UNIT II ARRAYS AND STRINGS

Introduction to Arrays: Declaration, Initialization – One dimensional array – Two dimensional arrays - String operations: length, compare, concatenate, copy – Selection sort, linear and binary search.

Array

Meaning : An *array* is a collection of variables of the same data type that are referred to through a common name. A specific element in an array is accessed by an index / subscript. Each subscript must be expressed as nonnegative integer. An array is allocated contiguously in memory, and cannot be empty.

In real life array is used, List of employees in the organization.
List of products and their cost sold by store
List of customers and their telephone numbers.

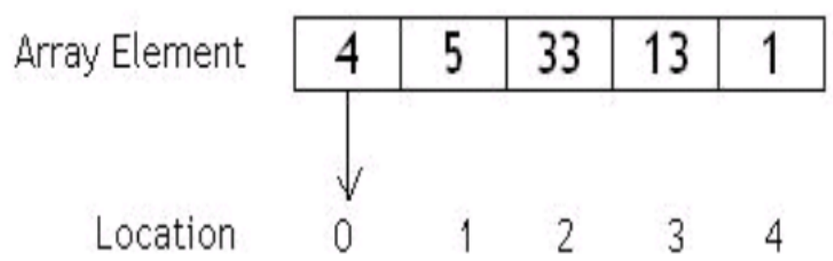
Example of an Array:
Suppose we have to store the roll numbers of the 100 students the we have to declare 100
variables named as roll1, roll2, roll3, ……. roll100 which is very difficult job.

Format

```
Storage-Class  data-type  arrayname
```

```
static   int      x[100];
         char     name[20];
```

Storage class refers to the storage class of the array. It is optional. Default values are automatic for arrays within a function..
Data-type is the type of data
Array is the array name.
Expression is a positive-valued integer expression.



The above array is declared as int a [5];

a[0] = 4; a[1] = 5; a[2] = 33; a[3] = 13; a[4] = 1;

In the above figure 4, 5, 33, 13, 1 are actual data items. 0, 1, 2, 3, 4 are index variables.

1. Individual data items can be accessed by the name of the array and an integer enclosed
in square bracket called subscript variable / index.
2. Subscript Variables helps us to identify the item number to be accessed in the contiguous
memory.

Defining an array

- ✓ Arrays are defined in the same manner, as ordinary variables except that array name must be accompanied by a size specification.
- ✓ The size determines the number of elements in the array.
- ✓ The size must be a positive integer enclosed in square brackets.
  int marks[6];

Characteristics /Features of an array:

1. All the elements of an array share the same name and they are distinguished from one
another with the help of the element number.
2. The element number in an array plays a major role for calling each element.
3. Any particular element of an array can be modified separately without disturbing the
other elements.
4. Any element of an array a[ ] can be assigned or equated to another ordinary variable or
array variable of its type.
5. Array elements are stored in contiguous memory locations.

Array Declaration:

Array is nothing but the collection of elements of similar data types.
To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows:

Syntax:

```
type arrayName [ arraySize ];
```

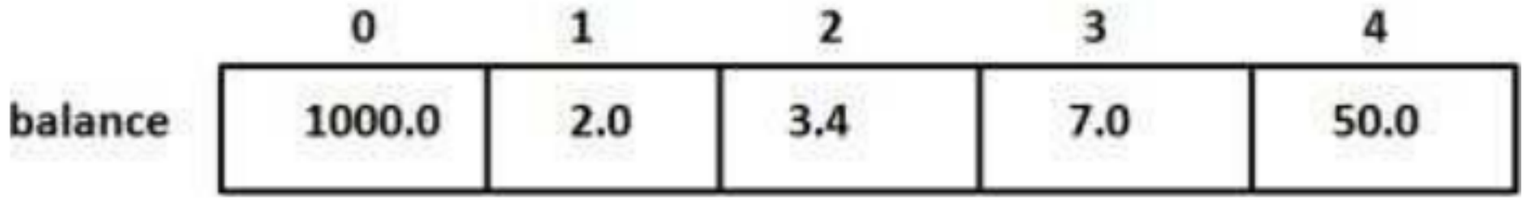
If declare in multi dimensional array,
Syntax: <data type> array name [size1][size2].....[sizen];

