Module-II

BCSE102L_STRUCTURED-AND-OBJECT-ORIENTED-PROGRAMMING

Prepared by

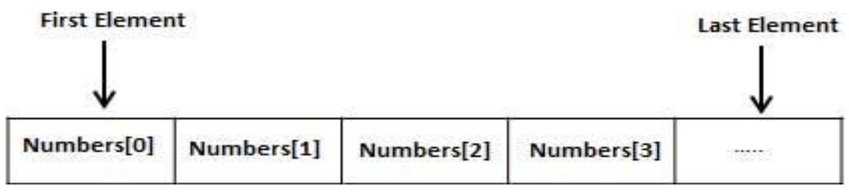
Dr.M.Suguna

D1 Slot and C1-Slot

Arrays & Strings

Introduction:

- An array is a collection of data items, all of the same type, accessed using a common name.
- A one-dimensional array is like a list; A two dimensional array is like a table; The C language places no limits on the number of dimensions in an array, though specific implementations may.
- Some texts refer to one-dimensional arrays as **vectors**, two-dimensional arrays as **matrices**, and use the general term **arrays** when the number of dimensions is unspecified or unimportant.
- All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



Declaring Arrays:

- 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:

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

- Array variables are declared identically to variables of their data type, except that the variable name is followed by one pair of square [ ] brackets for each dimension of the array.
- Uninitialized arrays must have the dimensions of their rows, columns, etc. listed within the square brackets.

- This is called a single-dimensional array. The **arraySize** must be an integer constant greater than zero and **type** can be any valid C data type. For example, to declare a 10-element array called **balance** of type double, use this statement:

```
double balance[10];
```

- Now balance is avariable array which is sufficient to hold upto 10 double numbers.

Initializing Arrays:

- Arrays may be initialized when they are declared, just as any other variables.

- Place the initialization data in curly {} braces following the equals sign. Note the use of commas in the examples below.
- An array may be partially initialized, by providing fewer data items than the size of the array. The remaining array elements will be automatically initialized to zero.
- If an array is to be completely initialized, the dimension of the array is not required. The compiler will automatically size the array to fit the initialized data.

  Examples:

  ```
  int i =  5, intArray[ 6 ] = { 1, 2, 3, 4, 5, 6 }, k;
  float sum  = 0.0f, floatArray[ 100 ] = { 1.0f, 5.0f, 20.0f };
  double  piFractions[ ] = { 3.141592654, 1.570796327, 0.785398163 };
  ```

The following program initializes an integer array with five values and prints the array.

```
#include <stdio.h>
#include <conio.h>
int main()
    {
    int numbers[]={1,2,3,4,5};
    int i;
    clrscr();
    printf("Array elements are\n");
    for(i=0;i<=4;i++)
           printf("%d\n",numbers[i]);
    getch();
    return 0;
   }
```

Using Arrays

- Elements of an array are accessed by specifying the index *(offset)* of the desired element within square [ ] brackets after the array name.
- Array subscripts must be of integer type. (int, long int, char, etc.)
- Array indices start at zero in C, and go to one less than the size of the array. For example, a five element array will have indices zero through four. This is because the index in C is actually an offset from the beginning of the array. *(The* first element is at the beginning of the array, and hence has zero offset.)
- The most common mistake when working with arrays in C is forgetting that indices start at zero and stop one less than the array size.
- Arrays are commonly used in conjunction with loops, in order to perform the same calculations on all *(or* some *part)* of *the* data items in the array.

There are 2 types of C arrays. They are,

1. One dimensional array
2. Multi dimensional array

    - Two dimensional array
        - Three dimensional array

Declaration of one-dimensional array

- Syntax : data-type arr_name[array_size];

Array declaration	Array initialization	Accessing array
Syntax: data_type arr_name [arr_size];	data_type arr_name [arr_size]= (value1, value2, value3,….);	arr_name[index];
int age [5];	int age[5]={0, 1, 2, 3, 4, 5};	age[0]; /*0 is accessed*/ age[1]; /*1 is accessed*/ age[2]; /*2 is accessed*/

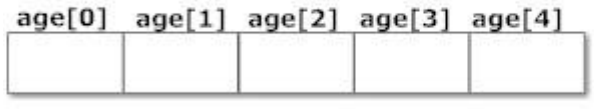
char str[10];	char str[10]={'H','a','i'}; (or)
	char str[0] = 'H';
	char str[1] = 'a';
	char str[2] = 'i;
		str[0]; /*H is accessed*/
		str[1]; /*a is accessed*/
		str[2]; /* i is accessed*/

For example:
int age[5];

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

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

int age[5];



Array elements

Initialization of one-dimensional array:

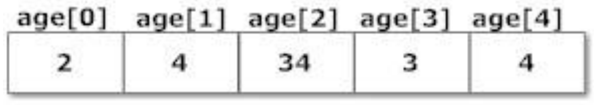
Arrays can be initialized at declaration time in this source code as:

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

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

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

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



Initialization of one-dimensional array

Accessing array elements

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

For example:

```
scanf("%d",&age[2]);
/* statement to insert value in the third element of array age[]. */

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

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

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

Example program for one dimensional array in C:

```
#include<stdio.h>
int main()
{
   int i;
   int arr[5] = {10,20,30,40,50};
   // declaring and Initializing array in C
   //To initialize all array elements to 0, use int arr[5]={0};
   /* Above array can be initialized as below also
      arr[0] = 10;
      arr[1] = 20;
      arr[2] = 30;
```

```
        arr[3] = 40;
        arr[4] = 50;
    */
    for (i=0;i<5;i++)
    {
        // Accessing each variable
        printf("value of arr[%d] is %d \n", i, arr[i]);
    }
}
```

Output:

```
value of arr[0] is 10
value of arr[1] is 20
value of arr[2] is 30
value of arr[3] is 40
value of arr[4] is 50
```

/* C program to find the sum marks of n students using arrays */

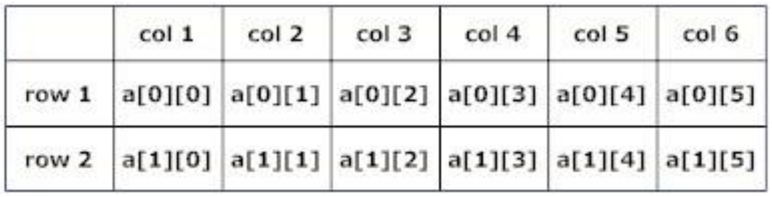
```
#include <stdio.h>
int main(){
     int marks[10],i,n,sum=0;
     printf("Enter number of students: ");
     scanf("%d",&n);
     for(i=0;i<n;++i){
          printf("Enter marks of student%d: ",i+1);
          scanf("%d",&marks[i]);
          sum+=marks[i];
     }
     printf("Sum= %d",sum);
return 0;
}
```

Output

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

Two dimensional array:

An array of two dimensions, which is an example of multidimensional array. This array has 2 rows and 6 columns



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

Figure: Multidimensional Arrays

- Two dimensional array is nothing but array of array.
- syntax : data_type array_name[num_of_rows][num_of_column]

S.no	Array declaration	Array initialization	Accessing array
1	Syntax: data_type arr_name [num_of_rows][num_of_column];	data_type arr_name[2][2] = {{0,0},{0,1},{1,0},{1,1}};	arr_name[index];

2	Example:
	int arr[2][2];
		int arr[2][2] = {1,2, 3, 4};	arr [0] [0] = 1;
			arr [0]]1] = 2;
			arr [1][0] = 3;
			arr [1] [1] = 4;

Initialization of 2D Array

There are many ways to initialize two Dimensional arrays –

```
int disp[2][4] = {
    {10, 11, 12, 13},
    {14, 15, 16, 17}
};
```

OR

```
int disp[2][4] = { 10, 11, 12, 13, 14, 15, 16, 17};
```

Example

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

   /* 2D array declaration*/
   int disp[3][5];
   /*Counter variables for the loop*/
   int i, j;

   for(i=0; i<=2; i++)
   {
      for(j=0;j<=4;j++)
      {
         printf("Enter value for disp[%d][%d]:", i, j);
         scanf("%d", &disp[i][j]);
      }
    }
    return 0;
}
```

Initialization of Multidimensional Arrays

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

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

Example program for two dimensional array in C:

```
#include<stdio.h>
int main()
{
    int i,j;
    // declaring and Initializing array
    int arr[2][2] = {10,20,30,40};
    /* Above array can be initialized as below also
       arr[0][0] = 10;   // Initializing array
       arr[0][1] = 20;
       arr[1][0] = 30;
       arr[1][1] = 40;
    */
    for (i=0;i<2;i++)
    {
       for (j=0;j<2;j++)
       {         // Accessing variables
          printf("value of arr[%d] [%d] : %d\n",i,j,arr[i][j]);
       }
    }}
```

Output:

```
value of arr[0] [0] is 10
value of arr[0] [1] is 20
value of arr[1] [0] is 30
value of arr[1] [1] is 40
```

- For two dimensional arrays, the first dimension is commonly considered to be the number of rows, and the second dimension the number of columns

consider while initializing 2D array –

```
/* Valid declaration*/
int abc[2][2] = {1, 2, 3 ,4 }
/* Valid declaration*/
int abc[][2] = {1, 2, 3 ,4 }
/* Invalid declaration – you must specify second dimension*/
int abc[][] = {1, 2, 3 ,4 }
/* Invalid because of the same reason  mentioned above*/
int abc[2][] = {1, 2, 3 ,4 }
```

Initialization of three-dimensional Array

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

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

String

- C Strings are nothing but array of characters ended with null character (‘\0’).
- This null character indicates the end of the string.
- Strings are always enclosed by double quotes. Whereas, character is enclosed by single quotes in C.

Example for C string:

- char string[20] = { ‘A’ , ’E’ , ‘R’ , ‘O’ , ‘N’ , ‘\0’}; (or)
- char string[20] = “Welcome to kct”; (or)
- char string []   = “good morning”;

- Difference between above declarations are, when we declare char as “string[20]“, 20 bytes of memory space is allocated for holding the string value.

