Back Slash Character Constant (Escape sequence)

In C++ a character constant is a single character enclosed within a pair of single quote (`'`). But there are few special type character constants which start with back slash (\) and hence called **Back Slash Constant**. Back slash constant are known as **Escape Sequence**. Some commonly used back slash constant are listed below:

Back Slash	Description	ASCII Code
`'\a'`	Alert (Bell) character - Makes a beep sound	7
`'\b'`	Back space - Moves the cursor one column to the left	8
`'\n'`	New line - Shifts the cursor to the beginning of next line	10
`'\r'`	Carriage return - Takes the cursor to the beginning of the current line	13
`'\t'`	Tab character - Cursor jump 8 columns to the right	9

These back slash constants are also white space characters also, meaning these back slash constant are not visible on the screen. Back slash constant can be embedded within a string. Examples of back slash constants are given below:

```
void main()
{
   cout<<"MANGAF"<<'\b'<<"KUWAIT\nAHAMDI";
   cout<<"\tFAIPS\t"<<"DPS\n";
   cout<<"FAIPS\rDPS";
}
```

Running of the program produces following output

```
MangaKuwait
AHMADI  FAIPS    DPS
DPSPS
```

Explanation of output

- First `cout` displays `Mangaf` on the screen and then Back space character takes the cursor back one column to left. Therefore `K` of `KUWAIT` replaces `F` of `MANGAF`. New line character takes the cursor to the beginning of the next line. Then AHMADI is displayed in the beginning of the new line
- Second `cout` starts with tab character and display `FAIPS`, from the 9th column. Next tab character takes cursor to the 17th column and displays `DPS`. New line character takes the cursor to the next line.
- Third `cout` display `FAIPS`, then carriage return character takes the cursor to the beginning of the current line. So `DPS` replaces `FAI` of `FAIPS`.

User Defined Constant (Named Constant or Read-Only Identifier)

A variable created in C++ program are also know as **Read-Write** identifier because value can be stored in a variable (**Write**) and value stored in variable can be retrieved (**Read**) for display or for further calculation. An example is given below:

```
int x=5;
cout<<x<<endl;
cin>>x;
cout<<x<<endl;
int y=x*10;
cout<<y<<endl;
```

Variable **x** is created with initial values **5**. Value stored in variables **x** is displayed. New value is inputted in variables **x**. Inputted values stored in variable **x** is displayed. New variable **y** is created storing product of **10** and **x**. Value stored in **y** is displayed.

Sometimes in a C++ program we need to create an identifier whose value can only be **read** and its value cannot be **updated**. To do that we prefix keyword **const** before the data type, it then creates a **Read-Only** identifier, an identifier whose value cannot be altered in a program. **Read-Only** identifiers are also known as **User Defined Constant** or **Named Constant**.

Rule: **const** DataType ConstantName=ConstantValue;

Examples of User Defined Constants are given below:

```
const char BELL='\a';
const int MAX=20;
const double PI=3.14159;
```

Character constant **BELL** stores Bell character. Integer constant **MAX** represents value 20. Floating point constant **PI** has a value **3.14159**.

Normally User Defined Constant names are written in uppercase but that's just the convention. If we try to update value stored in User Defined Constant, compiler flags an error. Examples are given below:

```
const int MAX=20;
const double PI=3.14159;
PI=22.0/7;
cin>>MAX;
```

Following statements will flag syntax errors:
PI=22.0/7;
cin>>MAX;
Because the codes updates value stored in constant identifier.

If data type omitted while creating a User Defined Constant, then by default User Defined Constant becomes an **int** type. Examples are given below to illustrate the concept:

```
const MAX=20;
const PI=3.14159;
cout<<MAX<<endl;
cout<<PI<<endl;
```

Program segment displays **20** and **3**. Since integer part of **3.14159** is stored in **PI**, that is, **3** is stored in **PI**.

Generally a User Defined Constants are created as a **Global** identifier, so that the User Defined Constant can be used through out the program. But it is syntactically correct to create a User Defined Constant which is **Local** to block and in that case the constant identifier can be used inside the block and the block nested below. An example is given below:

```
void area(double rad)
{
   double ca=PI*rad*rad;
   double sa=4*PI*rad*rad;
   cout<<"Circle Area ="<<ca<<endl;
   cout<<"Surface Area="<<sa<<endl;
}
void main()
{
   const double PI=3.14159;
   double radius;
   cout<<"Input Radius? "; cin>>radius;
   double cir=2*PI*radius;
   cout<<"Circumference="<<cir<<endl;
   area(radius);
}
```

Compiler flags syntax errors in
PI*rad*rad
4*PI*rad*rad
Because constant **PI** is a local constant created in the **main**() function.

Macro

Macro is an identifier created by using Compiler Directive `#define`. Macro identifier represents replacement text. A replacement text either represents replacement text (symbolic constant or single line / multi-line code. A programmer can use a Macro identifier instead of User Defined Constant identifier or a programmer can use a Macro in place of user defined function.

Rule for creating Macro as Symbolic Constant:

```
#define MacroName ReplacementText
```

`MacroName`: C++ identifier name
`ReplacementText`: Integer / Character / Floating point / String constant

Usage

```
#define PI 3.14159
#define SIZE 20
```

`#define` is a compiler directive. It tells the compiler that it should replace Macro Identifier Name with Replacement Text (Symbolic Constant Name). A Complete program is given below showing the usage of Macro as a Replacement Text.

```
#include<iostream.h>
#define PI 3.14159
void main()
{
   double rad;
   cout<<"Input Radius? "; cin>>rad;
   double area=PI*rad*rad;
   double volume=4.0/3*PI*rad*rad*rad;
   double sarea=4*PI*rad*rad;
   cout<<"Area of Circle  ="<<area<<endl;
   cout<<"Volume of Sphere="<<volume<<endl;
   cout<<"Surface Area    ="<<sarea<<endl;
}
```

When C++ compiler compiles the program, every occurrence of `PI` is replaced by `3.14159` (Replacement Text / Symbolic Constant). Running of the program produces following output:

```
Input Radius? 7.0
Area of Circle  =153.938
Volume of Sphere=1436.75
Surface Area    = 615.752
```

Differences between Macro Identifier and Constant Identifier are given below:

Macro Identifier	Constant Identifier
• Macro Identifier has **no data type** • **No memory is allocated** to a Macro Identifier • Compiled code **does not contain** Macro Identifier name	• Constant Identifier has a **data type** • **Memory is allocated** to a Constant Identifier • Compiled code **contains** Constant Identifier name

`random(),randomize()`

Built-in function `random()` is used to generate pseudo random integer.

Rule: `IntegerVariable = random(n);`

Function `random()` needs header file `<stdlib.h>`. Return value of the function is **`int`**. Function `random()` will generate a pseudo random integer between `0` and `n-1` where `n` is an integer.

Usage of `random()`

```
#include<iostream.h>
#include<stdlib.h>
void main()
{
   int n1=random(100);
   int n2=random(200);
   cout<<"n1="<<n1<<",n2="<<n2;
}
```

Function **`random(100)`** will generate a random integer between **`0`** and **`99`**. Function **`random(200)`** will generate a random integer between **`0`** and **`199`**. Every time the program is run, it displays the **same** pair of output. If we want different pair of values at different run, then we need function **`randomize`**().