Array Declaration tell to Compiler as:

1. Type of the Array
2. Name of the Array
3. Number of Dimension
4. Number of Elements in Each Dimension

Array Initialization

You can initialize array in C either one by one or using a single statement as follows:
double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
The number of values between braces { } can not be larger than the number of elements that we declare for the array between square brackets [ ].



Types of Arrays

The number of subscripts determines the dimensionality of the array.
1. One dimensional array
2. Two-dimensional arrays
3. Multidimensional arrays

1. One dimensional Arrays

A list of items can be given one variable using only one subscript and such a variable is called a single-subscripted variable or one-dimensional array.

Declaration of One dimensional array

Like any other variable, arrays must be declared before they are used. The general form of array declaration is

type variable-name[size];

float height[50];

- ✓ Type specifies the type of element that will be contained in the array such as int, float or char and Size indicates the maximum number of elements that can be stored inside the array.
- ✓ Any reference to the arrays outside the declared limits produce unpredictable results.
- ✓ The size should be either a numeric constant or symbolic constant.

char x[10];

declares the name as a character array (string) variable, can hold a maximum of 10 characters.

Each character of string is treated as an element of the array X and stored in the memory.

‘w’
‘e’
‘l’
‘l’
‘ ’
‘d’
‘o’
‘n’
‘e’
‘\0’

When the compiler sees a character string, it terminates it with an additional null character. The element X [10] holds the null character ‘\0’.

Initialisation of One-dimensional array

Arrays are initialized with a brace-enclosed list of constant expressions.

An array can be initialized at either of the following stages.

a) At Compile Time b) At Run Time.

i) Compile Time Initialisation

We can initialize the elements of arrays in the same way as the ordinary variables when they are declared.

General format

type arrayname [size] = { List of values } ;

int num [3] = { 0,0,0 } ; /* declares the variable ‘num’ as an array of size 3 and

will assign zero to each element. */

float total[5]= { 0.0, 15.75, -10 }; /* initializes first 3 elements and the remaining 2 elements to zero. */

c) int x[ ] = { 25, 40, 50 }; /* the size may be omitted. In such cases, the compiler allocates enough space for all initialized

elements. */

char name [ ] = { ‘J’, ‘o’, ‘h’, ‘n’, ‘\0’ }; /* declares the name to be an array of 5

characters with null character */

d) char name[ ] = “John”; /* can assign the string literal directly */

ii)Run Time Initialisation

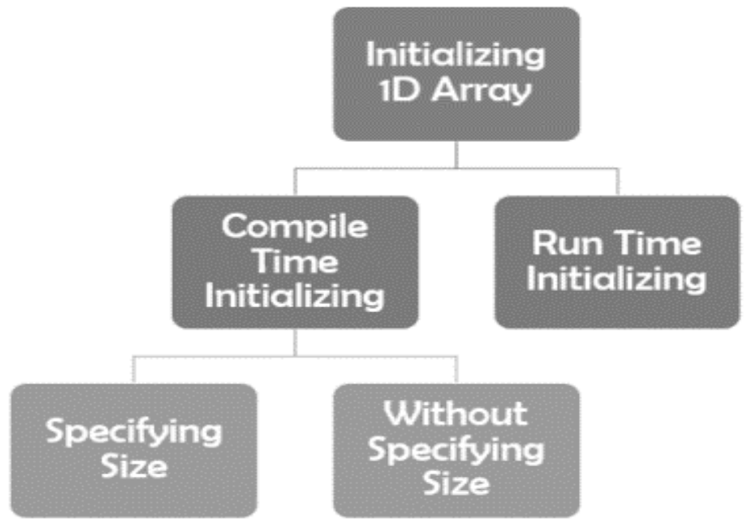
An array can be explicitly initialized at run time. This approach is usually applied for initializing large arrays.

```
for ( i=0;  i< 100;   i++)
{
  if ( i<50)
     sum[i]=0.0;
  else
     sum [i] = 1.0;
}
```

The first 50 elements of the array sum are initialized to 0 wile the remaining 50 elements are initialized to 1.0 at run time.