- When we declare char as “string[]“, memory space will be allocated as per the requirement during execution of the program.

String declaration and initialization

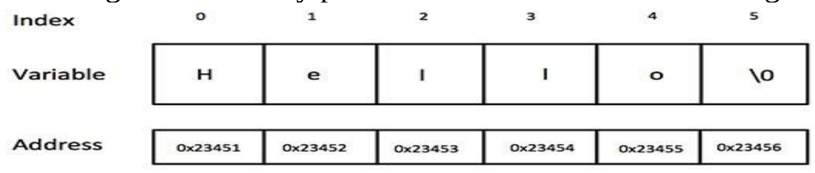
The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

If you follow the rule of array initialization then you can write the above statement as follows:

char greeting[] = "Hello";

Following is the memory presentation of above defined string in C/C++:



Actually, do not place the *null* character at the end of a string constant. The C compiler automatically places the '\0' at the end of the string when it initializes the array. Let us try to print above mentioned string:

```
#include <stdio.h>
int main ()
{
   char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
   printf("Greeting message: %s\n", greeting );
   return 0;
}
```

When the above code is compiled and executed, it produces result something as follows:

Output: Greeting message: Hello

C supports a wide range of functions that manipulate null-terminated strings:

As it can see the character array is declared in the same way as a normal array. This array can hold only 19 characters,

```
#include<stdio.h>
int main()
{           char mystring[20];
            mystring[0] = 'H';
            mystring[1] = 'E';
            mystring[2] = 'L';
            mystring[3] = 'L';
            mystring[4] = 'O';
            mystring[5] = '\n';
            mystring[6] = '\0';
            printf("%s", mystring);
            return 0;
}
```

Note: %s is used to print a string. (The 0 without the ” will in most cases also work).
String pointers are declared as a pointer to a char. When there is a value assigned to the **string pointer** the NULL is put at the end automatically. Take a look at this example:

```
#include<stdio.h>
int main()
{           char *ptr_mystring;
            ptr_mystring = "HELLO";
            printf("%s\n", ptr_mystring);
            return 0;
}
```

Example program for C string:

```
#include <stdio.h>
int main ()
{
  char string[20] = "welcome to kct";
  printf("The string is : %s \n", string );
  return 0;
}
```

Output:

```
The string is : welcome to kct
```

string.h or strings.h

The C language provides no explicit support for strings in the language itself. The string-handling functions are implemented in libraries. String I/O operations are implemented in

<stdio.h> (puts , gets, etc). A set of simple string manipulation functions are implemented in <string.h>, or on some systems in <strings.h>.

The string library (string.h or strings.h) has some useful functions for working with strings, like strcpy, strcat, strcmp, strlen, strcoll, etc. We will take a look at some of these string operations.

C String functions:

- String.h header file supports all the string functions in C language. All the string functions are given below.

S.no	String functions	Description
1	**strcat ( )**	Concatenates str2 at the end of str1.
2	**strncat ( )**	appends a portion of string to another
3	**strcpy ( )**	Copies str2 into str1
4	**strncpy ( )**	copies given number of characters of one string to another
5	**strlen ( )**	gives the length of str1.
6	**strcmp ( )**	Returns 0 if str1 is same as str2. Returns <0 if strl < str2. Returns >0 if str1 > str2.
7	**strcmpi ( )**	Same as strcmp() function. But, this function negotiates case. "A" and "a" are treated as same.
8	**strchr ( )**	Returns pointer to first occurrence of char in str1.
9	**strrchr ( )**	last occurrence of given character in a string is found
10	**strstr ( )**	Returns pointer to first occurrence of str2 in str1.
11	**strrstr ( )**	Returns pointer to last occurrence of str2 in str1.
12	**strdup ( )**	duplicates the string
13	**strlwr ( )**	converts string to lowercase
14	**strupr ( )**	converts string to uppercase
15	**strrev ( )**	reverses the given string
16	**strset ( )**	sets all character in a string to given character
17	**strnset ( )**	It sets the portion of characters in a string to given character
18	**strtok ( )**	tokenizing given string using delimiter

1. 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.

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 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.

Example program for strcat( ) function in C:

  - In this program, two strings "swetha" and "sri" are concatenated using strcat( ) function and result is displayed as "swetha sri".

```
#include <stdio.h>
#include <string.h>
int main( )
{
  char source[ ] = " swetha " ;
  char target[ ]= " sri " ;
  printf ( "\nSource string = %s", source ) ;
  printf ( "\nTarget string = %s", target ) ;
  strcat ( target, source ) ;
  printf ( "\nTarget string after strcat( ) = %s", target ) ;
}
```

Output:

```
Source string                =swetha
Target string                 = sri
Target string after strcat( ) = swetha sri
```

2.strncat( ) function in C language concatenates (appends) portion of one string at the end of another string. Syntax for strncat( ) function is given below.

char * strncat (char * destination, const char * source, size_t num);

- **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.

- 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.

Example program for strncat( ) function in C:

  - In this program, first 5 characters of the string "fresh2refresh" is concatenated at the end of the string "C tutorial" using strncat( ) function and result is displayed as "C tutorial fres".

```
#include <stdio.h>
#include <string.h>
int main( )
{
  char source[ ] = " first year " ;
  char target[ ]= " welcome " ;
  printf ( "\nSource string = %s", source ) ;
  printf ( "\nTarget string = %s", target ) ;
  strncat ( target, source, 5 ) ;
  printf ( "\nTarget string after strncat( ) = %s", target ) ;
}
```

Output:

```
Source string                    = first year
Target string                    = welcome
Target string after strcat( ) = welcome first
```

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

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 program for strcpy( ) function in C:

  - In this program, source string "Coimbatore" is copied into target string using strcpy( ) function.

```
#include <stdio.h>
#include <string.h>
int main( )
{
  char source[ ] = "Coimbatore" ;
  char target[20]= "" ;
  printf ( "\nsource string = %s", source ) ;
  printf ( "\ntarget string = %s", target ) ;
  strcpy ( target, source ) ;
  printf ( "\ntarget string after strcpy( ) = %s", target ) ;
  return 0;
}
```

Output:

```
source string = Coimbatore
target string =
target string after strcpy( ) = Coimbatore
```

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

char * strncpy (char * destination, const char * source, size_t num);

- **Example:**

strncpy (str1, str2, 4) – It copies first 4 characters of str2 into str1.
strncpy (str2, str1, 4) – It copies first 4 characters 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.
- If you want to copy 25 characters from source string using strncpy( ) function, only 20 characters from source string will be copied into destination string and remaining 5 characters won't be copied and will be truncated.

Example program for strncpy( ) function in C:

  - In this program, only 5 characters from source string "Coimbatore" is copied into target string using strncpy( ) function.

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

int main( )
{
  char source[ ] = " Coimbatore " ;
  char target[20]= "" ;
  printf ( "\nsource string = %s", source ) ;
  printf ( "\ntarget string = %s", target ) ;
  strncpy ( target, source, 5 ) ;
  printf ( "\ntarget string after strcpy( ) = %s", target ) ;
  return 0;
}
```

Output:

```
source string = Coimbatore
target string =
target string after strncpy( ) = coimb
```

<u>5.strlen( ) function</u> in C gives the length of the given string. Syntax for strlen( ) function is given below.

size_t strlen (const char * str);

- strlen( ) function counts the number of characters in a given string and returns the integer value.
- It stops counting the character when null character is found. Because, null character indicates the end of the string in C.

Example program for strlen() function in C:

- In below example program, length of the string "www.google.com" is determined by strlen( ) function as below. Length of this string 17 is displayed as output.

```
#include <stdio.h>
#include <string.h>
int main( )
{
   int len;
   char array[20]="www.google.com " ;
   len = strlen(array) ;
   printf ( "\string length  = %d \n" , len ) ;
   return 0;
}
```

Output:

```
string length = 14
```

6. 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 for strcmp( ) function is given below.

```
int strcmp ( const char * str1, const char * str2 );
```

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

Example program for strcmp( ) function in C:

- In this program, strings “fresh” and “refresh” are compared. 0 is returned when strings are equal. Negative value is returned when str1 < str2 and positive value is returned when str1 > str2.

```
#include <stdio.h>
#include <string.h>
int main( )
{
  char str1[ ] = "fresh" ;
  char str2[ ] = "refresh" ;
  int i, j, k ;
  i = strcmp ( str1, "fresh" ) ;
  j = strcmp ( str1, str2 ) ;
  k = strcmp ( str1, "f" ) ;
  printf ( "\n%d %d %d", i, j, k ) ;
  return 0;
}
```

Output:

```
0 -1 1
```

7.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. Where as, strcmp() function treats “A” and “a” as different characters.

- strcmpi() function is non standard function which may not available in standard library in C.
- 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. Syntax for strcmp( ) function is given below.

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

Example program for strcmpi( ) function in C:

- In this program, strings "fresh" and "refresh" are compared. 0 is returned when strings are equal. Negative value is returned when str1 < str2 and positive value is returned when str1 > str2.

```
#include <stdio.h>
#include <string.h>
int main( )
{
   char str1[ ] = "fresh" ;
   char str2[ ] = "refresh" ;
   int i, j, k ;
   i = strcmpi ( str1, "FRESH" ) ;
   j = strcmpi ( str1, str2 ) ;
   k = strcmpi ( str1, "f" ) ;
   printf ( "\n%d %d %d", i, j, k ) ;
   return 0;
}
```

Output:

```
0 -1 1
```

8.strchr( ) function returns pointer to the first occurrence of the character in a given string. Syntax for strchr( ) function is given below.

char *strchr(const char *str, int character);

Example program for strchr() function in C:

- In this program, strchr( ) function is used to locate first occurrence of the character 'i' in the string "This is a string for testing". Character 'i' is located at position 3 and pointer is returned at first occurrence of the character 'i'.