Running of the program three times, produces following outputs:

```
n1=40,n2=20
n1=40,n2=20
n1=40,n2=20
```

Rule: `randomize();`

Function `randomize()` needs header file `<stdlib.h>`. Return value of the function is **`void`** and function has no parameter. Function `randomize()` is used with function `random()`, so that function `random()` will generate different random integer for every run.

Usage of `randomize()`

```
#include<iostream.h>
#include<stdlib.h>
void main()
{
   randomize();
   int n1=random(100);
   int n2=random(200);
   cout<<"n1="<<n1<<",n2="<<n2;
}
```

The program is almost similar to previous program but only difference is addition of function **`randomize`**(). Every time the program is run, it displays the **different** pair of output because of inclusions of function **`randomize`**().

Running the programs two times, produces following outputs:

```
n1=51,n2=129
n1=27,n2=182
```

Function `random()` can generate only non-negative integer values. But kindly note:

- `100+random(100)` will generate random integer between `100` and `199`
- `-random(100)` will generate random integer between `0` and `-99`
- `random(100)/10.0` will generate random floating point value between `0` and `9.9`
- **`char`**`(65+random(26))` will generate random character between `'A'` and `'Z'`
- **`char`**`(97+random(26))` will generate random character between `'a'` and `'z'`

- Program to generate 15 random negative integers:

```
#include<iostream.h>
#include<stdlib.h>
void main()
{
   randomize();
   for (int x=0; x<15; x++)
   {
      int num=-random(100);
      cout<<num<<' ';
   }
}
```

Running of the program produces following output:

```
-53 -24 -78 -97 -41 -42 -20 -73 -91 -67 -98 -37 -34 -55 -97
```

- Program to generate 15 random floating point values:

```
#include<iostream.h>
#include<stdlib.h>
void main()
{
   randomize();
   for (int x=0; x<15; x++)
   {
      int num=10+random(90);
      double val=num/10.0;
      cout<<val<<' ';
   }
}
```

Expression **num**=**10**+**random**(**90**) will store a random integer between **10** and **99** in the variable **num**. Expression **val**=**num**/**10.0** will store a random floating point between **1.0** and **9.9** in the variable **val**.

Running of the program produces following output:

```
8.3 3.7 3.3 3.4 4.6 7.9 5 5.6 5.9 7.9 8.7 7 5.3 2.2 7.6
```

- Program to generate 20 random uppercase characters:

```
#include<iostream.h>
#include<stdlib.h>
void main()
{
   randomize();
   for (int x=0; x<20; x++)
   {
      char ch=char(65+random(26));
      cout<<ch<<' ';
   }
}
```

Expression **65**+**random**(**26**) will generate random integer between **65** and **90** (ASCII codes of uppercase letters). Typecasting the expression into a **char** will generate random uppercase characters.

Running of the program produces following output:

```
Z P J M B R B Q B X H O C Y E K V Y J O
```

Array

An array is special type of variable used to represent many values of the same data type under one name in the computer's main storage (RAM). Till now the variable of the fundamental date type that we have used in our program, stores only one value. An example is given below:

```
double num=6.5;
cout<<num<<endl;
cin>>num;
cout<<num<<endl;
```

Variable **num** is created with initial value **6.5**. Value stored in variable **num** is displayed. New value is inputted in variables **num** and it replaces the old value stored in the variable **num**. A variable of the fundamental type can store only one value at a time.

Array Definition: Array is a **collection** of **homogeneous** data type, stored **under one variable name** and memory is **allocated contiguously**. Collection means many, homogeneous mean similar and contiguous mean a block of continuous memory location (no hole in within the block). Array name is the name of the entire block.

Just like variable of fundamental type, array variable has to be created. Creating an array variable is almost similar to creating a variable of fundamental type. Only difference is, array size has to be specified with square brackets (`[]`) after the array variable name. Creating an array variable is also known as declaring or defining an array.

Rule: `DataType ArrayName[ArraySize];`

`DataType` : `DataType` could be **fundamental** data type or **derived** data type
`ArrayName` : `ArrayName` is the array variable name. It is an identifier name.
`ArraySize` : `ArraySize` is used to specify **number values** to be stored under `ArrayName`. `ArraySize` has to be a **positive integer constant** or a **positive integer expression** involving only **integer constants**.

Examples of **correct** (syntactically correct) array declarations are given below:

```
int roll[10];
char name[20];
double marks[40];
double rates[30];
```

Or,

```
const MAX=20;
int roll[MAX/2];
char name[MAX];
double marks[2*MAX];
double rates[MAX+10];
```

Or,

```
#define MAX 20
int roll[MAX/2];
char name[MAX];
double marks[2*MAX];
double rates[MAX+10];
```

Array size is either:
- Positive integer constant
- Constant identifier name representing positive integer constant
- Expression involving integer constants
- Macro identifier where replacement text is a positive integer constant

Examples of **incorrect** (syntactically incorrect) array declarations are given below:

```
int max = 25, size;
cin>>size;
char s1[size], s2[max];
int c1[0], c2[];
double a1[-20], a2[25.5];
```

- Array size cannot be a variable
- Array size cannot be a negative integer or a floating point value
- Array size cannot be zero or blank

In the computer's main storage an array is represented as contiguous block of memory. Suppose an array is created as:

```
double marks[10];
```

We can visualize the array `marks[]` may be represented in the computer's main storage as shown in the figure given below:

0	1	2	3	4	5	6	7	8	9

Each component of an array is called an **element** (**cell**) of an array. Since array name represents many values, C++ compiler assigns a **reference** number to every element of an array. The reference number is called an **index** (**subscript**) of an array. In C++, array index starts from zero (0). First element of the array is the 0th element of an array. Index of the last element of array is calculated as array size – 1. For example if array size is 10 then index of 10th element is 9. Array index is a non-negative integer starting from 0. Every element of an array is called **subscripted** variable. Every element of an array is treated as a variable. Examples are given below:

```
int roll[10];
char name[20];
double marks[40];
```

- ✓ Array `roll[]` is an array of integers and therefore every element of array `roll[]` will be treated like an integer type variable.
- ✓ Array `name[]` is an array of characters and therefore every element of `name[]` will be treated like a character variable.
- ✓ Array `marks[]` is array of floating point values and therefore every element of `marks[]` will be treated like a floating point variable.

After the creation of the array, we need to access elements of an array either to **store value** in elements of an array or to **read** value stored in elements of an array.

Rule to access an element of an array: `ArrayName[Index]`

Usage:

```
double marks[5];
marks[0]
marks[1]
marks[2]
marks[3]
marks[4]
```

`double marks[5]` → Array **`marks`** is created
`marks[0]` → Represents the 1st element of **`marks`**
`marks[1]` → Represents the 2nd element of **`marks`**
`marks[2]` → Represents the 3rd element of **`marks`**
`marks[3]` → Represents the 4th element of **`marks`**
`marks[4]` → Represents the 5th element of **`marks`**

There are several ways to store values in an array. One way to store values in the array is by using assignment operator. Values are to be assigned (stored) to every element of the array. An example is given below:

```
double marks[5];
marks[0]=76.5;
marks[1]=82.0;
marks[2]=69.5;
marks[3]=84.5;
marks[4]=59.0;
```

But generally values are stored in an array by inputting values from a keyboard. An example is given below:

```
double marks[5];
cout<<"Marks of Subject 1? "; cin>>marks[0];
cout<<"Marks of Subject 2? "; cin>>marks[1];
cout<<"Marks of Subject 3? "; cin>>marks[2];
cout<<"Marks of Subject 4? "; cin>>marks[3];
cout<<"Marks of Subject 5? "; cin>>marks[4];
```

Above method of inputting values in an array is only suitable when numbers of elements in array are relatively less. But more elegant way of inputting values in any array will be by using a loop. An example is given below:

```
double marks[5];
for (int k=0; k<5; k++)
{
   cout<<"Marks of Subject "<<(k+1)<<"? ";
   cin>>marks[k];
}
```

Incorrect way of inputting values in an array:

```
double marks[5];
cout<<"Input Marks? ";
cin>>marks;
```

cin>>arrayname will flag syntax error in most arrays (syntax error will not be flagged if **arrayname** is an array of **char**).

