = ftl
Target string = welcome to
Target string after strncat()= welcome to ft
```

5) strcmp( ) Function :

strcmp( ) function in C compares two given strings and returns zero if they are same. If length of string1 < string2, it returns < 0 value. If length of string1 > string2, it returns > 0 value.

Syntax : int strcmp (const char * str1, const char * str2);

strcmp( ) function is case sensitive. i.e., "A" and "a" are treated as different characters.

Example :

char city[20] = "Madras";

char town[20] = "Mangalore";

strcmp(city, town);

This will return an integer value "-10" which is the difference in the ASCII values of the first mismatching letters "D" and "N".

* Note that the integer value obtained as the difference may be assigned to an integer variable as follows:

int n;

n = strcmp(city, town);

6) strcmpi() function :

strcmpi( ) function in C is same as strcmp() function. But, strcmpi( ) function is not case sensitive. i.e., "A" and "a" are treated as same characters. Whereas, strcmp() function treats "A" and "a" as different characters.

• **strcmpi()** function is non standard function which may not available in standard library.

• Both functions compare two given strings and returns zero if they are same.

• If length of string1 < string2, it returns < 0 value. If length of string1 > string2, it returns > 0 value.

strcmp( ) function is case sensitive. i.e., "A" and "a" are treated as different characters.

Syntax : int strcmpi (const char * str1, const char * str2);

Example :

m = 0.⇒m=strcmpi(“ DELHI ”, “ delhi ”);

7) strlwr() function :

strlwr() function converts a given string into lowercase.

Syntax : char *strlwr(char *string);

strlwr() function is non standard function which may not available in standard library in C.

```
#include<stdio.h>
#include<string.h>
intmain()
{
charstr[]= “MODIFY This String To Lower”;
printf(“%s\n”,strlwr(str));
return0;
}
```

Output :

```
modify this string to lower
```

8) strupr() function :

strupr() function converts a given string into uppercase.

Syntax : char *strupr(char *string);

strupr() function is non standard function which may not available in standard library in C.

```
#include<stdio.h>
#include<string.h>

intmain()
{
charstr[]= “Modify This String To Upper”;

printf(“%s\n”,strupr(str));

return0;
}
```

Output :

```
MODIFY THIS STRING TO UPPER
```

9) strrev() function :

strrev() function reverses a given string in C language.

Syntax : char *strrev(char *string);

strrev() function is non standard function which may not available in standard library in C.

Example :
char name[20]=”ftl”; then
strrev(name)= ltf

```
#include<stdio.h>
#include<string.h>

intmain()
{
charname[30]= “Hello”;

printf(“String before strrev():%s\n”, name);

printf(“String after strrev():%s”,strrev(name));

return0;
}
```

Output :

```
String before strrev( ) : Hello
String after strrev( ) : olleH
```

IMPORTANT QUESTIONS:

1. Explain Arrays in C with example ?

Arrays:

An array is defined as the collection of similar type of data items stored at contiguous memory locations.

Types of Arrays in C

In c programming language, arrays are classified into **two types**. They are as follows...

1. **Single Dimensional Array / One Dimensional Array**
2. **Multi Dimensional Array**

Single Dimensional Array

In c programming language, single dimensional arrays are used to store list of values of same data type. In single dimensional array, data is stored in linear form. Single dimensional arrays are also called as **one-dimensional arrays**, **Linear Arrays** or simply **1-D Arrays**.

Declaration of Single Dimensional Array

We use the following general syntax for declaring a single dimensional array...

datatype arrayName [size] ;

Example Code

```
int rollNumbers[60] ;
```

The above declaration of single dimensional array reserves 60 continuous memory locations of 2 bytes each with the name **rollNumbers** and tells the compiler to allow only integer values into those memory locations.

Initialization of Single Dimensional Array

We use the following general syntax for declaring and initializing a single dimensional array with size and initial values.

datatype arrayName [size] = {value1, value2, ...} ;

Example Code

```
int marks [6] = { 89, 90, 76, 78, 98, 86 } ;
```

The above declaration of single dimensional array reserves 6 contiguous memory locations of 2 bytes each with the name **marks** and initializes with value 89 in first memory location, 90 in second memory location, 76 in third memory location, 78 in fourth memory location, 98 in fifth memory location and 86 in sixth memory location.

We can also use the following general syntax to intialize a single dimensional array without specifying size and with initial values...

datatype arrayName [ ] = {value1, value2, ...} ;

The array must be initialized if it is created without specifying any size. In this case, the size of the array is decided based on the number of values initialized.

Example Code

```
int marks [] = { 89, 90, 76, 78, 98, 86 } ;
```

Accessing Elements of Single Dimensional Array

We use the following general syntax to access individual elements of single dimensional array...

arrayName [indexValue]

Example Code

```
marks [2] = 99 ;
```

In the above statement, the third element of **'marks'** array is assinged with value **'99'**.

Multi Dimensional Array

An array of arrays is called as multi dimensional array. In simple words, an array created with more than one dimension (size) is called as multi dimensional array.

Multi dimensional array can be of **two dimensional array** or **three dimensional array** or **four dimensional array** or more.

Most popular and commonly used multi dimensional array is **two dimensional array**. The 2-D arrays are used to store data in the form of table. We also use 2-D arrays to create mathematical **matrices**.

Declaration of Two Dimensional Array

We use the following general syntax for declaring a two dimensional array...

datatype arrayName[rowSize] [columnSize] ;

Example Code