```
#include <stdio.h>
#include <string.h>
int main ()
{
  char string[55] ="This is a string for testing";
  char *p;
  p = strchr (string,'i');

  printf ("Character i is found at position %d\n",p-string+1);
  printf ("First occurrence of character \"i\" in \"%s\" is" \
         " \"%s\"",string, p);

   return 0;
}
```

Output:

```
Character i is found at position 3
First occurrence of character “i” in “This is a string for testing” is “is is a string for testing”
```

9.strrchr( ) function in C returns pointer to the last occurrence of the character in a given string. Syntax for strrchr( ) function is given below.

char *strrchr(const char *str, int character);

Example program for strrchr() function in C:

- In this program, strrchr( ) function is used to locate last occurrence of the character ‘i’ in the string ”This is a string for testing”. Last occurrence of character ‘i’ is located at position 26 and pointer is returned at last occurrence of the character ‘i’.

```
#include <stdio.h>
#include <string.h>
int main ()
{
  char string[55] ="This is a string for testing";
  char *p;
  p = strrchr (string,'i');
  printf ("Character i is found at position %d\n",p-string+1);
  printf ("Last occurrence of character \"i\" in \"%s\" is" \" \"%s\"",string, p);
   return 0;
}
```

Output:

```
Character i is found at position 26
Last occurrence of character “i” in “This is a string for testing” is “ing”
```

10.strstr( ) function returns pointer to the first occurrence of the string in a given string. Syntax for strstr( ) function is given below.

char *strstr(const char *str1, const char *str2);

Example program for strstr() function in C:

- In this program, strstr( ) function is used to locate first occurrence of the string “test” in the string ”This is a test string for testing”. Pointer is returned at first occurrence of the string “test”.

```
#include <stdio.h>
#include <string.h>
int main ()
{  char string[55] ="This is a test string for testing";
  char *p;
  p = strstr (string,"test");
  if(p)
  {    printf("string found\n" );
    printf ("First occurrence of string \"test\" in \"%s\" is"\
          " \"%s\"",string, p);  }
  else printf("string not found\n" );
   return 0;}
```

Output:

```
string found
First occurrence of string “test” in “This is a test string for testing” is “test string for testing”
```

11.strrstr( ) function returns pointer to the last occurrence of the string in a given string. Syntax for strrstr( ) function is given below.

char *strrstr(const char *str1, const char *str2);

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

Example program for strrstr() function in C:

- In this program, strrstr( ) function is used to locate last occurrence of the string “test” in the string ”This is a test string for testing”. Pointer is returned at last occurrence of the string “test”.

```
#include <stdio.h>
#include <string.h>
int main ()
{  char string[55] ="This is a test string for testing";
  char *p;
  p = strrstr (string,"test");
  if(p)
  {    printf("string found\n" );
    printf ("Last occurrence of string \"test\" in \"%s\" is"\" \"%s\"",string, p);
  }
  else printf("string not found\n" );
   return 0;}
```

Output:

```
string found
Last occurrence of string “test” in “This is a test string for testing” is “testing”
```

12.strdup( ) function in C duplicates the given string. Syntax for strdup( ) function is given below.

char *strdup(const char *string);

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

Example program for strdup() function in C:

- In this program, string “Hari” is duplicated using strdup( ) function and duplicated string is displayed as output.

```
#include <stdio.h>
#include <string.h>
int main()
{    char *p1 = "Hari";
    char *p2;
    p2 = strdup(p1);
    printf("Duplicated string is : %s", p2);
    return 0;}
```

Output:

```
Duplicated string is : Hari
```

13.strlwr( ) function converts a given string into lowercase. Syntax for strlwr( ) function is given below.

char *strlwr(char *string);

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

Example program for strlwr() function in C:

- In this program, string "MODIFY This String To LOwer" is converted into lower case using strlwr( ) function and result is displayed as "modify this string to lower".

```
#include<stdio.h>
#include<string.h>
int main()
{
   char str[ ] = "MODIFY This String To LOwer";
   printf("%s\n",strlwr (str));
   return  0;
}
```

Output:

```
modify this string to lower
```

14.strtok( ) function in C tokenizes/parses the given string using delimiter. Syntax for strtok( ) function is given below.

char * strtok (char * str, const char * delimiters);

Example program for strtok() function in C:

- In this program, input string "Test,string1,Test,string2:Test:string3″ is parsed using strtok() function. Delimiter comma (,) is used to separate each sub strings from input string.

```
#include <stdio.h>
#include <string.h>
int main ()
{  char string[50] ="Test,string1,Test,string2:Test:string3";
  char *p;
  printf ("String  \"%s\" is split into tokens:\n",string);
  p = strtok (string,",:");
  while (p!= NULL)
  {    printf ("%s\n",p);
    p = strtok (NULL, ",:");  }
  return 0;}
```

Output:

```
String "Test,string1,Test,string2:Test:string3″ is split into tokens:
Test
string1
Test
string2
Test
string3
```

15. strnset( ) function sets portion of characters in a string to given character. Syntax for strnset( ) function is given below.

char *strnset(char *string, int c);

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

Example program for strnset() function in C:

- In this program, first 4 characters of the string “Test String” is set to “#” using strnset( ) function and output is displayed as “#### String”.

```
#include<stdio.h>
#include<string.h>
int main()
{
   char str[20] = "Test String";
   printf("Original string is : %s", str);
   printf("Test string after string n set" \
          " : %s", strnset(str,'#',4));
   printf("After string n set : %s", str);
   return 0;
}
```

Output:

```
Original string is              : Test String
Test string after string set : #### String
```

<u>16.strset( ) function</u> sets all the characters in a string to given character. Syntax for strset( ) function is given below.

char *strset(char *string, int c);

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

Example program for strset() function in C:

- In this program, all characters of the string “Test String” is set to “#” using strset( ) function and output is displayed as “###########”.

```
#include<stdio.h>
#include<string.h>
int main()
{
  char str[20] = "Test String";
  printf("Original string is : %s", str);
  printf("Test string after strset() : %s",strset(str,'#'));
  printf("After string set: %s",str);
  return 0;
}
```

Output:

```
Original string is            : Test String
Test string after strset() : ###########
```

<u>17.strrev( ) function</u> reverses a given string in C language. Syntax for strrev( ) function is given below.

char *strrev(char *string);

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

Example program for strrev() function in C:

- In below program, string “Hello” is reversed using strrev( ) function and output is displayed as “olleH”.

```
#include<stdio.h>
#include<string.h>
int main()
{
  char name[30] = "Hello";
  printf("String before strrev( ) : %s\n",name);
  printf("String after strrev( )  : %s",strrev(name));
  return 0;
}
```

Output:

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

Problem solving using C

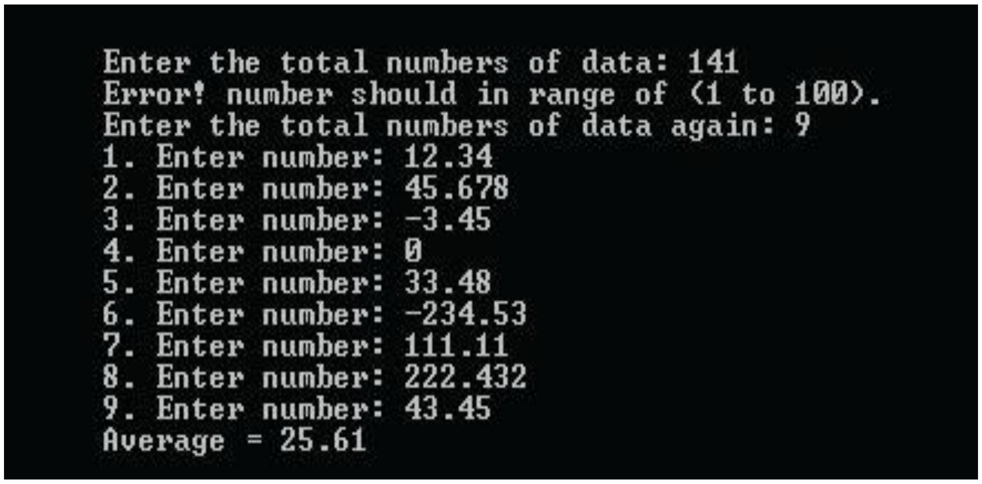
UNIT-II

Example program using Arrays and Strings

<u>1.Calculate Average Using Arrays</u>

```
#include <stdio.h>
int main()
{
   int n, i;
   float num[100], sum=0.0, average;
   printf("Enter the numbers of data: ");
   scanf("%d",&n);
  for(i=0; i<n; i++)
  {
     printf("%d. Enter number: ",i+1);
     scanf("%f",&num[i]);
     sum=sum+num[i];
  }
  average=sum/n;
  printf("Average = %.f",average);
  return 0;
}
```

Output

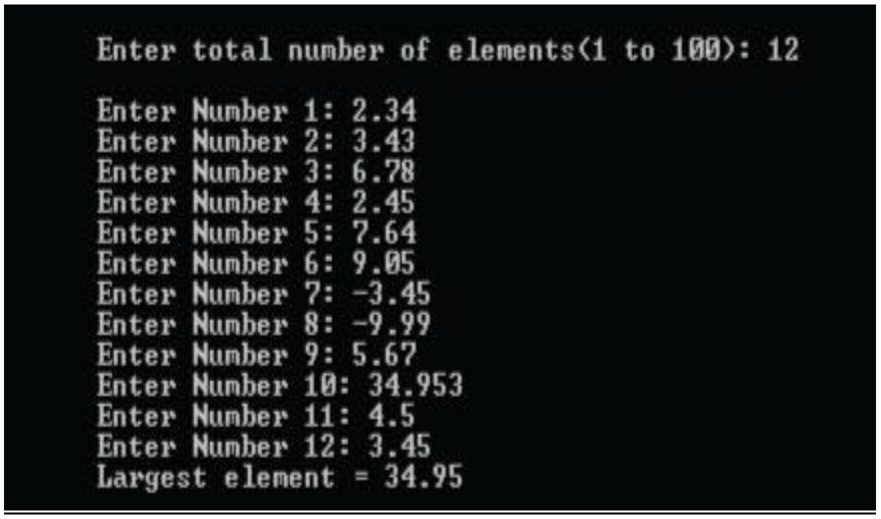