Values stored in elements of an array can be displayed by using `cout`. An example is given below:

```
double marks[5];
for (int k=0; k<5; k++)
{
   cout<<"Marks of Subject "<<(k+1)<<"? "; cin>>marks[k];
}
cout<<"Marks of Subject 1="<<marks[0]<<endl;
cout<<"Marks of Subject 2="<<marks[1]<<endl;
cout<<"Marks of Subject 3="<<marks[2]<<endl;
cout<<"Marks of Subject 4="<<marks[3]<<endl;
cout<<"Marks of Subject 5="<<marks[4]<<endl;
```

Again this way of displaying an array is suitable if the array size is small. But a smart way of displaying values stored in an array is by using a loop. A complete program is given below showing inputting values in an array and displaying values stored in an array by using a loop:

```
#include<iostream.h>
void main()
{
   double a[10];
   for(int x=0; x<10; x++)
   {
      cout<<"Value? ";
      cin>>a[x];
   }
   for(int k=0; k<10; k++)
      cout<<a[k]<<endl;
}
```

```
#include<iostream.h>
const MAX=10;
void main()
{
   int a[MAX];
   for(int x=0; x<MAX; x++)
   {
      cout<<"Value? "; cin>>a[x];
   }
   for(int k=0; k<MAX; k++)
      cout<<a[k]<<endl;
}
```

Example is given below showing incorrect way of displaying an array on the screen.

```
double a[5];
for(int x=0; x<5; x++)
{
   cout<<"Input value? "; cin>>a[x];
}
cout<<a;
```

`cout<<marks;` (generally `cout<<arrayname;`) is a syntactically correct statement. But `cout<<arrayname;` will not display values stored in the array `a[]`, instead it will display address of `arrayname`. Again this is true for most arrays. If the `arrayname` is an array of **`char`**, then `cout<<arrayname;` will display the content of the array (display the string on the screen). Address of a variable, string and array of **`char`** will be discussed later.

Initializer

Initializer is another way to assign values to elements of an array. An initializer can only be used with an array, **when the array is getting created**. After the creation of an array we cannot use an initializer to assigns values to elements of an array. An example is given below:

```
double a[5]={11.1, 33.3, 77.7, 22.2, 55.5};
for(int x=0; x<5; x++)
   cout<<a[x]<<endl;
```

Running of the program segment produces following output:

```
11.1
33.3
77.7
22.2
55.5
```

In the above example, the number of values inside the initializer is equal to size of the array. The first value in the inside the initializer is assigned to the first element in the array, the second value inside the initializer is assigned to the second element in the array …, the last value inside the initializer is assigned to the last element in the array.

Bur what happens when values inside the initializer are either less or more than size of the array? If the values inside the initializer are **less** than the size of the array, then the **remaining** elements of the array will be assigned the value 0 (zero).

```
double arr[5]={11.1, 33.3, 22.2};
for(int x=0; x<5; x++)
   cout<<ar[x]<< endl;
```

Running of the program segment produces following output:

```
11.1
33.3
22.2
0
0
```

Remaining two elements of array `arr[]`, `arr[3]` and `arr[4]` are assigned the value 0 (zero). **An initializer must contain at least one value**. Statement given below will flag error:

```
int arr[10]={};                          //Syntax Error
```

If the values inside the initializer are **more** than the size of the array, then the compiler will flag syntax error. An example is given below:

```
double arr[5]={1.1,3.3,2.2,5.5,4.4,6.6};      //Syntax Error
```

Array size is 5 but number of values inside the initializer is 6, compiler will flag syntax error. When an array is assigned values using an initializer, array size may be omitted from array declaration (definition). In that case number of values inside the initializer decides the size of the array. Array without a size is called **open** array. Some examples are given below:

```
int arr1[]={33.3, 22.2, 55.5, 44.4, 77.7, 66.6, 99.9, 88.8};
double arr2[]={30, 20, 50, 40, 60};
```

Array `arr1[]` will be created with size 8 since inside the initializer there are 8 values. Array `arr2[]` will be created with size 5 since inside the initializer there are 5 values.

Built-in functions `random()` and `randomize()` can be used to assign random values to elements of an array. Some examples are given below:

```
//Array of random integers
#include<iostream.h>
#include<stdlib.h>
void main()
{
   int a[10];
   randomize();
   for(int x=0; x<10; x++)
      a[x]=random(100);
   for(int k=0; k<10; k++)
      cout<<a[k]<<endl;
}
```

```
//Array of random integers
#include<iostream.h>
#include<stdlib.h>
const MAX=10;
void main()
{
   int a[MAX];
   randomize();
   for(int x=0; x<MAX; x++)
      a[x]=random(100);
   for(int k=0; k<MAX; k++)
      cout<<a[k]<<endl;
}
```

```
//Array of random double values
#include<iostream.h>
#include<stdlib.h>
const MAX=20;
void main()
{
   double a[MAX];
   randomize();
   for(int x=0; x<MAX; x++)
      a[x]=random(1000)/10.0;
   for(int k=0; k<MAX; k++)
      cout<<a[k]<<endl;
}
```

```
//Array of random characters
#include<iostream.h>
#include<stdlib.h>
const MAX=40;
void main()
{
   char a[MAX];
   randomize();
   for(int x=0; x<MAX; x++)
      a[x]=char(97+random(26));
   for(int k=0; k<MAX; k++)
      cout<<a[k];
}
```

Array as Global variable and array as Local variable

An array created just after the header file and before any function block is called a **Global** array. A Global array can be used throughout the program and in every block of the program. An array created in block is called a **Local** array. A Local array can be used inside the block in which it was created and blocks nested below. Inside a block, if a **Local** array and a **Global** array have same name, then **scope resolution operator** (::) is to be used with the **Global** array name, so that Global array and Local array can be used inside the block. An example of **Global** array and **Local** array is given below:

```
#include<iostream.h>
#include<stdlib.h>
int ga[10];
void arrfn()
{
   int la1[10];
   for (int x=0; x<10; x++)
   {
      ga[x]=random(100);
      la1[x]=random(100);
   }
   for (int k=0; k<10; k++)
      cout<<ga[k]<<la1[k]<<endl;
}
void main()
{
   int la2[10];
   randomize();
   for (int x=0; x<10; x++)
   {
      ga[x]=random(200);
      la2[x]=random(200);
   }
   for (int k=0; k<10; k++)
      cout<<ga[k]<<la2[k]<<endl;
}
```

```
#include<iostream.h>
#include<stdlib.h>
double ga[10];
void arrfn()
{
   double la1[10];
   for (int x=0; x<10; x++)
   {
      ga[x]=random(100)/10.0
      la1[x]=random(100)/10.0;
   }
   for (int k=0; k<10; k++)
      cout<<ga[k]<<la1[k]<<endl;
}
void main()
{
   double la2[10];
   randomize();
   for (int x=0; x<10; x++)
   {
      ga[x]=random(200)/5.0;
      la2[x]=random(200)/5.0;
   }
   for (int k=0; k<10; k++)
      cout<<ga[k]<<la2[k]<<endl;
}
```

Array `ga[]` is **Global** array it can be used throughout the program, that is, array `ga[]` can be used in the `main()` function and in the function `arrfn()`. Arrays `la1[]` and `la2[]` are **Local** arrays. Array `la1[]` is local to function `arrfn()` and array `la2[]` is local to `main()` function.