Ways of Array Initializing 1-D Array:

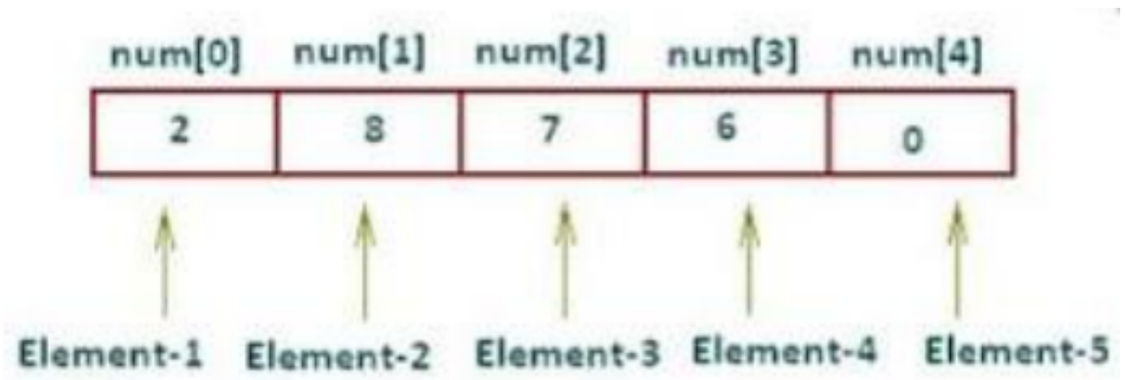
1. Size is Specified Directly
2. Size is Specified Indirectly



Method 1: Array Size Specified Directly

In this method, we try to specify the Array Size directly.

int num [5] = {2,8,7,6,0};



a) int num [3] = { 0,0,0 } ; /* declares the variable ‘num’ as an array of size 3 and

will assign zero to each element. */

b) float total[5]= { 0.0, 15.75, -10 }; /* initializes first 3 elements and the remaining 2

elements to zero. */

Method 2: Size Specified Indirectly

In this scheme of compile time Initialization, We do not provide size to an array but instead we provide set of values to the array.

int num[ ] = {2,8,7,6,0};

c) int x[ ] = { 25, 40, 50 }; /* the size may be omitted. In such cases, the compiler allocates enough space for all initialized

elements. */

c) char name [ ] = { ‘J’, ‘o’, ‘h’, ‘n’, ‘\0’ }; /* declares the name to be an array of 5 haracters

with null character */

d) char name[ ] = “John”; /* can assign the string literal directly */

1. Compiler Counts the Number of Elements written inside pair of braces and determines
the Size of An Array.
2. After counting the number of elements inside the braces, The size of array is considered
as 5 during complete execution.
3. This type of Initialization Scheme is also Called as “Compile Time Initialization.

ACCESSING OF ARRAY ELEMENT:

/*Write a program to input values into an array and display

```
#include<stdio.h>
int main()
{
int arr[5],i;
for(i=0;i<5;i++)
{
printf( “enter a value for arr[%d] \n” ,i);
scanf( “%d” ,&arr[i]);
}
printf( “the array elements are: \n” );
for (i=0;i<5;i++)
{
printf( “%d\t” ,arr[i]);
```

OUTPUT:

```
Enter a value for arr[0] = 12
Enter a value for arr[1] =45
Enter a value for arr[2] =59
Enter a value for arr[3] =98
Enter a value for arr[4] =21

The array elements are 12 45 59 98
21
```

```
}
return 0;
}
```