```
Enter the total numbers of data: 141
Error! number should in range of (1 to 100).
Enter the total numbers of data again: 9
1. Enter number: 12.34
2. Enter number: 45.678
3. Enter number: -3.45
4. Enter number: 0
5. Enter number: 33.48
6. Enter number: -234.53
7. Enter number: 111.11
8. Enter number: 222.432
9. Enter number: 43.45
Average = 25.61
```

2. Display Largest Element of an array

```
#include <stdio.h>
int main()
{
   int i,n,Max;
   float arr[100];
   printf("Enter total number of elements(1 to 100): ");
   scanf("%d",&n);
   printf("\n");
   for(i=0;i<n;i++)
   {
      printf("Enter Number %d: ",i+1);
      scanf("%f",&arr[i]);
   }
  Max= arr[0];
for(i=1;i<n;i++)
   {
      if(Max<arr[i])
          Max=arr[i];
   }
   printf("Largest element = %d",arr[0]);

}
```

Output



```
Enter total number of elements(1 to 100): 12

Enter Number 1: 2.34
Enter Number 2: 3.43
Enter Number 3: 6.78
Enter Number 4: 2.45
Enter Number 5: 7.64
Enter Number 6: 9.05
Enter Number 7: -3.45
Enter Number 8: -9.99
Enter Number 9: 5.67
Enter Number 10: 34.953
Enter Number 11: 4.5
Enter Number 12: 3.45
Largest element = 34.95
```

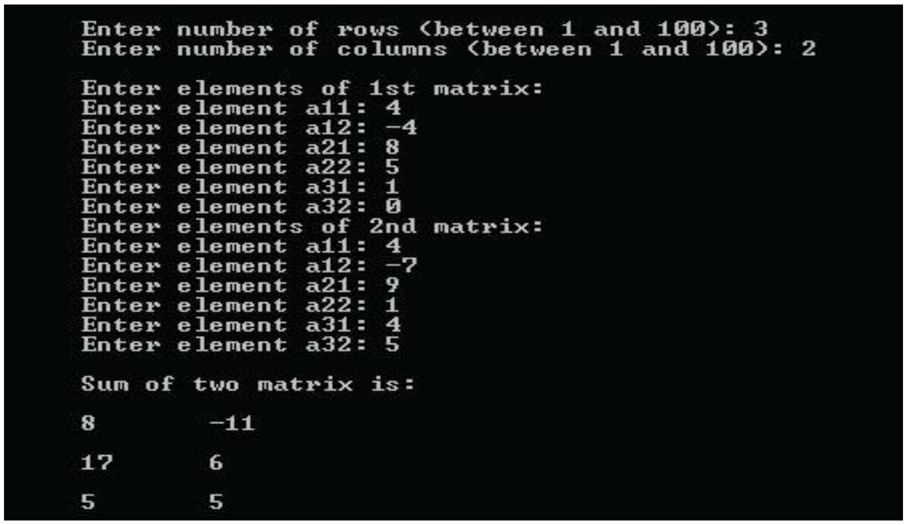
3. Display smallest Element of an array

```
#include <stdio.h>
int main()
{
   int i,n,Min;
   float arr[100];
   printf("Enter total number of elements(1 to 100): ");
   scanf("%d",&n);
   printf("\n");
   for(i=0;i<n;i++)
    {
       printf("Enter Number %d: ",i+1);
       scanf("%f",&arr[i]);
    }
   Min= arr[0];
for(i=1;i<n;i++)
    {
       if(Min>arr[i])
          {
Min=arr[i];
Loc= i+1;
   }
    printf("smallest element = %d",Min);
    printf("location = %d",Loc);

}
```

4. Addition of Two Dimensional Matrix

```
#include <stdio.h>
int main()
{
   int r,c,a[100][100],b[100][100],sum[100][100],i,j;
   printf("Enter number of rows (between 1 and 100): ");
   scanf("%d",&r);
   printf("Enter number of columns (between 1 and 100): ");
   scanf("%d",&c);
   printf("\nEnter elements of 1st matrix:\n");
   for(i=0;i<r;i++)
   {
            for(j=0;j<c;j++)
            {
                        printf("Enter element a%d%d: ",i+1,j+1);
                        scanf("%d",&a[i][j]);
            }
    }
   for(i=0;i<r;i++)
   {
            for(j=0;j<c;j++)
            {
                        printf("Enter element b%d%d: ",i+1,j+1);
                        scanf("%d",&b[i][j]);
            }
    }
    for(i=0;i<r;i++)
   {
              for(j=0;j<c;j++)
              {
                        sum[i][j]=a[i][j]+b[i][j];
           }
    }
   printf("\nSum of two matrix is: \n\n");
   for(i=0;i<r;i++)
    {
         for(j=0;j<c;j++)
        {
            printf("%d   ",sum[i][j]);
        } }
```

Output



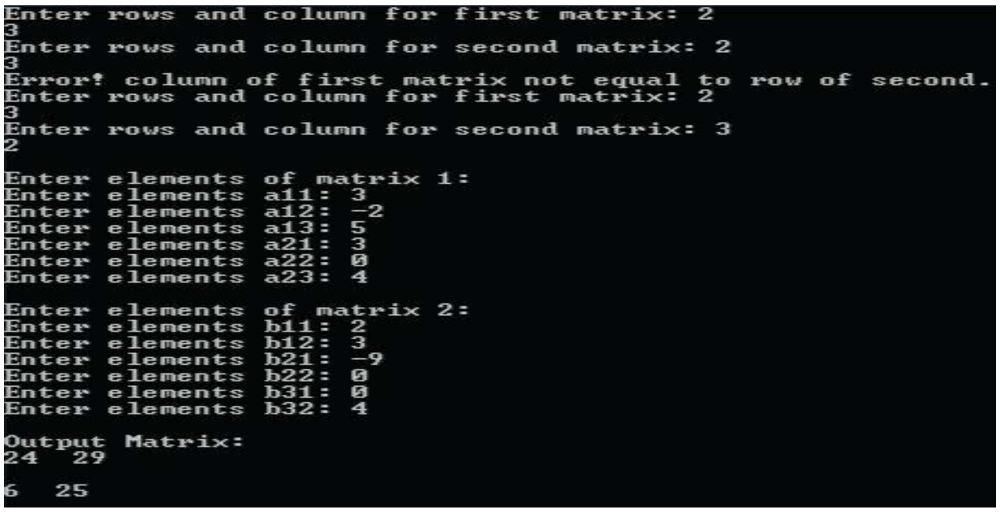
```
Enter number of rows (between 1 and 100): 3
Enter number of columns (between 1 and 100): 2

Enter elements of 1st matrix:
Enter element a11: 4
Enter element a12: -4
Enter element a21: 8
Enter element a22: 5
Enter element a31: 1
Enter element a32: 0
Enter elements of 2nd matrix:
Enter element a11: 4
Enter element a12: -7
Enter element a21: 9
Enter element a22: 1
Enter element a31: 4
Enter element a32: 5

Sum of two matrix is:

8       -11

17      6

5       5
```

5. Multiplication of Two Dimensional Matrix

```
#include <stdio.h>
int main()
{
   int a[10][10], b[10][10], mult[10][10], r1, c1, r2, c2, i, j, k;
   printf("Enter rows and column for first matrix: ");
   scanf("%d%d", &r1, &c1);
   printf("Enter rows and column for second matrix: ");
   scanf("%d%d",&r2, &c2);
/* Storing elements of first matrix. */
   printf("\nEnter elements of matrix 1:\n");
   for(i=0; i<r1; i++)
   {
            for(j=0; j<c1; j++)
            {
               printf("Enter elements a%d%d: ",i+1,j+1);
               scanf("%d",&a[i][j]);
            }
    }
/* Storing elements of second matrix. */
   printf("\nEnter elements of matrix 2:\n");
   for(i=0; i<r2; ++i)
   {
       for(j=0; j<c2; ++j)
            {
               printf("Enter elements b%d%d: ",i+1,j+1);
               scanf("%d",&b[i][j]);
            }
}
   for(i=0; i<r1; ++i)
   {
```

```
            for(j=0; j<c2; ++j)
              {
                mult[i][j]=0;
                for(k=0; k<c1; ++k)
                {
                   mult[i][j]= mult[i][j]+a[i][k]*b[k][j];
                }
              }
   }
   printf("\nOutput Matrix:\n");
   for(i=0; i<r1; i++)
   {
             for(j=0; j<c2; j++)
             {
                printf("%d  ",mult[i][j]);
          }
   }
}
```

Output



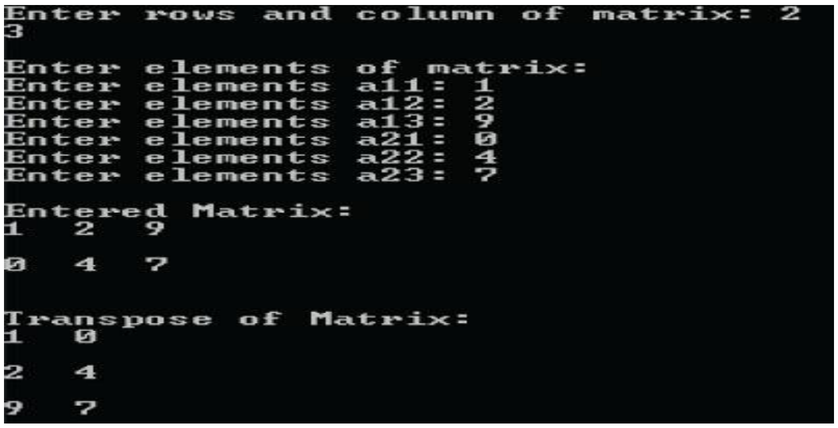
```
Enter rows and column for first matrix: 2
3
Enter rows and column for second matrix: 2
3
Error! column of first matrix not equal to row of second.
Enter rows and column for first matrix: 2
3
Enter rows and column for second matrix: 3
2