Array as parameter to a user defined function

Just like any other data type, array type can be passed as parameter to a function. There are several ways of passing an array as parameter to a function. **Array is always passed by reference to a function**. The simplest way to pass an array as a parameter to function is to pass the **array name** as a parameter to the function. Function invocation will have function name and array name as parameter. Function header will have array name and the data type of the array as a parameter. Examples are given below to illustrate the concept:

```
const MAX=20;
void input(int a[])
{
   for(int x=0; x<MAX; x++)
      a[x]=random(100);
}
Or,
void input(int a[])
{
   for(int x=0; x<MAX; x++)
   {
      cout<<"Value? ";cin>>a[x];
   }
}
void main()
{
   int arr[MAX];
   randomize();
   input(arr);
   for(int k=0; k<MAX; k++)
      cout<<arr[k]<<endl;
}
```

```
const MAX=20;
void input(double a[])
{
   for(int x=0; x<MAX; x++)
      a[x]=random(200)/5.0;
}
Or,
void input(double a[])
{
   for(int x=0; x<MAX; x++)
   {
      cout<<"Value?";cin>>a[x];
   }
}
void main()
{
   double arr[MAX];
   randomize();
   input(arr);
   for(int k=0; k<MAX; k++)
      cout<<arr[k]<<endl;
}
```

```
void input(int a[])
{
   for(int x=0; x<20; x++)
      a[x]=random(100);
}
Or,
void input(int a[])
{
   for(int x=0; x<MAX; x++)
   {
      cout<<"Value?";cin>>a[x];
   }
}
void main()
{
   int a1[20], a2[20];
   randomize();
   input(a1);
   input(a2);
   for(int k=0; k<20; k++)
      cout<<a1[k]<<a2[k]<<endl;
}
```

```
void input(double a[])
{
   for(int x=0; x<20; x++)
      a[x]=random(200)/5.0;
}
Or,
void input(double a[])
{
   for(int x=0; x<20; x++)
   {
      cout<<"Value?";cin>>a[x];
   }
}
void main()
{
   double a1[20], a2[20];
   randomize();
   input(a1);
   input(a2);
   for(int k=0; k<20; k++)
      cout<<a1[k]<<a2[k]<<endl;
}
```

Function input() is used to store values in the array a[] (a[] is the formal parameter) by either using random() or taking input from keyboard. Function definition of input() has only one formal parameter – array name (a[]) and data type of the array (**int** / **double**). An open array (a[]) is used to represent array as formal parameter. Size of the array is either user defined constant MAX or integer constant 20. When invoking function input() actual parameter is array name (arr – is the actual parameter). This way passing array name as a parameter to a function is suitable when the program deals with either one single array (array arr[] in the first example) or many arrays having same size (array a1[] and a2[] in the second example).

But when the program has to deal with many arrays of different sizes then we have a problem because in the user defined function we cannot use a constant as an array size. In such cases user defined function needs two parameters – array name (with data type) and size of the array (number elements in the array). Function header will have two parameters – array name (and data type of the array) and number of elements (integer type). Invocation of function will have two actual parameters – array name and number elements in the array. Some examples are given below:

```
const MAX=20;
void input(int a[], int n)
{
   for(int x=0; x<n; x++)
      a[x]=random(100);
}
Or,
void input(int a[], int n)
{
   for(int x=0; x<MAX; x++)
   {
      cout<<"Value?";cin>>a[x];
   }
}
void display(int a[], int n)
{
   for(int x=0; x<n; x++)
      cout<<a[x]<<' ';
   cout<<endl;
}
void main()
{
   int arr1[MAX], arr2[30];
   randomize();
   input(arr1, MAX);
   input(arr2, 30);
   display(arr1, MAX);
   display(arr2, 30);
}
```

```
const MAX=20;
void input(double a[], int n)
{
   for(int x=0; x<n; x++)
      a[x]=random(200)/5.0;
}
Or,
void input(double a[], int n)
{
   for(int x=0; x<MAX; x++)
   {
      cout<<"Value?";cin>>a[x];
   }
}
void display(double a[], int n)
{
   for(int x=0; x<n; x++)
      cout<<a[x]<<' ';
   cout<<endl;
}
void main()
{
   double arr1[MAX], arr2[30];
   randomize();
   input(arr1, MAX);
   input(arr2, 30);
   display(arr1, MAX);
   display(arr2, 30);
}
```

User defined functions input() and display() have two formal parameters – array name (**int** / **double** a[]) and number of elements in the array (**int** n). Invocation of user defined functions input() and display() also have two actual parameters – array name (arr1 / arr2) and number of elements in the array (MAX / 30). This is the best way of passing array as parameter to a function since function can accept arrays of different sizes.

Program to find sum and average of values stored in an array.

```
#include<iostream.h>
#include<stdlib.h>
const MAX=20;
void sumavg(int ar[], int n)
{
   int s=0;
   for(int k=0; k<n; k++)
      s+=ar[k];
   double av=s/double(n);
   cout<<"Sum="<<s<<endl;
   cout<<"Average="<<av<<endl;
}
void main()
{
   int arr[MAX];
   randomize();
   for(int x=0; x<MAX; x++)
      arr[x]=random(1000);
   sumavg(arr, MAX);
   for(int k=0; k<MAX; k++)
      cout<<arr[k]<<endl;
}
```

```
#include<iostream.h>
#include<stdlib.h>
const MAX=20;
void sumavg(double ar[], int n)
{
   double s=0;
   for(int k=0; k<n; k++)
      s+=ar[k];
   double av=s/n;
   cout<<"Sum="<<s<<endl;
   cout<<"Average="<<av<<endl;
}
void main()
{
   double arr[MAX];
   randomize();
   for(int x=0; x<MAX; x++)
      arr[x]=random(2000)/5.0;
   sumavg(arr, MAX);
   for(int k=0; k<MAX; k++)
      cout<<arr[k]<<endl;
}
```

Program to find GM and HM of values stored in an array.

```
#include<iostream.h>
#include<stdlib.h>
#include<math.h>
const MAX=20;
void gmhm(int ar[], int n)
{
   double s=0, p=1;
   for(int k=0; k<n; k++)
   {
      p*=ar[k]; s+=1.0/ar[k]
   }
   double gm=pow(p, 1.0/n);
   double hm=n/s;
   cout<<"GM="<<gm<<endl;
   cout<<"HM="<<hm<<endl;
}
void main()
{
   int arr[MAX];
   randomize();
   for(int x=0; x<MAX; x++)
      arr[x]=100+random(900);
   gmhm(arr, MAX);
   for(int k=0; k<MAX; k++)
      cout<<arr[k]<<endl;
}
```

```
#include<iostream.h>
#include<stdlib.h>
#include<math.h>
const MAX=20;
void gmhm(double ar[], int n)
{
   double s=0, p=1;
   for(int k=0; k<n; k++)
   {
      p*=ar[k]; s+=1/ar[k]
   }
   double gm=pow(p, 1.0/n);
   double hm=n/s;
   cout<<"GM="<<gm<<endl;
   cout<<"HM="<<hm<<endl;
}
void main()
{
   double arr[MAX];
   randomize();
   for(int x=0; x<MAX; x++)
      arr[x]=(10+random(90))/5.0;
   gmhm(arr, MAX);
   for(int k=0; k<MAX; k++)
      cout<<arr[k]<<endl;
}
```