Example

```
#include <stdio.h>
int main( )
{
 int avg, sum = 0 ;
 int i ,  marks[30] ;                    /* array declaration */
for ( i = 0 ; i <= 29 ; i++ )
{
 printf ( "\nEnter marks " ) ;
scanf ( "%d", &marks[i] ) ;          /* store data in array */
}
for ( i = 0 ; i <= 29 ; i++ )
 sum = sum + marks[i] ;              /* read data from an array*/
 avg = sum / 30 ;
printf ( "\nAverage marks = %d", avg ) ;
}
```

How a[i] Works?

We have following array which is declared like int arr[] = { 51,32,43,24,5,26};
As we have elements in an array, so we have track of base address of an array.
Below things
are important to access an array.

Expression	Description	Example
arr	It returns the base address of an array	Consider 2000
*arr	It gives zeroth element of an array	51

Expression	Description	Example
*(arr+0)	It also gives zeroth element of an array	51
*(arr+1)	It gives first element of an array	32

1. C Program to display array elements with addresses

```
#include<stdio.h>
#include<stdlib.h>
#define size 10
int main() {
int a[3] = { 11, 22, 33 };
printf("\n a[0],value=%d : address=%u", a[0], &a[0]);
printf("\n a[1],value=%d : address=%u", a[1], &a[1]);
printf("\n a[2],value=%d : address=%u", a[2], &a[2]);
return (0);
}
```

Output:
a[0],value=11 : address=2358832
a[1],value=22 : address=2358836
a[2],value=33 : address=2358840

---

2. C Program for Reading and printing Array Elements

```
#include<stdio.h>
int main()
{
int i, arr[50], num;
printf("\nEnter no of elements :");
scanf("%d", &num);
//Reading values into Array
printf("\nEnter the values :");
for (i = 0; i < num; i++) {
scanf("%d", &arr[i]); }
//Printing of all elements of array
for (i = 0; i < num; i++) {
printf("\narr[%d] = %d", i, arr[i]); }
return (0);
}
```

Output:

```
Enter no of elements : 5
Enter the values : 10 20 30 40 50
arr[0] = 10
arr[1] = 20
arr[2] = 30
arr[3] = 40
arr[4] = 50
```

Multidimensional array

Array having more than one subscript variable is called Multi-Dimensional array.
Multi Dimensional Array is also called as Matrix.

Syntax: <data type> <array name> [row subscript][column subscript];

```
type arrayName [ x ][ y ];
```

Declaration: Char name[50][20];

Initialization:

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

Two Dimensional Arrays:

1. Two Dimensional Array requires Two Subscript Variables
2. Two Dimensional Array stores the values in the form of matrix.
3. One Subscript Variable denotes the “Row” of a matrix.
4. Another Subscript Variable denotes the “Column” of a matrix.

Accessing 2D Array Elements:

To access every 2D array we requires 2 Subscript variables.

i – Refers the Row number

j – Refers Column Number

	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]

Declaration and use of 2D Arrays:

```
int a[3][3];

for(i=0;i<row,i++)
    for(j=0;j<col,j++) {
       printf("%d",a[i][j]); }
```

```
#include <stdio.h>
int main ()
{
 /* an array with 5 rows and 2 columns*/
int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};
 int i, 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] );
 }
 }
 return 0;
}
```

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

Method 1: Initializing all Elements row wise

For initializing 2D Array we need to assign values to each element of an array using the below

syntax.

```
int a[3][2] = { {1, 4}, {5, 2}, {6, 5} };
 #include<stdio.h>
int main()
{
int i, j;
int a[3][2] = { { 1, 4 }, { 5, 2 }, { 6, 5 } };
for (i = 0; i < 3; i++) {
```

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

Method 2: Combine and Initializing 2D Array

Initialize all Array elements but initialization is much straight forward. All values are assigned sequentially and row-wise

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

Method 3: Some Elements could be initialized

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

Now we have again going with the way 1 but we are removing some of the elements from the
array. Uninitialized elements will get default 0 value.