Enter elements of matrix 1:
Enter elements a11: 3
Enter elements a12: -2
Enter elements a13: 5
Enter elements a21: 3
Enter elements a22: 0
Enter elements a23: 4

Enter elements of matrix 2:
Enter elements b11: 2
Enter elements b12: 3
Enter elements b21: -9
Enter elements b22: 0
Enter elements b31: 0
Enter elements b32: 4

Output Matrix:
24  29

6  25
```

<u>6. To Find Transpose of a Matrix</u>

```
#include <stdio.h>
int main()
{
   int a[10][10], trans[10][10], r, c, i, j;
```

```
    printf("Enter rows and column of matrix: ");
    scanf("%d %d", &r, &c);
    printf("\nEnter elements of matrix:\n");
    for(i=0; i<r; ++i)
    {
        for(j=0; j<c; ++j)
            {
                printf("Enter elements a%d%d: ",i+1,j+1);
                scanf("%d",&a[i][j]);
            }
    }
    /* Displaying the matrix a[][] */
    printf("\nEntered Matrix: \n");
    for(i=0; i<r; ++i)
    {
            for(j=0; j<c; ++j)
            {
                printf("%d  ",a[i][j]);
            }
    }

   /* Finding transpose of matrix a[][] and storing it in array trans[][]. */
    for(i=0; i<r; ++i)
   {
             for(j=0; j<c; ++j)
                {
                  trans[i][j]=a[j][i];
                }
    }
    /* Displaying the transpose,i.e, Displaying array trans[][]. */
    printf("\nTranspose of Matrix:\n");
    for(i=0; i<c; ++i)
    {
             for(j=0; j<r; ++j)
             {
                printf("%d  ",trans[i][j]);
             }

}
```

Output



```
Enter rows and column of matrix: 2
3

Enter elements of matrix:
Enter elements a11: 1
Enter elements a12: 2
Enter elements a13: 9
Enter elements a21: 0
Enter elements a22: 4
Enter elements a23: 7

Entered Matrix:
1  2  9

0  4  7


Transpose of Matrix:
1  0

2  4

9  7
```

7. Count number of words and characters in string

```
#include<stdio.h>
#include<conio.h>
int main()
{   int count_words=0,i;
 int count_char=0;
 char str[20];
 printf("Enter string : ");
 gets(str);
 for(i=0; str[i]!='\0'; i++)
 {
   count_char++;
   if(str[i]==' ')
      count_words++;
 }
 printf("\nNumber of characters in string : %d",count_char);
 printf("\nNumber of words in string : % d",count_words+1);
 getch();
 return 0;
}
```

Output:

```
Enter string : kct college
Number of characters in string : 10
Number of words in string : 2
```

8. Find the Frequency of Characters

```
#include <stdio.h>
int main(){
   char c[1000],ch;
   int i,count=0;
```

```
  printf("Enter a string: ");
  gets(c);
  printf("Enter a character to find frequency: ");
  scanf("%c",&ch);
  for(i=0;c[i]!='\0';++i)
  {
     if(ch==c[i])
         ++count;
  }
  printf("Frequency of %c = %d", ch, count);
  return 0;
}
```

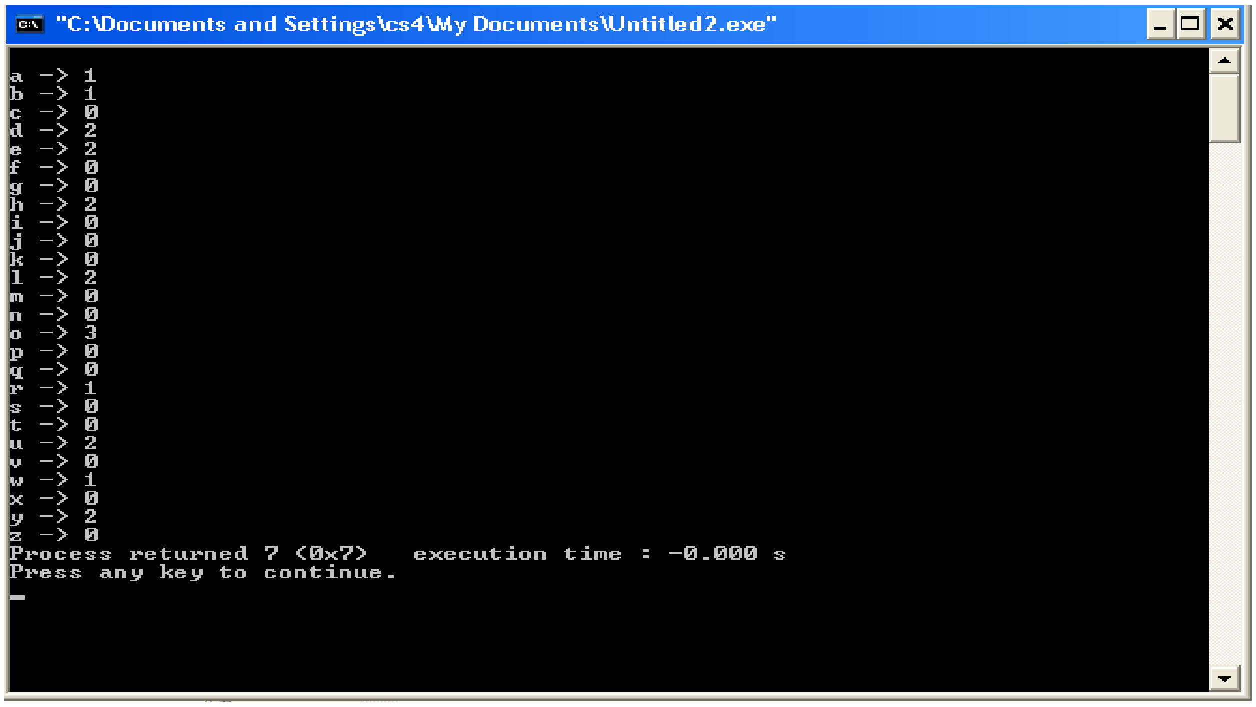
Output

```
Enter a string: Kumaraguru college
Enter a frequency to find frequency: a
Frequency of a=2
```

<u>9. Number of occurrence of character in string</u>

```
#include<stdio.h>
main()
{
   int i = 0,j=0,count[26]={0};
   char ch = 97;
   char string[100]="Hello how are you buddy ?";
   for (i = 0; i < 100; i++)
   {
      for(j=0;j<26;j++)
          {
           if (tolower(string[i]) == (ch+j))
                       {
           count[j]++;
                       }
            }
   }
   for(j=0;j<26;j++)
       {
           printf("\n%c -> %d",97+j,count[j]);
   }
}
```

Output:



```
a -> 1
b -> 1
c -> 0
d -> 2
e -> 2
f -> 0
g -> 0
h -> 2
i -> 0
j -> 0
k -> 0
l -> 2
m -> 0
n -> 0
o -> 3
p -> 0
q -> 0
r -> 1
s -> 0
t -> 0
u -> 2
v -> 0
w -> 1
x -> 0
y -> 2
z -> 0
Process returned 7 (0x7)   execution time : -0.000 s
Press any key to continue.
_
```

10. Reverse the string

```
 #include<stdio.h>
 #include<string.h>
 int main() {
 char str[100], temp;
 int i, j = 0;
 printf("\nEnter the string :");
gets(str);
 i = 0;
j = strlen(str) - 1;
while (i < j) {
   temp = str[i];
   str[i] = str[j];
   str[j] = temp;
   i++;
   j--;
}
 printf("\nReverse string is :%s", str);
return (0);
}
```

Output:

```
Enter the string  : Pritesh
Reverse string is : hsetirP
```

11. Find Number of Vowels, Consonants, Digits and White Space Character

```
#include<stdio.h>
int main(){
   char line[150];
   int i,v,c,ch,d,s,o;
   o=v=c=ch=d=s=0;
   printf("Enter a line of string:\n");
   gets(line);
   for(i=0;line[i]!='\0';++i)
   {
      if(line[i]=='a' || line[i]=='e' || line[i]=='i' || line[i]=='o' || line[i]=='u' || line[i]=='A' ||
line[i]=='E' || line[i]=='I' || line[i]=='O' || line[i]=='U')
          ++v;
      else if((line[i]>='a'&& line[i]<='z') || (line[i]>='A'&& line[i]<='Z'))
          ++c;
      else if(line[i]>='0'&&c<='9')
          ++d;
      else if (line[i]==' ')
          ++s;
   }
   printf("Vowels: %d",v);
   printf("\nConsonants: %d",c);
   printf("\nDigits: %d",d);
   printf("\nWhite spaces: %d",s);
   return 0;
}
```

Output

```
Enter a line of string:
This program is easy 2 understand
Vowels: 9
Consonants: 18
Digits: 1
White spaces: 5
```

12.Calculate Length without Using strlen() Function

```
#include <stdio.h>
int main()
{
   char s[1000],i;
   printf("Enter a string: ");
   scanf("%s",s);
   for(i=1; s[i]!='\0'; ++i);
```

```
    printf("Length of string: %d",i);
    return 0;
}
```

Output

```
Enter a string: Programiz
Length of string: 9
```

13. Concatenate Two Strings Manually

```
#include <stdio.h>
int main()
{
    char s1[100], s2[100], i, j;
    printf("Enter first string: ");
    scanf("%s",s1);
    printf("Enter second string: ");
    scanf("%s",s2);
    for(i=0; s1[i]!='\0'; ++i);  /* i contains length of string s1. */
    for(j=0; s2[j]!='\0'; ++j, ++i)
    {
        s1[i]=s2[j];
    }
    s1[i]='\0';
    printf("After concatenation: %s",s1);
    return 0;
}
```

Output

```
Enter first string: lol
Enter second string: :)
After concatenation: lol:)
```

14. Code to Copy String Manually