Program to find maximum and minimum values stored in an array.

```
#include<iostream.h>
#include<stdlib.h>
void maxmin(int ar[], int n)
{
   int hi=ar[0], lo=ar[0];
   for(int k=0; k<n; k++)
      if (ar[k]<lo)
         lo=ar[k];
   for(int x=0; x<n; x++)
      if (ar[x]>hi)
         hi=ar[x];
   cout<<"Min="<<lo<<endl;
   cout<<"Max="<<hi<<endl;
}
void main()
{
   int arr[MAX];
   randomize();
   for(int x=0; x<MAX; x++)
      arr[x]=100+random(900);
   maxmin(arr, MAX);
   for(int k=0; k<MAX; k++)
      cout<<arr[k]<<endl;
}
```

```
#include<iostream.h>
#include<stdlib.h>
void maxmin(double ar[], int n)
{
   double hi=ar[0], lo=ar[0];
   for(int k=0; k<n; k++)
      if (ar[k]<lo)
         lo=ar[k];
   for(int x=0; x<n; x++)
      if (ar[x]>hi)
         hi=ar[x];
   cout<<"Min="<<lo<<endl;
   cout<<"Max="<<hi<<endl;
}
void main()
{
   double arr[MAX];
   randomize();
   for(int x=0; x<MAX; x++)
      arr[x]=(10+random(90))/5.0;
   maxmin(arr, MAX);
   for(int k=0; k<MAX; k++)
      cout<<arr[k]<<endl;
}
```

Program to find minimum value stored in an array and its position in the array.

```
#include<iostream.h>
#include<stdlib.h>
void arrmin(int ar[], int n)
{
   int lo=ar[0];
   int pos=0;
   for(int k=0; k<n; k++)
      if (ar[k]<lo)
      {
         lo=ar[k]; pos=k;
      }
   cout<<"Min="<<lo<<endl;
   cout<<"Position="<<pos<<endl;
}
void main()
{
   int arr[MAX];
   randomize();
   for(int x=0; x<MAX; x++)
      arr[x]=100+random(900);
   arrmin(arr, MAX);
   for(int k=0; k<MAX; k++)
      cout<<arr[k]<<endl;
}
```

```
#include<iostream.h>
#include<stdlib.h>
void arrmin(double ar[], int n)
{
   double lo=ar[0];
   int pos=0;
   for(int k=0; k<n; k++)
      if (ar[k]<lo)
      {
         lo=ar[k]; pos=k;
      }
   cout<<"Min="<<lo<<endl;
   cout<<"Position="<<pos<<endl;
}
void main()
{
   double arr[MAX];
   randomize();
   for(int x=0; x<MAX; x++)
      arr[x]=(10+random(90))/5.0;
   arrmin(arr, MAX);
   for(int k=0; k<MAX; k++)
      cout<<arr[k]<<endl;
}
```

Program to reverse elements of an array

```
#include<iostream.h>
#include<stdlib.h>
void reverse(int ar[], int n)
{
   int ri=n-1;
   for(int le=0; le<ri; le++)
   {
      int t=ar[le];
      ar[le]=ar[ri];
      ar[ri]=t;
      ri--;
   }
}
void main()
{
   int arr[MAX];
   randomize();
   for(int x=0; x<MAX; x++)
      arr[x]=100+random(900);
   cout<<"Before reversing\n";
   for(int k=0; k<MAX; k++)
      cout<<arr[k]<<endl;
   reverse(arr, MAX);
   cout<<"After reversing\n";
   for(int k=0; k<MAX; k++)
      cout<<arr[k]<<endl;
}
```

```
#include<iostream.h>
#include<stdlib.h>
void reverse(double ar[], int n)
{
   int ri=n-1;
   for(int le=0; le<ri; le++)
   {
      double t=ar[le];
      ar[le]=ar[ri];
      ar[ri]=t;
      ri--;
   }
}
void main()
{
   double arr[MAX];
   randomize();
   for(int x=0; x<MAX; x++)
      arr[x]=random(2000)/5.0;
   cout<<"Before reversing\n";
   for(int k=0; k<MAX; k++)
      cout<<arr[k]<<endl;
   reverse(arr, MAX);
   cout<<"After reversing\n";
   for(int k=0; k<MAX; k++)
      cout<<arr[k]<<endl;
}
```

Program displays Prime numbers stored in an array

```
void arrprime(int ar[], int n)
{
   for (int x=0; x<n; x++)
   {
      int num=ar[x], k=2, prime=1;
      while (k<num && prime==1)
         if (num%k==0)
            pr=0;
         else
            k++;
      prime==1? cout<<num<<" Prime\n" : cout<<num<<endl;
   }
}
void main()
{
   int arr[MAX];
   randomize();
   for(int x=0; x<MAX; x++)
      arr[x]=100+random(900);
   arrprime(arr, MAX);
}
```

Program Counts Odd/Even integers, find sum of Odd/Even integers and calculates average of Odd/Even integers of an array.

```
#include<iostream.h>
#include<stdlib.h>
void oddeven(int ar[], int n)
{
   int ecount=0, ocount=0, esum=0, osum=0;
   for(int k=0; k<n; k++)
      if (ar[k]%2==0)
      {
         ecount++;
         esum+=ar[k];
      }
      else
      {
         ocount++;
         osum+=ar[k];
      }
   double eavg=esum/double(ecount), oavg=osum/double(ocount);
   cout<<"Number of Even Integers ="<<ecount<<endl;
   cout<<"Number of Odd Integers  ="<<ocount<<endl;
   cout<<"Sum of Even Integers    ="<<esum<<endl;
   cout<<"Sum of Odd Integers     ="<<esum<<endl;
   cout<<"Average of Even Integers="<<eavg<<endl;
   cout<<"Average of Odd Integers ="<<eavg<<endl;
}
void main()
{
   int arr[MAX];
   randomize();
   for(int x=0; x<MAX; x++)
      arr[x]=100+random(900);
   oddeven(arr, MAX);
   for(int k=0; k<MAX; k++)
      cout<<arr[k]<<endl;
}
```

String Variable – Array of Characters

String constant is a sequence of characters enclosed within a pair of double quotes ("). To store a string in computer's main memory we need a string variable. In C++ an array of characters is generally called a string variable. But every array of characters is not a string variable. A string variable is an array of characters terminated by a **Nul** character (`'\0'`). ASCII code of **Nul** character is 0 (zero). Normally creating a string variable simply means creating an array of characters. Examples are given below:

```
char myname[20];
char address[80];

const MAX=20;
char myname[MAX];
char address[4*MAX];
```

Arrays **myname** and **address** are array of characters but they are not a string variable since they are not terminated by nul character.