```
int matrixA [2][3] ;
```

The above declaration of two dimensional array reserves 6 continuous memory locations of 2 bytes each in the form of **2 rows** and **3 columns**.

Initialization of Two Dimensional Array

We use the following general syntax for declaring and initializing a two dimensional array with specific number of rows and coloumns with initial values.

datatype arrayName [rows][colmns] = { {r1c1value, r1c2value, ...},{r2c1, r2c2,...}...} ;

Example Code

```
int matrixA [2][3] = { {1, 2, 3},{4, 5, 6} } ;
```

We can also initialize as follows...

Example Code

```
int matrixA [2][3] = {
                        {1, 2, 3},
                        {4, 5, 6}
                      } ;
```

Accessing Individual Elements of Two Dimensional Array

We use the following general syntax to access the individual elements of a two-dimensional array...

arrayName [rowIndex] [columnIndex]

Example Code

```
matrixA [0][1] = 10 ;
```

In the above statement, the element with row index 0 and column index 1 of **matrixA** array is assinged with value **10**.

Q2. Explain Strings in C with example ?

Strings in C

Strings are defined as an array of characters.

Declaration of strings:

Declaring a string is as simple as declaring a one dimensional array. Below is the basic syntax for declaring a string.

```
char str_name[size];
```

In the above syntax str_name is any name given to the string variable and size is used to define the length of the string, i.e the number of characters string will store.

Initializing a String:

A string can be initialized in different ways. We will explain this with the help of an example. Below is an example to declare a string with name as str and initialize it with “GeeksforGeeks”.

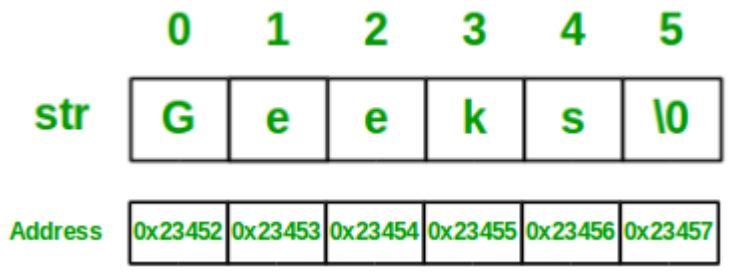
```
1. char str[ ] = "GeeksforGeeks";

2. char str[50] = "GeeksforGeeks";

3. char str[ ] = {'G','e','e','k','s','f','o','r','G','e','e','k','s','\0'};

4. char str[14] = {'G','e','e','k','s','f','o','r','G','e','e','k','s','\0'};
```

Below is the memory representation of a string “Geeks”.



Below is a sample program to read a string from user:

// C program to read and print strings

```
#include<stdio.h>

int main()
{
   // declaring string

   char str[50];
     printf(“Enter a String:”);

   // reading string
   scanf("%s",str);

   // print string
   printf("%s",str);

   return 0;
}
```

INPUT/OUTPUT:
Enter a String: Hello

Hello

You can see in the above program that string can also be read using a single scanf statement.

String Handling Functions In C Language:

1) strlen( ) Function :

strlen( ) function is used to find the length of a character string.

Example:

```
int  n;
char st[20] = “Bangalore”;
n = strlen(st);
```

2) strcpy( ) Function :

Example:

strcpy (str1, str2) – It copies contents of str2 into str1.

strcpy (str2, str1) – It copies contents of str1 into str2.

Example : **char city[15];**

strcpy(city, "BANGALORE") ;

This will assign the string "BANGALORE" to the character variable city.

3) strcat( ) Function :

strcat( ) function in C language concatenates two given strings. It concatenates source string at the end of destination string.

Example :

strcat (str2, str1); - str1 is concatenated at the end of str2.

strcat (str1, str2); - str2 is concatenated at the end of str1.

4) Strncat() function :

strncat( ) function in C language concatenates (appends) portion of one string at the end of another string.

Example :
strncat (str2, str1, 3); – First 3 characters of str1 is concatenated at the end of str2.
strncat (str1, str2, 3); - First 3 characters of str2 is concatenated at the end of str1.

5) strcmp( ) Function :

strcmp( ) function in C compares two given strings and returns zero if they are same. If length of string1 < string2, it returns < 0 value. If length of string1 > string2, it returns > 0 value.

Example :
char city[20] = "Madras";
char town[20] = "Mangalore";
strcmp(city, town);

This will return an integer value "-10" which is the difference in the ASCII values of the first mismatching letters "D" and "N".

6) strcmpi() function :

strcmpi( ) function in C is same as strcmp() function. But, strcmpi( ) function is not case sensitive. i.e., "A" and "a" are treated as same characters. Whereas, strcmp() function treats "A" and "a" as different characters.

Example :

m = 0. ⇒ m=strcmpi(" DELHI ", " delhi ");

7) strlwr() function :

strlwr() function converts a given string into lowercase.

```
Example:
strlwr (str);
```

8) strupr() function :

strupr() function converts a given string into uppercase.

```
Example:
strupr(str);
```

9) strrev() function :

strrev() function reverses a given string in C language.

```
Example:
strrev(name);
```