```
#include <stdio.h>
int main()
{
    char s1[100], s2[100], i;
    printf("Enter string s1: ");
    scanf("%s",s1);
    for(i=0; s1[i]!='\0'; ++i)
    {
        s2[i]=s1[i];
    }
    s2[i]='\0';
    printf("String s2: %s",s2);
    return 0;
}
```

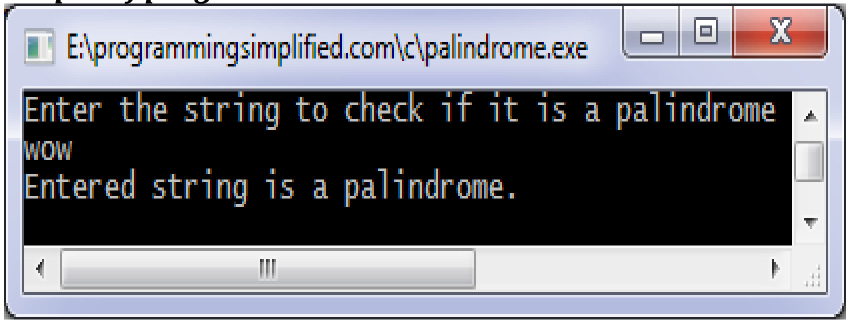
Output

```
Enter String s1: programiz
String s2: programiz
```

15. Find given string is palindrome or not

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

int main()
{
   char a[100], b[100];
    printf("Enter the string to check if it is a palindrome\n");
   gets(a);
    strcpy(b,a);
   strrev(b);
    if( strcmp(a,b) == 0 )
      printf("Entered string is a palindrome.\n");
   else
      printf("Entered string is not a palindrome.\n");
    return 0;}
```

Output of program:



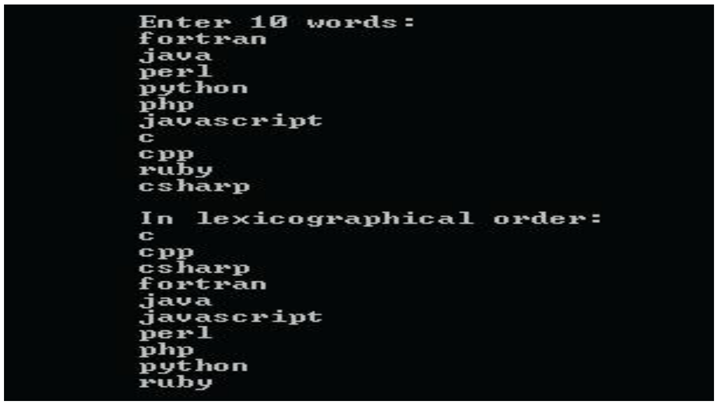
```
Enter the string to check if it is a palindrome
wow
Entered string is a palindrome.
```

16. To Sort Words in Dictionary Order

```
#include<stdio.h>
#include <string.h>
int main(){
    int i,j;
```

```
   char str[10][50],temp[50];
   printf("Enter 10 words:\n");
   for(i=0;i<10;++i)
       gets(str[i]);
   for(i=0;i<9;++i)
      for(j=i+1;j<10 ;++j){
         if(strcmp(str[i],str[j])>0)
         {
           strcpy(temp,str[i]);
           strcpy(str[i],str[j]);
           strcpy(str[j],temp);
         }
   }
   printf("In lexicographical order: \n");
   for(i=0;i<10;++i){
      puts(str[i]);
   }
return 0;
}
```

Output

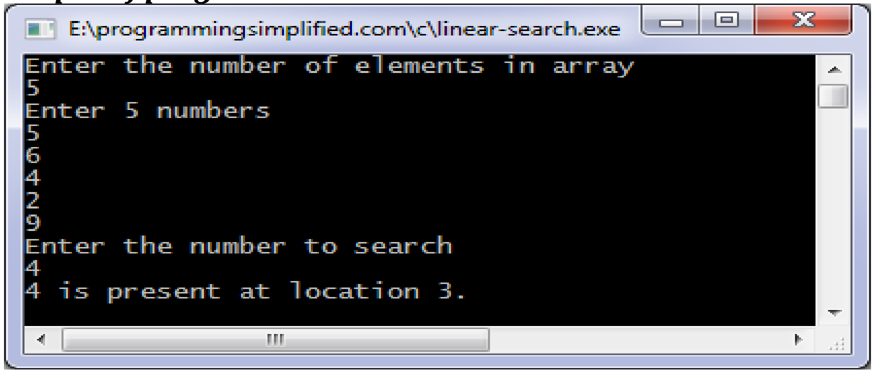


```
Enter 10 words:
fortran
java
perl
python
php
javascript
c
cpp
ruby
csharp

In lexicographical order:
c
cpp
csharp
fortran
java
javascript
perl
php
python
ruby
```

<u>17. Linear search c program</u>

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

  printf("Enter the 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 the number to search\n");
   scanf("%d", &search);
    for (c = 0; c < n; c++)
   {
     if (array[c] == search)     /* if required element found */
     {
       printf("%d is present at location %d.\n", search, c+1);
       break;
     }
   }
   if (c == n)
     printf("%d is not present in array.\n", search);

   return 0;
}
```

Output of program:



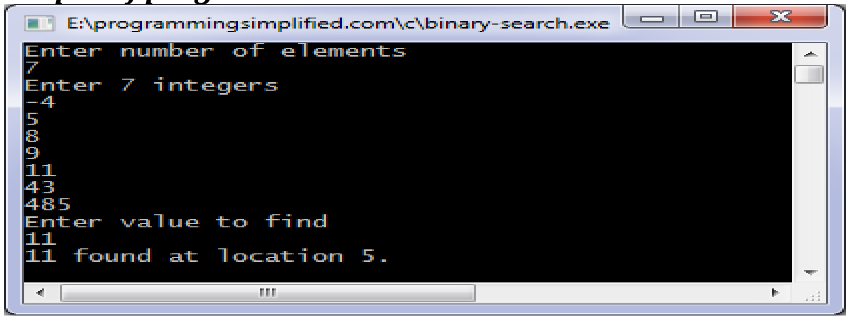
E:\programmingsimplified.com\c\linear-search.exe

```
Enter the number of elements in array
5
Enter 5 numbers
5
6
4
2
9
Enter the number to search
4
4 is present at location 3.
```

18. Binary search

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

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

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

   printf("Enter value to find\n");
   scanf("%d",&search);
   first = 0;
  last = n - 1;
  middle = (first+last)/2;

  while( first <= last )
  {
    if ( array[middle] < search )
      first = middle + 1;
    else if ( array[middle] == search )
    {
      printf("%d found at location %d.\n", search, middle+1);
      break;
    }
    else
      last = middle - 1;

    middle = (first + last)/2;
  }
  if ( first > last )
    printf("Not found! %d is not present in the list.\n", search);

  return 0;
}
```

Output of program:



```
E:\programmingsimplified.com\c\binary-search.exe
Enter number of elements
7
Enter 7 integers
-4
5
8
9
11
43
485
Enter value to find
11
11 found at location 5.
```

Binary search is faster than linear search but list should be sorted, hashing is faster than binary search and perform searches in constant time.

19. To Sort Elements using Bubble Sort Algorithm

```
#include <stdio.h>
int main()
{
   int data[100],i,j,n;
   printf("Enter the number of elements to be sorted: ");
   scanf("%d",&n);
   printf("enter the elements:");
   for(i=0;i<n;i++)
   {
      scanf("%d",&data[i]);
   }
   for(i=0;i<n;i++)
   {
            for(j=i+1;j<n;j++)
            {
                        if(data[i]>data[i+1])   /* To sort in descending order, change > to < in this
line. */
                        {
                    temp=data[i];
                    data[i]=data[j];
                    data[j]=temp;
                  }
               }
               printf("In ascending order: ");
               for(i=0;i<n;i++)
               {
                        printf("%d  ",data[i]);
               }
               printf("In descending order: ");
               for(i=n;i<=0;i--)
               {
                        printf("%d  ",data[i]);
               }
            Printf("the maximum number is: %d",a[n-1]);
            Printf("the minimum number is: %d",a[0]);
            }
```

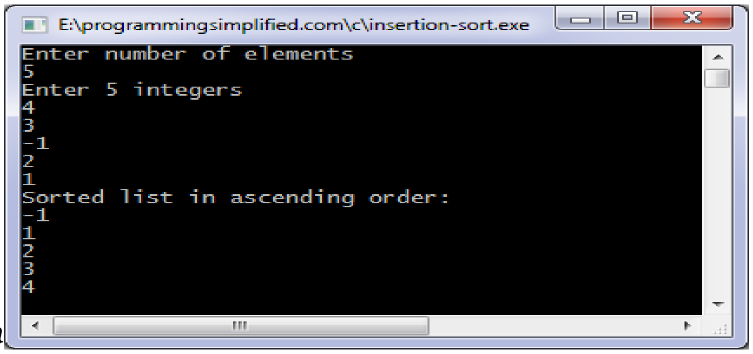
Output:

```
Enter the number of elements to be sorted: 6
12
3
0
-3
1
-9
In ascending order: -9 -3 0 1 3 13
```

In descending order:13 3 1 0 -3 -9
Maximum number is 13
Minimum number is -9

20. Insertion sort algorithm

```
/* insertion sort ascending order */
 #include <stdio.h>
 int main()
{
             int n, array[1000], c, d, t;
              printf("Enter number of elements\n");
             scanf("%d", &n);
              printf("Enter %d integers\n", n);
              for (c = 0; c < n; c++)
               {
                   scanf("%d", &array[c]);
                }
              for (c = 1 ; c <= n - 1; c++)
              {
               d = c;
                while ( d > 0 && array[d] < array[d-1])
               {
                            t  = array[d];
                            array[d]   = array[d-1];
                            array[d-1] = t;
                             d--;
                   }
             }
              printf("Sorted list in ascending order:\n");
              for (c = 0; c <= n - 1; c++)
            {
               printf("%d\n", array[c]);
             }
```



```
Enter number of elements
5
Enter 5 integers
4
3
-1
2
1
Sorted list in ascending order:
-1
1
2
3
4
```

```
            return 0;}}
```

FUNCTIONS, STORAGE CLASSES

Content: Function, Storage Class, Recursion

C functions are basic building blocks in a program. All C programs are written using functions to improve re-usability, understandability and to keep track on them.