Initializer: An initializer can be used to assign a value to a string. Initializer is used with a string variable to assign value when string variable is being created. If number of characters inside the initializer is less than the array size, then the remaining elements of the string variable are assigned **Nul** character. If number of characters inside initializer is more than the size of the array, then compiler will flag error. Examples are given below:

```
char str1[10]={'A', 'H', 'M', 'A', 'D', 'I'};
char str2[10]="AHMADI";
char str3[]={'A', 'H', 'M', 'A', 'D', 'I', '\0'};
char str4[]="AHMADI";
```

Strings `str1` and `str2` will store `"AHMADI"` in computer's main storage in the following way:

0	1	2	3	4	5	6	7	8	9
A	H	M	A	D	I	\0	\0	\0	\0

All elements after `'I'` will have Nul character. String `st1` is initialized using an initializer but string `str2` is initialized by assigning a string constant. Assigning a string constant to initialize a string is always preferred method as compared to using an initializer. Strings `str3` and `str4` will store `"AHMADI"` in computer's main storage in the following way:

0	1	2	3	4	5	6
A	H	M	A	D	I	\0

After `'I'` (5th character) there will be terminating **Nul** character (6th character). Size of strings `str3` will be 6 since there are 6 characters inside the initializer. Size of string `str4` will be 6 as well even though string constant `"AHMADI"` has only 5 characters. This is because one more extra element is needed to store the terminating **Nul** character.

String Input: Input from the keyboard is the most common way of storing value in a string variable. As mentioned earlier `cin>>ArrayName;` where `ArrayName` **is not an array of `char`**, will be flagged as syntax error. But if `ArrayName` is an array of **`char`** then it is possible to input a string from a keyboard. An example is given below:

```
char name[15];
cout<<"Input name? "; cin>>name;
```

Running of the above program segment produces screen like:

```
Input name? ARINDAM↵
```

String constant `"ARINDAM"` get stored in the string variable `name` in computer's main storage in the following way:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	R	I	N	D	A	M	\0	?	?	?	?	?	?	?

First `7` (seven) elements of string variable `name` will contain `"ARINDAM"` and there will be a terminating **Nul** character after `'M'`. Remaining elements of string variable `name` will contain garbage character (`'?'` represent garbage character).

```
char name[15];
cout<<"Input name? "; cin>>name;
```

Running of the above program segment produces screen like:

```
Input name? ARINDAM GUPTA↵
```

But string constant "ARINDAM" only get stored in the string variable name. This is because cin **treats space as a separator**. So according to cin, "ARINDAM" and "GUPTA" are two pieces of data. Therefore it is not possible to input a string containing space using cin. If we want to input a string containing space then we have to use function gets() from the header file <stdio.h>.

Rule for using gets():

```
gets(StringVar);
```

Usage:

```
char name[15];
cout<<"Input name? "; gets(name);
```

Running of the above program segment produces screen like:

```
Input name? ARINDAM GUPTA↵
```

String constant "ARINDAM GUPTA" will be stored in the string variable name in computer's main storage in the following way:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	R	I	N	D	A	M		G	U	P	T	A	\0	?

First 13 (thirteen) characters of string variable name will contain string constant "ARINDAM GUPTA". 14th character will be a terminating **Nul** character. Function gets() can be used to input a string without space as well.

String Copy: Assignment operator (=) is used to copy value to a variable of fundamental type. Assigning value to a variable can be interpreted as **copying** value to a variable. Assignment operator cannot be used to copy a string. To copy a string we need a function strcpy(). Function strcpy() requires header file <string.h>. Example of strcpy() is given below:

```
char song[15];
strcpy(song, "NOVEMBER RAIN");
cout<<song;

char name[15]="NOVEMBER RAIN", song[15];
strcpy(song, name);
cout<<song;
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
N	O	V	E	M	B	E	R		R	A	I	N	\0	?

Function strcpy() will copy string constant "NOVEMBER RAIN" into a string variable song and **Nul** character will be appended after 'N' (13th character of the string variable song) .

String Display

As discussed earlier, a loop is used to display values stored in an array. Also we mentioned ealier that `cout<<ArrayName;` if `ArrayName`, where `ArrayName` **is not an array of `char`**, will display **address** of `ArrayName`. But if `ArrayName` is an array of **`char`** then it will display string value stored in the `ArrayName`. Function `puts()` also displays a string and it requires header file <stdio.h>. There is no added advantage of using `puts()` to display a string as compared to `cout`. When a string is displayed using `cout` (`puts()`), `cout` (`puts()`) assumes that the string terminated by a **Nul** character. An example is given below:

```
char song[15], str[4]="ABCD";
cout<<"Song Name? "; gets(song);
cout<<"Song Name: "<<song<<endl;
cout<<"Song Name: ";
puts(song);
puts(str);
```

Running of the program segment produces following screen

```
Song Name? NOVEMBER RAIN
Song Name: NOVEMBER RAIN
Song Name: NOVEMBER RAIN
ABCD╕ ~↕
```

Last `puts(str)` displays `ABCD╕ ~↕` since string variable `str` has 4 elements, initializer has 4 characters, hence no room for the terminating **Nul** character. So `puts(str)` displays `ABCD`, four characters of the string then look for terminating **Nul** character. But `puts()` could not find **Nul** character after `D` and therefore displays few garbage characters (`╕ ~↕`), till it finds the **Nul** character. There are **two** minor differences between `cout` and `puts()`.

`cout`	`puts()`
• `cout` is an object (variable) and it needs header file <`iostream.h`>	• `puts()` is a function and it needs header file <`stdio.h`>
• `cout` does not add a new line character, cursor remains on the same line	• `puts()` adds a new line character, cursor is shifted to the beginning of the next line

Important string functions are given below:

Function Name	Header File	Usage
`gets(s)`	`stdio.h`	Input string `s` from keyboard, with or without space
`puts(s)`	`stdio.h`	Displays string `s` on the screen, assumes that string is terminated by **Nul** character
`strlen(s)`	`string.h`	Counts number of characters in string `s` excluding Nul character.
`strcpy(s1,s2)`	`string.h`	String `s2` is copied to string `s1`
`strupr(s)`	`string.h`	Converts string `s` into uppercase
`strlwr(s)`	`string.h`	Converts string `s` into lowercase
`strrev(s)`	`string.h`	Reverses a string `s`, string DOWN becomes NWOD
`strcat(s1,s2)`	`string.h`	Appends (concatenate) string `s2` after string `s1`
`strcmp(s1,s2)`	`string.h`	Compares 2 strings `s1` and `s2` lexically using ASCII code
`stricmp(s1,s2)` `strcmpi(s1,s2)`	`string.h`	Compares 2 strings `s1` and `s2` lexically using ASCII code but ignoring case

C++ functions which can be used in place of functions of header file <`string.h`>:

```
int stringlength(char str[])
{
   int len=0;
   while (str[len])
      len++;
   return len;
}
void stringcopy(char des[], char src[])
{
   int x;
   for (x=0; src[x]; x++)
      des[x]=src[x];
   des[x]='\0';
}
void stringupper(char str[])
{
   for(int x=0; str[x]; x++)
      if (str[x]>='a' && str[x]<='z')
         str[x]=char(str[x]-32);
}
void stringlower(char str[])
{
   for(int x=0; str[x]; x++)
      if (str[x]>='A' && str[x]<='Z')
         str[x]=char(str[x]+32);
}
void stringconcat(char des[], char src[])
{
   int x=0, k=0;
   while (des[x])
      x++;
   while (src[k])
      des[x++]=src[k++];
   des[x]='\0';
}
void stringreverse(char str[])
{
   int len=0;
   while (str[len])
      len++;
   int left=0, right=len-1;
   while (left<right)
   {
      char t=str[left];
      str[left]=str[right];
      str[right]=t;
      left++; right--;
   }
}
```

Or,