C Program to find smallest element in an array

```
#include<stdio.h>
int main() {
int a[30], i, num, smallest;
printf("\nEnter no of elements :");
scanf("%d", &num);
//Read n elements in an array
for (i = 0; i < num; i++)
scanf("%d", &a[i]);
//Consider first element as smallest
smallest = a[0];

for (i = 0; i < num; i++) {
if (a[i] < smallest) {
smallest = a[i]; } }


// Print out the Result
printf("\nSmallest Element : %d", smallest);
return (0);
}
```

Output:

```
Enter no of elements : 5
11 44 22 55 99
Smallest Element : 11
```

---

1. C Program for addition of two matrices

```
#include<stdio.h>
int main() {
int i, j, mat1[10][10], mat2[10][10], mat3[10][10];
int row1, col1, row2, col2;
printf("\nEnter the number of Rows of Mat1 : ");
scanf("%d", &row1);
```

```
printf("\nEnter the number of Cols of Mat1 : ");
scanf("%d", &col1);
printf("\nEnter the number of Rows of Mat2 : ");
scanf("%d", &row2);
printf("\nEnter the number of Columns of Mat2 : ");
scanf("%d", &col2);
/* before accepting the Elements Check if no of rows and columns of both matrices is
equal */
if (row1 != row2 || col1 != col2) {
printf("\nOrder of two matrices is not same ");
exit(0); }
//Accept the Elements in Matrix 1
for (i = 0; i < row1; i++) {
    for (j = 0; j < col1; j++) {
      printf("Enter the Element a[%d][%d] : ", i, j);
      scanf("%d", &mat1[i][j]); } }
//Accept the Elements in Matrix 2
for (i = 0; i < row2; i++)
    for (j = 0; j < col2; j++) {
        printf("Enter the Element b[%d][%d] : ", i, j);
        scanf("%d", &mat2[i][j]); }
//Addition of two matrices
for (i = 0; i < row1; i++)
for (j = 0; j < col1; j++) {
mat3[i][j] = mat1[i][j] + mat2[i][j];}
//Print out the Resultant Matrix
printf("\nThe Addition of two Matrices is : \n");
for (i = 0; i < row1; i++) {
for (j = 0; j < col1; j++) {
printf("%d\t", mat3[i][j]); }
printf("\n"); }
return (0);
}
```

Output:

```
Enter the number of Rows of Mat1 : 3
Enter the number of Columns of Mat1 : 3
Enter the number of Rows of Mat2 : 3
Enter the number of Columns of Mat2 : 3
Enter the Element a[0][0] : 1
Enter the Element a[0][1] : 2
Enter the Element a[0][2] : 3
Enter the Element a[1][0] : 2
Enter the Element a[1][1] : 1
Enter the Element a[1][2] : 1
Enter the Element a[2][0] : 1
Enter the Element a[2][1] : 2
Enter the Element a[2][2] : 1
Enter the Element b[0][0] : 1
```

```
Enter the Element b[0][1] : 2
Enter the Element b[0][2] : 3
Enter the Element b[1][0] : 2
Enter the Element b[1][1] : 1
Enter the Element b[1][2] : 1
Enter the Element b[2][0] : 1
Enter the Element b[2][1] : 2
Enter the Element b[2][2] : 1
The Addition of two Matrices is :
2 4 6
4 2 2
2 4 2
```

STRINGS

A string is a sequence of character enclosed with in double quotes (“ ”) but ends with \0. The compiler puts \0 at the end of string to specify the end of the string.

To get a value of string variable we can use the two different types of formats.