1. What is C function?

A large C program is divided into basic building blocks called C function. **A function is a self contained block of program that performs a particular task.** C function contains set of instructions enclosed by "{ }" which performs specific operation in a C program. Actually, Collection of these functions creates a C program.

2. Uses and need of C functions:

- C functions are used to avoid rewriting same logic/code again and again in a program.
- There is no limit in calling C functions to make use of same functionality wherever required.
- We can call functions any number of times in a program and from any place in a program.
- A large C program can easily be tracked when it is divided into functions.
- The core concept of C functions are, re-usability, dividing a big task into small pieces to achieve the functionality and to improve understandability of very large C programs.
- Different programmers working on one large project can divide the workload by writimg different functions.

3. C function declaration, function call and function definition:

There are 3 aspects in each C function. They are,

- Function declaration or prototype - This informs compiler about the function name, function parameters and return value's data type.
- Function call – This calls the actual function
- Function definition – This contains all the statements to be executed.

S.no	C function aspects	Syntax

1	function definition	return_type function_name (arguments list)
		{
		Body of function;
		}

S.no	C function aspects	Syntax
2	function call	function_name (arguments list);
3	function declaration(prototype)	return_type function_name (argument list);

Simple example program for C function:

- Functions should be declared before calling in a C program.
- In the below program, function “square” is called from main function.
- The value of “m” is passed as argument to the function “square”. This value is multiplied by itself in this function and multiplied value “p” is returned to main function from function “square”.

```
#include<stdio.h>
//      function prototype, also called
function declaration float square ( float x );
//      main function, program starts from here

int main( )
{
      float m, n ;
      printf ( "\nEnter some number for finding square \n");
      scanf ( "%f", &m ) ;
      // function call
      n = square ( m ) ;
      printf ( "\nSquare of the given number %f is %f",m,n );
}

float square ( float x )  // function definition
{
      float p ;
      p = x * x ;
      return ( p ) ;
}
```

Output:

```
Enter some number for finding square  2
Square of the given number 2.000000 is 4.000000
```

4. How to call C functions in a program?

There are two ways that a C function can be called from a program. They are,

1. Call by value
2. Call by reference

1. Call by value:

- In call by value method, the value of the variable is passed to the function as parameter.

- The value of the actual parameter can not be modified by formal parameter.
- Different Memory is allocated for both actual and formal parameters. Because, value of actual parameter is copied to formal parameter.

Note:

- Actual parameter – This is the argument which is used in function call.
- Formal parameter – This is the argument which is used in function definition

Example program for C function (using call by value):

- In this program, the values of the variables "m" and "n" are passed to the function "swap".

- These values are copied to formal parameters "a" and "b" in swap function and used.

```
#include<stdio.h>
//        function prototype, also called function
declaration void swap(int a, int b);
int main()
{
   int m = 22, n = 44;
   // calling swap function by value
   printf(" values before swap m = %d \nand n = %d", m,
   n); swap(m, n);
}
```

```
void swap(int a, int b)
{
    int tmp;
    tmp = a;
    a = b;
    b = tmp;
    printf(" \nvalues after swap m = %d\n and n = %d", a, b);
}
```

Output:

```
values before swap m = 22
and n = 44

values after swap m = 44
and n = 22
```

2. Call by reference:

- In call by reference method, the address of the variable is passed to the function as parameter.

- The value of the actual parameter can be modified by formal parameter.
- Same memory is used for both actual and formal parameters since only address is used by both parameters.

Example program for C function (using call by reference):

- In this program, the address of the variables “m” and “n” are passed to the function “swap”.

- These values are not copied to formal parameters “a” and “b” in swap function.
- Because, they are just holding the address of those variables.
- This address is used to access and change the values of the variables.

```
#include<stdio.h>
//        function prototype, also called function
declaration void swap(int *a, int *b);

int main()
{

    int m = 22, n = 44;
    // calling swap function by reference
    printf("values before swap m = %d \n and n =
    %d",m,n); swap(&m, &n);

}

void swap(int *a, int *b)
{
    int tmp;
    tmp = *a;
    *a = *b;
    *b = tmp;
    printf("\n values after swap a = %d \nand b = %d", *a, *b);}
```

Output:

```
values before swap m = 22
and n = 44
values after swap a = 44
and b = 22
```

C- Argument, return value

All C functions can be called either with arguments or without arguments in a C program. These functions may or may not return values to the calling function. Now, we will see simple example C programs for each one of the below.

1. C function with arguments (parameters) and with return value
2. C function with arguments (parameters) and without return value
3. C function without arguments (parameters) and without return value
4. C function without arguments (parameters) and with return value

S.no	C function	Syntax

1	with arguments and with return values	int function (int); // function declaration
		function (a); // function call
		int function(int a) // function definition
		{statements; return a;}

2	with arguments and without return values	void function (int); // function declaration
		function(a); // function call
		void function(int a
		) // function definition
		{statements;}

3	without arguments and without return values	void function(); // function declaration
		
		function(); // function call
		void function() // function definition
		{statements;}

4	without arguments and with return values	int function ( ); // function declaration
		function ( ); // function call
		int function( ) // function definition
		{statements; return a;}

Note:

- If the return data type of a function is “void”, then, it can’t return any values to the calling function.

- If the return data type of the function is other than void such as “int, float, double etc”, then, it can return values to the calling function.

Example program for with arguments & with return value:

In this program, two integer values are passed to the function “test” and integer value is returned from this function to main function.

```
#include<stdio.h>

Int test(int,int);

void main()
{
int a = 50, b = 80,c;
c=test(a,b);
printf(“The value of c is:%d”,c);
}

Int test(int i ,int j)
{
Int x;
x=i+j;
return(x);
}
```

Output:

```
The value of c is:130
```

2. Example program for with arguments & without return value:

In this program, values are passed to the function “test” and no values are returned from this function to main function.

```
#include<stdio.h>
void test(int ,int);
void main()
{
int a = 50, b = 80;
test(a,b);
}
void test(int i,int j)
```

```
{
Int x;
x=i+j;
  printf("\nvalues : a = %d and b = %d",i,j);
printf("The sum is:%d",x);
}
```

Output:

```
values : a = 50 and b = 80      The sum is :130
```

3. Example program for without arguments & without return value:

In this program, no values are passed to the function "test" and no values are returned from this function to main function.

```
#include<stdio.h>

void test();

int main()
{
   test();
   return 0;
}

void test()
{
      int a = 50, b = 80;
      printf("\nvalues : a = %d and b = %d", a, b);
}
```

Output:

```
values : a = 50 and b = 80
```

4. Example program for without arguments & with return value:

In this program, no arguments are passed to the function "sum". But, values are returned from this function to main function. Values of the variable a and b are summed up in the function "sum" and the sum of these value is returned to the main function.

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

```
int sum();

int main()
{
   int addition;
   addition = sum();
   printf("\nSum of two given values = %d", addition);
   return 0;
}

int sum()
{
     int a = 50, b = 80, sum;

     sum = a + b;
     return sum;
}
```

Output:

```
Sum of two given values = 130
```

Do you know how many values can be return from C functions?

- Always, Only one value can be returned from a function.
- If you try to return more than one values from a function, only one value will be returned that appears at the right most place of the return statement.
- For example, if you use “return a,b,c” in your function, value for c only will be returned and values a, b won’t be returned to the program.
- In case, if you want to return more than one values, pointers can be used to directly change the values in address instead of returning those values to the function.

Types of functions in C:

- There are 2 types of functions in C. They are, library functions and user defined functions.
- Library functions are inbuilt functions which are available in common place called C library. Where as, User defined functions are the functions which are written by us for our own requirement.

<u>Library functions:</u>

- Library functions in C language are inbuilt functions which are grouped together and placed in a common place called library.

- Each library function in C performs specific operation.
- We can make use of these library functions to get the pre-defined output instead of writing our own code to get those outputs.
- These library functions are created by the persons who designed and created C compilers.
- All C standard library functions are declared in many header files which are saved as file_name.h.

- Actually, function declaration, definition for macros are given in all header files.
- We are including these header files in our C program using "#include<file_name.h>" command to make use of the functions those are declared in the header files.
   When we include header files in our C program using "#include<filename.h>" command, all C code of the header files are included in C program. Then, this C program is compiled by compiler and executed.

List of most used header files in C:

- o Check the below table to know all the C library functions and header files in which they are declared.

S.No	Header file	Description
1	**stdio.h**	This is standard input/output header file in which Input/Output functions are declared
2	**conio.h**	This is console input/output header file
3	**string.h**	All string related functions are defined in this header file
4	**stdlib.h**	This header file contains general functions used in C programs
5	**math.h**	All maths related functions are defined in this header file
6	**time.h**	This header file contains time and clock related functions
7	**ctype.h**	All character handling functions are defined in this header file
8	**stdarg.h**	Variable argument functions are declared in this header file
9	**signal.h**	Signal handling functions are declared in this file

10	**setjmp.h**	This file contains all jump functions

Adding user defined functions in C library:

- Do you know that we can add our own user defined functions in C library?
- Yes. It is possible to add, delete, modify and access our own user defined function to or from C library.
- The advantage of adding user defined function in C library is, this function will be available for all C programs once added to the C library.
- We can use this function in any C program as we use other C library functions.
- In latest version of GCC compilers, compilation time can be saved since these functions are available in library in the compiled form.
- Normal header files are saved as ”file_name.h” in which all library functions are available. These header files contain source code and this source code is added in main C program file where we add this header file using “#include <file_name.h>” command.
- Where as, precompiled version of header files are saved as “file_name.gch”.