```
void stringreverse(char str[])
```

```
{
   int len=0;
   while (str[len])
      len++;
   for (int x=0; x<len/2; x++)
   {
      char t=str[x];
      str[x]=str[len-x-1];
      str[len-x-1]=t;
   }
}
int stringcompare(char s1[], char s2[])
{
   int x=0;
   while (s1[x] && s2[x] && s1[x]==s2[x])
      x++;
   return s1[x]-s2[x];
}
```

Some useful C++ string functions:

```
void stringtoggle(char str[])
{
   for(int x=0; str[x]; x++)
      if (str[x]>='A' && str[x]<='Z')
         str[x]=char(str[x]+32);
      else
      if (str[x]>='a' && str[x]<='z')
         str[x]=char(str[x]-32);
}
int chkpalindrome(char str[])
{
   int len=0;
   while (str[len])
      len++;
   int left=0, right=len-1, palin=1;
   while (left<right && palin==1)
      if (str[left]==str[right])
      {
         left++; right--;
      }
      else
         palin=0;
   return palin;
}
```

Or,

```
int chkpalindrome(char str[])
{
   int len=0;
   while (str[len])
      len++;
   int left=0, right=len-1, palin=1;
```

```
   while (left<right && palin==1)
      if (str[left]!=str[right])
         palin=0;
      else
      {
         left++;
         right--;
      }
   return palin;
}
```

Or,

```
int chkpalindrome(char str[])
{
   int len=0;
   while (str[len])
      len++;
   int left=0, right=len-1, palin=1;
   while (left<right && palin==1)
      if (str[left++]!=str[right--])
         palin=0;
   return palin;
}
```

Or,

```
int chkpalindrome(char str[])
{
   int len=0, palin=1
   while (str[len])
      len++;
   for (int x=0; x<len/2 && palin==1; x++)
      if (str[x]!=str[len-x-1])
         palin=0;
   return palin;
}
void countchars(char str[])
{
   int u=0, l=0, d=0, s=0
   for (int x=0; str[x]; x++)
      if (str[x]>='A' && str[x]<='Z')
         u++;
      else
      if (str[x]>='a' && str[x]<='z')
         l++;
      else
      if (str[x]>='0' && str[x]<='9')
         d++;
      else
         s++;
      cout<<"Uppercase="<<u<<endl<<"Lowercase="<<l<<endl;
      cout<<"Digits="<<u<<endl<<"Special Characters="<<s<<endl;
}
```

C++ array (array of `int/double`) functions (sort, search, merge, insert & delete):

```
void bubblesort(int arr[], int n)
{
   for (int x=1; x<n; x++)
      for (int k=0; k<n-x; k++)
         if (arr[k]>arr[k+1])
         {
            int temp=arr[k];
            arr[k]=arr[k+1];
            arr[k+1]=temp;
         }
}
void bubblesort(double arr[], int n)
{
   for (int x=1; x<n; x++)
      for (int k=0; k<n-x; k++)
         if (arr[k]>arr[k+1])
         {
            double temp=arr[k];
            arr[k]=arr[k+1];
            arr[k+1]=temp;
         }
}
void insertionsort(int arr[], int n)
{
   for(int k=1; k<n; k++)
   {
      int t=arr[k];
      int x=k-1;
      while(x>=0 && t<arr[x])
      {
         arr[x+1]=arr[x];
         x--;
      }
      arr[x+1]=t;
   }
}
void insertionsort(double arr[], int n)
{
   for(int k=1; k<n; k++)
   {
      double t=arr[k];
      int x=k-1;
      while(x>=0 && t<arr[x])
      {
         arr[x+1]=arr[x];
         x--;
      }
      arr[x+1]=t;
   }
}
```

```
void selectionsort(int arr[], int n)
{
   for (int x=0; x<n-1; x++)
   {
      int min=arr[x], pos=x;
      for (int k=x+1; k<n; k++)
         if (arr[k]<min)
         {
            min=arr[k];
            pos=k;
         }
      arr[pos]=arr[x];
      arr[x]=min;
   }
}
void selectionsort(double arr[], int n)
{
   for(int x=0; x<n-1; x++)
   {
      double min=arr[x];
      int pos=x;
      for (int k=x+1; k<n; k++)
         if (arr[k]<min)
         {
            min=arr[k];
            pos=k;
         }
      arr[pos]=arr[x];
      arr[x]=min;
   }
}
int binarysearch(int arr[], int n, int item)
{
   int lb=0, ub=n-1, mid, found=0;
   while (lb<=ub && found==0)
   {
      mid=(lb+ub)/2;
      if (item<arr[mid])
         ub=mid-1;
      else
      if (item>arr[mid])
         lb=mid+1;
      else
         found=1;
   }
   return found;
}
```

Or,

```
void binarysearch(int arr[], int n, int item)
{
   int lb=0, ub=n-1, mid, found=0;
```

```
   while (lb<=ub && found==0)
   {
      mid=(lb+ub)/2;
      if (item<arr[mid])
         ub=mid-1;
      else
      if (item>arr[mid])
         lb=mid+1;
      else
         found=1;
   }
   if (found==1)
      cout<<item<<" found in the array\n";
   else
      cout<<item<<" not found in the array\n";
}
```

Or,

```
void binarysearch(int arr[], int n)
{
   int lb=0, ub=n-1, mid, found=0, item;
   cout<<"Item to search? "; cin>>item;
   while (lb<=ub && found==0)
   {
      mid=(lb+ub)/2;
      if (item<arr[mid])
         ub=mid-1;
      else
      if (item>arr[mid])
         lb=mid+1;
      else
         found=1;
   }
   if (found==1)
      cout<<item<<" found in the array\n";
   else
      cout<<item<<" not found in the array\n";
}
int linearsearch(int arr[], int n, int item)
{
   int k=0, found=0;
   while (k<n && found==0)
      if (item==arr[k])
         found=1;
      else
         k++;
   return found;
}
```

Or,

```
void linearsearch(int arr[], int n, int item)
{
   int k=0, found=0;
```

```
   while (k<n && found==0)
      if (item==arr[k])
         found=1;
      else
         k++;
   if (found==1)
      cout<<item<<" found in the array\n";
   else
      cout<<item<<" not found in the array\n";
}
```

Or,

```
void linearsearch(int arr[], int n)
{
   int k=0, found=0, item;
   cout<<"Item to search? "; gets(item);
   while (k<n && found==0)
      if (item==arr[k])
         found=1;
      else
         k++;
   if (found==1)
      cout<<item<<" found in the array\n";
   else
      cout<<item<<" not found in the array\n";
}
void merge(int a[], int b[], int c[], int n1, int n2)
{
   int i=0, j=0, k=0;
   while (i<n1 && j<n2)
      if (a[i]<b[j])
         c[k++]=a[i++];
      else
         c[k++]=b[j++];
   while (i<n1)
      c[k++]=a[i++];
   while (j<n2)
      c[k++]=b[j++];
}
void merge(double a[], double b[], double c[], int n1, int n2)
{
   int i=0, j=0, k=0;
   while (i<n1 && j<n2)
      if (a[i]<b[j])
         c[k++]=a[i++];
      else
         c[k++]=b[j++];
   while (i<n1)
      c[k++]=a[i++];
   while (j<n2)
      c[k++]=b[j++];
}
```