Using scanf() function as: scanf(“%s”, string variable);
Using gets() function as : gets(string variable);

String Manipulation Functions

Function	Use
strchr()	Appends one string to another
strchr()	Finds first occurrence of a given character in a string
strcmp()	Compares two strings
strcmpi()	Compares two strings without regard to case
strcpy()	Copies one string to another
strdup()	Duplicates a string
stricmp()	Compares two strings without regard to case (identical to strcmpi)
strlen()	Finds length of a string
strlwr()	Converts a string to lowercase
strncat()	Appends a portion of one string to another
strncmp()	Compares a portion of one string with portion of another string
strncpy()	Copies a given number of characters of one string to another
strnicmp()	Compares a portion of one string with a portion of another without regard to case
strrchr()	Finds last occurrence of a given character in a string
strrev()	Reverses a string
strset()	Sets all characters in a string to a given character
strstr()	Finds first occurrence of a given string in another string
strupr()	Converts a string to uppercase

(i) strlen() function

strlen() is used to return the length of the string , that means counts the number of characters present in a string.

Syntax

integer variable = strlen (string variable);

Example

```
#include<stdio.h>
#include<conio.h>
void main()
{
char str[20];
int strlength;
clrscr();
printf( ,  Enter String:‘);
gets(str);
strlength=strlen(str);
printf( ,  Given String Length Is: %d‘, strlength);
getch();
}
```

(ii) strcat() function

The strcat() is used to concatenate two strings. The second string will be appended to the end of the first string. This process is called concatenation.

Syntax

strcat (StringVariable1, StringVariable 2);

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
char str1[20],str2[20];
clrscr();
printf( ,  Enter First String:‘);
scanf( ,  %s‘,str1);
printf( ,  Enter Second String:‘);
scanf( ,  %s‘,str2);
printf( ,   Concatenation String is:%s‘, strcat(str1,str2));
getch();
}
```

Output:

```
Enter First String
Good
Enter Second String
Morning
Concatenation String is: GoodMorning
```

(iii) strcmp() function

strcmp() function is used to compare two strings. strcmp() function does a case sensitive comparison between two strings. The two strings are compared character by character until there is a mismatch or end of one of the strings is reached (whichever occurs first). If the two strings are identical, strcmp( ) returns a value zero. If they‟re not, it returns the numeric difference between the ASCII values of the first non-matching pairs of characters.

Syntax
strcmp(StringVariable1, StringVariable2);
Example:
```
#include<stdio.h>
#include<conio.h>
void main()
{
char str1[20], str2[20];
int res;
clrscr();
printf( , Enter First String:‘);
scanf( , %s‘,str1);
printf( , Enter Second String:‘);
scanf( , %s‘,str2);
res = strcmp(str1,str2);
printf( , Compare String Result is:%d‘,res);
getch();
}
```
Output:
```
Enter First String
Good
Enter Second String
Good
Compare String Result is: 0
```

(iv) strcmpi() function
strcmpi() function is used to compare two strings. strcmpi() function is not case sensitive.
Syntax
strcmpi(StringVariable1, StringVariable2);

(v) strcpy() function:
strcpy() function is used to copy one string to another. strcpy() function copy the contents of
second string to first string.
Syntax
strcpy(StringVariable1, StringVariable2);
Example:
```
#include<stdio.h>
#include<conio.h>
void main()
{
char str1[20], str2[20];
int res;
clrscr();
printf( , Enter First String:‘);
scanf( , %s‘,str1);
printf( , Enter Second String:‘);
scanf( , %s‘,str2);
strcpy(str1,str2)
```

```
printf( ,   First String is:%s',str1);
printf( ,   Second String is:%s',str2);
getch();
}
```

Output:

```
Enter First String
Hello
Enter Second String
welcome
First String is: welcome
Second String is: welcome
```

(vi) strlwr () function:

This function converts all characters in a given string from uppercase to lowercase letter.

Syntax

```
strlwr(StringVariable);
```

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
char str[20];
clrscr();
printf( ,  Enter String:');
gets(str);
printf( "Lowercase String : %s', strlwr(str));
getch();
}
```

Output:

```
Enter String
WELCOME
Lowercase String : welcome
```

(vii) strrev() function:

strrev() function is used to reverse characters in a given string.