Steps for adding our own functions in C library:

Step 1:

For example, below is a sample function that is going to be added in the C library. Write the below function in a file and save it as “addition.c”

```
addition(int i, int j)
{
int total;
total = i + j;
return total;
}
```

Step 2:

Compile “addition.c” file by using Alt + F9 keys (in turbo C).

step 3:

“addition.obj” file would be created which is the compiled form of “addition.c” file.

Step 4:

Use the below command to add this function to library (in turbo C).

```
c:\> tlib math.lib + c:\ addition.obj
```

+ means adding c:\addition.obj file in the math
library. We can delete this file using – (minus).

Step 5:

Create a file “addition.h” & declare prototype of addition() function like below. int addition (int i, int j);
Now addition.h file containing prototype of function “addition”.

Note : Please create, compile and add files in the respective directory as directory name may change for each IDE.

Step 6:

Let us see how to use our newly added library function in a C program.

```
# include <stdio.h>
//        Including our user defined function.
# include
“c:\\addition.h” int
main ()

{
      int total;
      // calling function from library
      total = addition (10, 20);
      printf ("Total = %d \n", total);
}
```

Output:

```
Total = 30
```

STORAGE CLASSES:

Storage class specifiers in C language tells the compiler where to store a variable, how to store the variable, what is the initial value of the variable and life time of the variable.

Syntax: storage_specifier data_type variable _name

Types of Storage Class Specifiers in C:

There are 4 storage class specifiers available in C language. They are,

1. auto
2. extern
3. static
4. register

Storage **Storage** **Initial /**

S.No.	Specifier	place	default value	Scope	Life
1	**auto**	CPU Memory	Garbage value	local	Within the function only.
2	**extern**	CPU memory	Zero	Global	Till the end of the main program. Variable definition might be anywhere in the C program
3	**static**	CPU memory	Zero	local	Retains the value of the variable between different function calls.
4	**register**	Register memory	Garbage value	local	Within the function

Note:

- For faster access of a variable, it is better to go for register specifiers rather than auto specifiers.
- Because, register variables are stored in register memory whereas auto variables are stored in main CPU memory.
- Only few variables can be stored in register memory. So, we can use variables as register that are used very often in a C program.

auto - Storage Class
auto is the default storage class for all local variables.

```
    {  int Count;
  auto int Month;}
```

The example above defines two variables with the same storage class. auto can only be used within functions, i.e. local variables.

Example program for auto variable in C:

The scope of this auto variable is within the function only. It is equivalent to local variable.
All local variables are auto variables by default.

```
#include<stdio.h>
void increment(void);

int main()
```

```
{
    increment();

    increment();
    increment();

    increment();
    return 0;

}
void increment(void)
{

    auto int i = 0 ;
    printf ( "%d ", i ) ;

    i++;
}
```

Output:

```
0 0 0 0
```

register - Storage Class

register is used to define local variables that should be stored in a register instead of RAM. This means that the variable has a maximum size equal to the register size (usually one word) and cant have the unary '&' operator applied to it (as it does not have a memory location).

```
  {
register int Miles;
  }
```

Register should only be used for variables that require quick access - such as counters. It should also be noted that defining 'register' goes not mean that the variable will be stored in a register. It means that it MIGHT be stored in a register - depending on hardware and implementation restrictions.

Example program for register variable in C:

- Register variables are also local variables, but stored in register memory. Whereas, auto variables are stored in main CPU memory.

- Register variables will be accessed very faster than the normal variables since they are stored in register memory rather than main memory.

- But, only limited variables can be used as register since register size is very low. (16 bits, 32 bits or 64 bits)

```
#include <stdio.h>

int main()
{

    register int i;
    int arr[5];                  // declaring array

    arr[0] = 10; // Initializing array arr[1] = 20;
    arr[2] = 30;
    arr[3] = 40;

    arr[4] = 50;
    for (i=0;i<5;i++)

    {
        // Accessing each variable

        printf("value of arr[%d] is %d \n", i, arr[i]);
    }

    return 0;
}
```

Output:

```
value of arr[0] is 10
value of arr[1] is 20
value of arr[2] is 30
value of arr[3] is 40
value of arr[4] is 50
```

static - Storage Class

static is the default storage class for global variables. The two variables below (count and road) both have a static storage class.

```
static int Count;
     int Road;

     {printf("%d\n", Road);}
```

static variables can be 'seen' within all functions in this source file. At link time, the static variables defined here will not be seen by the object modules that are brought in.static can also be defined within a function. If this is done the variable is initalised at run time but is not reinitalized when the function is called. This inside a function static variable retains its value during vairous calls.

```
void func(void);

static count=10; /* Global variable - static is the default */

main()
{
  while (count--)
  {func();}}

void func( void )
{
  static i = 5;
  i++;
  printf("i is %d and count is %d\n", i, count);
}
```

This will produce following result

```
i is 6 and count is 9
i is 7 and count is 8
i is 8 and count is 7
i is 9 and count is 6
i is 10 and count is 5
i is 11 and count is 4
i is 12 and count is 3
i is 13 and count is 2
i is 14 and count is 1
i is 15 and count is 0
```

Example program for static variable in C:

Static variables retain the value of the variable between different function calls.

```
//C static example
#include<stdio.h>

void increment(void);

int main()
```

```
{
    increment();

    increment();
    increment();

    increment();
    return 0;

}

void increment(void)
{

    static int i = 0 ;
    printf ( "%d ", i ) ;

    i++;
}
```

Output:

```
0 1 2 3
```

NOTE : Here keyword *void* means function does not return anything and it does not take any parameter. You can memorize void as nothing. static variables are initialized to 0 automatically.

Definition vs Declaration : Before proceeding, let us understand the difference between *definition* and *declaration* of a variable or function. Definition means where a variable or function is defined in reality and actual memory is allocated for variable or function. Declaration means just giving a reference of a variable and function. Through declaration we assure to the complier that this variable or function has been defined somewhere else in the program and will be provided at the time of linking. In the above examples *char func(void)* has been put at the top which is a declaration of this function whereas this function has been defined below to *main()*function.

extern - Storage Class

extern is used to give a reference of a global variable that is visible to ALL the program files. When you use 'extern' the variable cannot be initalized as all it does is point the variable name at a storage location that has been previously defined.

When you have multiple files and you define a global variable or function which will be used in other files also, then *extern* will be used in another file to give reference of defined variable or function. Just for understanding *extern* is used to decalre a global variable or function in another files.

File 1: main.c

```
int count=5;
```

```
  main()
  {
    write_extern();
  }
```

File 2: write.c

```
  void write_extern(void);

  extern int count;

  void write_extern(void)
  {
    printf("count is %i\n", count);
  }
```

Here *extern* keyword is being used to declare count in another file.

Count in 'main.c' will have a value of 5. If main.c changes the value of count - write.c will see the new value.

Example program for extern variable in C:

The scope of this extern variable is throughout the main program. It is equivalent to global variable. Definition for extern variable might be anywhere in the C program.

```
#include<stdio.h>
int x = 10 ;

int main( )
{

      extern int y ;
      printf ( "The value of x is %d \n", x ) ; printf (
      "The value of y is %d",y ) ; return 0;

}
int y = 50 ;
```

Output:

```
The value of x is 10
The value of y is 50
```

RECURSION:

A function that calls itself is known as recursive function and this technique is known as recursion in C programming.

Example of recursion in C programming

Write a C program to find sum of first n natural numbers using recursion. Note: Positive integers are known as natural number i.e. 1, 2, 3....n

```
#include <stdio.h>
int sum(int n);
int main(){
    int num,add;

    printf("Enter a positive integer:\n");
    scanf("%d",&num);
    add=sum(num);
    printf("sum=%d",add);
}
int sum(int n){
    if(n==0)
       return n;
    else
       return n+sum(n-1);     /*self call to function sum() */}
```

Output

```
Enter a positive integer:
5
15
```

In, this simple C program, sum() function is invoked from the same function. If *n* is not equal to 0 then, the function calls itself passing argument 1 less than the previous argument it was called with. Suppose, *n* is 5 initially. Then, during next function calls, 4 is passed to function and the value of argument decreases by 1 in each recursive call. When, n becomes equal to 0, the value of n is returned which is the sum numbers from 5 to 1. For better visualization of recursion in this example:

```
sum(5)
=5+sum(4)
=5+4+sum(3)
=5+4+3+sum(2)
=5+4+3+2+sum(1)
=5+4+3+2+1+sum(0)
=5+4+3+2+1+0
=5+4+3+2+1
=5+4+3+3
```

```
=5+4+6
=5+10
=15
```

Every recursive function must be provided with a way to end the recursion. In this example when, *n* is equal to 0, there is no recursive call and recursion ends.

Advantages and Disadvantages of Recursion

Recursion is more elegant and requires few variables which make program clean. Recursion can be used to replace complex nesting code by dividing the problem into same problem of its sub-type.

In other hand, it is hard to think the logic of a recursive function. It is also difficult to debug the code containing recursion

FACTORIAL:

```
#include<stdio.h>
int fact(int);
int main()
{
  int num,f;
  printf("\nEnter a number: ");
  scanf("%d",&num);
  f=fact(num);
  printf("\nFactorial of %d is: %d",num,f);
  return 0;
}

int fact(int n)
{
   if(n==1)
       return 1;
   else
       return(n*fact(n-1));  }
```

FIBONACCI SERIES:

```
#include<stdio.h>
void printFibonacci(int);
int main()
{
    int k,n;
    long int i=0,j=1,f;
    printf("Enter the range of the Fibonacci series: ");
    scanf("%d",&n);
    printf("Fibonacci Series: ");
    printf("%d %d ",0,1);
    printFibonacci(n);
    return 0;
}
void printFibonacci(int n)
{
    static long int first=0,second=1,sum;
    if(n>0)
   {
         sum = first + second;
         first = second;
         second = sum;
         printf("%ld ",sum);
         printFibonacci(n-1);
    }
}
```

Sample output:

```
Enter the range of the Fibonacci series: 10
Fibonacci Series: 0 1 1 2 3 5 8 13 21 34 55 89
```