```
void arrayins(int arr[], int& n, int pos, int item)
{
   if (n==MAX)       //Size of array is MAX
      cout<<"Overflow\n";
   else
   {
      for (int x=n-1; x>=pos; x--)
         arr[x+1]=arr[x];
      arr[pos]=item;
      n++;
      cout<<item<<" inserted in the array\n";
   }
}
void arrayins(double arr[], int& n, int pos, double item)
{
   if (n==MAX)       //Size of array is MAX
      cout<<"Overflow\n";
   else
   {
      for (int x=n-1; x>=pos; x--)
         arr[x+1]=arr[x];
      arr[pos]=item;
      n++;
      cout<<item<<" inserted in the array\n";
   }
}
void arraydel(int arr[], int& n, int pos)
{
   if (n==0)
      cout<<"Underflow\n";
   else
   {
      cout<<arr[pos]<<" deleted from the array\n";
      for (int x=pos+1; x<n; x++)
         arr[x-1]=arr[x];
      n--;
   }
}
void arraydel(double arr[], int& n, int pos)
{
   if (n==0)
      cout<<"Underflow\n";
   else
   {
      cout<<arr[pos]<<" deleted from the array\n";
      for (int x=pos+1; x<n; x++)
         arr[x-1]=arr[x];
      n--;
   }
}
```

Keyword typedef

Keyword **typedef** is used create a user defined data type. Use of keyword **typedef** is explained with examples given below:

```
//Without typedef
int mark[5];
char name[20]
cout<<"Name?";
gets(name);
for (int x=0; x<5; x++)
{
   cout<<"Mark?";
   cin>>mark[x];
}
```

```
//With typedef
typedef int marklist[5];
typedef char string[20];
marklist mark;
string name;
cout<<"Name?"; gets(name);
for (int x=0; x<5; x++)
{
   cout<<"Mark?";
   cin>>mark[x];
}
```

Statement **typedef int** `marklist[5];` creates a data type called `marklist` which is an array of integers. Statement **typedef char** `string[20];` creates a data type called `string` which is an array of characters. A variable `mark` (an array of integer) is created of the type `marklist` and a variable `name` (an array of character) is created of the type `string`. Data type `marklist` and `name` are the user defined data type created by keyword **typedef**.

C++ array (array of string) functions (sort, search, merge, insert & delete):

```
typedef char string[20];
void bubblesort(string arr[], int n)
{
   for (int x=1; x<n; x++)
      for (int k=0; k<n-x; k++)
         if (strcmp(arr[k], arr[k+1])>0)
         {
            string t;
            strcpy(t, arr[k]);
            strcpy(arr[k], arr[k+1]);
            strcpy(arr[k+1], t);
         }
}
void selectionsort(string arr[], int n)
{
   for (int x=0; x<n-1; x++)
   {
      string min;
      strcpy(min,arr[x]);
      int pos=x;
      for (int k=x+1; k<n; k++)
         if (strcmp(min, arr[k])>0)
         {
            strcpy(min, arr[k]);
            pos=k;
         }
      strcpy(arr[pos], arr[x]);
      strcpy(arr[x], min);
   }
}
```

```
void insertionsort(string arr[], int n)
{
   for (int x=1; x<n; x++)
   {
      string t;
      strcpy(t,arr[x]);
      int k=x-1;
      while (k>=0 && strcmp(t, arr[k])<0)
      {
         strcpy(arr[k+1], arr[k]);
         k--;
      }
      strcpy(arr[k+1], t);
   }
}
int binarysearch(string arr[], int n, string item)
{
   int lb=0, ub=n-1, mid, found=0;
   while (lb<=ub && found==0)
   {
      mid=(lb+ub)/2;
      if (strcmp(item, arr[mid])<0)
         ub=mid-1;
      else
      if (strcmp(item, arr[mid])>0)
         lb=mid+1;
      else
         found=1;
   }
   return found;
}
```

Or,

```
void binarysearch(string arr[], int n, string item)
{
   int lb=0, ub=n-1, mid, found=0;
   while (lb<=ub && found==0)
   {
      mid=(lb+ub)/2;
      if (strcmp(item, arr[mid])<0)
         ub=mid-1;
      else
      if (strcmp(item, arr[mid])>0)
         lb=mid+1;
      else
         found=1;
   }
   if (found==1)
      cout<<item<<" found in the array\n";
   else
      cout<<item<<" not found in the array\n";
}
```

Or,

```
void binarysearch(string arr[], int n)
{
   int lb=0, ub=n-1, mid, found=0;
   string item;
   cout<<"Item to search? "; gets(item);
   while (lb<=ub && found==0)
   {
      mid=(lb+ub)/2;
      if (strcmp(item, arr[mid])<0)
         ub=mid-1;
      else
      if (strcmp(item, arr[mid])>0)
         lb=mid+1;
      else
         found=1;
   }
   if (found==1)
      cout<<item<<" found in the array\n";
   else
      cout<<item<<" not found in the array\n";
}
int linearsearch(string arr[], int n, string item)
{
   int k=0, found=0;
   while (k<n && found==0)
      if (strcmp(item, arr[k])==0)
         found=1;
      else
         k++;
   return found;
}
```

Or,

```
void linearsearch(string arr[], int n, string item)
{
   int k=0, found=0;
   while (k<n && found==0)
      if (strcmp(item, arr[k])==0)
         found=1;
      else
         k++;
   if (found==1)
      cout<<item<<" found in the array\n";
   else
      cout<<item<<" not found in the array\n";
}
```

Or,

```
void linearsearch(string arr[], int n)
{
   int k=0, found=0;
   string item;
```

```
   cout<<"Item to search? "; gets(item);
   while (k<n && found==0)
      if (strcmp(item, arr[k])==0)
         found=1;
      else
         k++;
   if (found==1)
      cout<<item<<" found in the array\n";
   else
      cout<<item<<" not found in the array\n";
}
void merge(string a[], string b[], string c[], int n1, int n2)
{
   int i=0, j=0, k=0;
   while (i<n1 && j<n2)
      if (strcmp(a[i], b[j])<0)
         strcpy(c[k++], a[i++]);
      else
         strcpy(c[k++], b[j++]);
   while (i<n1)
      strcpy(c[k++], a[i++]);
   while (j<n2)
      strcpy(c[k++], b[j++]);
}
void arrayins(string arr[], int& n, int pos, string item)
{
   if (n==MAX)          //Size of array is MAX
      cout<<"Overflow\n";
   else
   {
      for (int x=n-1; x>=pos; x--)
         strcpy(arr[x+1], arr[x]);
      strcpy(arr[pos], item);
      n++;
      cout<<item<<" inserted in the array\n";
   }
}
void arraydel(string arr[], int& n, int pos)
{
   if (n==0)
      cout<<"Underflow\n";
   else
   {
      cout<<arr[pos]<<" deleted from the array\n";
      for (int x=pos+1; x<n; x++)
         strcpy(arr[x-1], arr[x]);
      n--;
   }
}
```