Syntax

```
strrev(StringVariable);
```

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
char str[20];
clrscr();
printf( "Enter String:');
gets(str);
```

printf(“Reverse String : %s‘, strrev(str));
getch();
}
Output:
Enter String
WELCOME
Reverse String : emoclew

viii) strupr() function:
strupr() function is used to convert all characters in a given string from lower case to
uppercase letter.

Syntax
strupr(Stringvariable);
Example:
#include<stdio.h>
#include<conio.h>
void main()
{
char str[20];
clrscr();
printf(, Enter String:‘);
gets(str);
printf(, Uppercase String : %s‘, strupr(str));
getch();
} Output:
Enter String
welcome
Uppercase String : WELCOME

Selection Sort

Selection sort is a simple and efficient sorting algorithm that works by repeatedly selecting the smallest (or largest) element from the unsorted portion of the list and moving it to the sorted portion of the list.

```
#include <stdio.h>
void main()
{
 int n,a[10],out,in,tmp;
 printf("Enter number of vaues :");
 scanf("%d",&n);
 for(out=0; out<n; ++out)
   scanf("%d", &a[out]);
```

```
for (out=0; out<n; ++out)
  {
  for(in=out+1; in<n; ++in)
  {
   if(a[out]>a[in])
     {
      tmp=(a[out]);
      a[out]=a[in];
      a[in]=tmp;
     }
    }
 printf("The sorted elements are \n: ");
 for (out=0;out<n;++out)
   printf("%d\n", a[out]);
}
```

OUTPUT

```
Enter number of vaues :7
23
5
65
14
27
16
31
The sorted elements are :
5
14
16
23
27
31
65
```

Linear Search

```
#include <stdio.h>

int linearSearch(int arr[], int n, int key) {
    int i;
    for (i = 0; i < n; i++) {
        if (arr[i] == key) {
            return i; // Return the index where key is found
        }
    }
    return -1; // Return -1 if key is not found
}
```

```
int main() {
    int arr[] = {2, 5, 7, 12, 15, 18};
    int n = sizeof(arr) / sizeof(arr[0]);
    int key = 15;

    int result = linearSearch(arr, n, key);
        if (result == -1) {
        printf("Element not found\n");
    } else {
        printf("Element found at index %d\n", result);
    }

    return 0;
}
```

Another method for getting input values and then find an element using Linear search.

```
#include <stdio.h>
int main()
{
  int array[100], search, c, n;

  printf("Enter number of elements in array\n");
  scanf("%d", &n);
  printf("Enter %d integer(s)\n", n);

  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);
  printf("Enter a number to search\n");
  scanf("%d", &search);

 for (c = 0; c < n; c++)
  {
    if (array[c] == search)    /* If required element is found */
    {
      printf("%d is present at location %d.\n", search, c+1);
      break;
    }
  }
  if (c == n)
    printf("%d isn't present in the array.\n", search);

  return 0;
}

Enter number of elements in array
7
Enter 7 integer(s)
25
```

```
27
14
21
22
23
17

Enter a number to search
22
22 is present at location 5.
```

C Header files

Header file	Description
stdio.h	Input and Output functions
conio.h	Console Input and Output functions
stdlib.h	Some standard library functions
math.h	Mathematical functions
string.h	String manipulation functions
ctype.h	Character handling functions
time.h	Time computing functions
malloc.h	Memory allocation and deallocation functions
graphics.h	Graphical functions
dos.h	Function linking DOS routines
wctype.h	Functions to classify and transform individual wide characters
limits.h	Functions define various symbolic names
float.h	Functions define set of various platform-dependent constants related to floating point values