MODULE-III

Arrays, Strings and Functions.

Definition of Array:

An array is a Single Variable to Store a number of Values with the Same data type.

The elements of an array are Stored in Consecutive Storage locations in memory. The name of an array is followed by a number in bracket [ ] these numbers Start at 0

Example: int A[5];

Integer Array

A[0]	10
A[1]	20
A[2]	30
A[3]	40
A[4]	50

int A[5];

Float Array

X[0]	1.1
X[1]	1.2
X[2]	2.1
X[3]	2.2
X[4]	4.5

float X[5];

Character Array

C[0]	'A'
C[1]	'B'
C[2]	'C'
C[3]	'D'
C[4]	'E'

Char C[5];

Invalid way of Storing array elements:-

int num[5];

15	8.5	10	"ABC"	50.
num[0]	num[1]	num[2]	num[3]	num[4]

Two types of an arrays:

* Single-dimensional array
* Multi-dimensional array.

Single-Dimensional Array: (one dimensional)

A single dimensional array is a linear list consisting of related data items of same data type.

Basic Properties Of Arrays:

* Array element should be of same datatype
* Data items are stored contigously in memory
* Subscript of first item is always zero.
* The index of array is always integer.
* Each element in an array is accessed through name of the array.

Example:- Char name[10] = "WELL DONE";

'W'	name[0]
'E'	name[1]
'L'	name[2]
'L'	name[3]
' '	name[4]
'D'	name[5]
'O'	name[6]
'N'	name[7]
'E'	name[8]

Declaration of Single Dimensional Arrays:

Syntax:

Name of an array

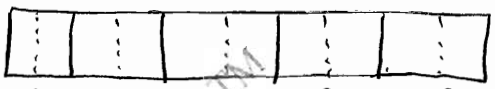
type arrayname [intexpression];

type → data type as int, float, char

intexpression → Expression must be evaluated to integer.

Example:

int a[5];

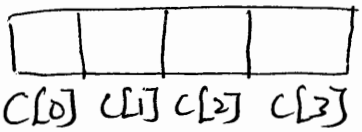


10 Memory Locations Reserved

float x[5];

20 bytes of Space Reserved.

Char c[4];



4 bytes of memory Reserved.

Storing Values in an Array:

The Values Can be Stored in an array using three methods.

* Initialization
* Assigning Values
* Input Values from Key Board.

Initialization of Single dimensional Arrays:-

Assigning the required Values to a Variable before Processing is Called initialization. Array elements Can be initialized at the time of declaration.

Syntax:

Name of the array.

array values

```
type  arrayname[expression] = {v1, v2, .., vn};
```

type → data type as int, float, char.

expression → Positive integer value. (index).

The Various ways of initializing arrays are:-

(i) Initializing all Specified memory locations

(ii) Partial Array initialization.

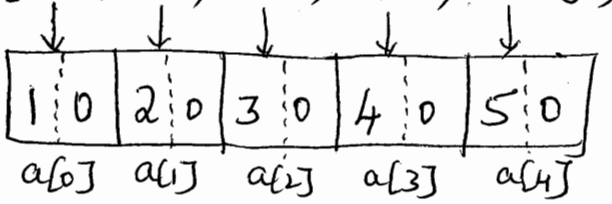
(iii) Initializing without Size.

(iv) String initialization.

(i) Initializing all Specified memory locations.

Arrays Can be initialized at the time of declaration when their initial Values are Known

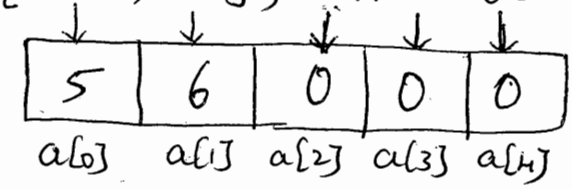
Example:- int a[5]={10, 20, 30, 40, 50};



Partial Array Initialization:

If number of values to be initialized is less than the size of an array, then elements are initialized in order from 0th location. The remaining blank locations initialized to zero.

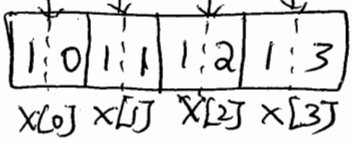
Example:- int a[5]={5, 6};



Initialization without Size:-

```
int x[] = {10, 11, 12, 13};
```

Size not specified.

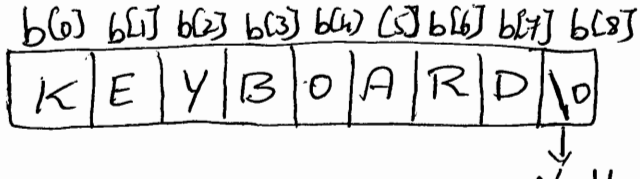


Here eventhough size not specified. the array size will be set to total number of initial values.

Array Initialization with String:

A sequence of characters enclosed within a double Quotes is a string. The string always

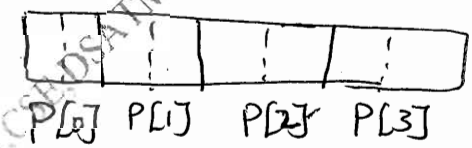
Example: Char b[] = "KEYBOARD";



Assigning Values to array:-

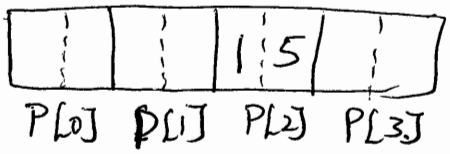
Here we can assign individual elements of an array using Assignment Operator (=).

Example:- int P[4];



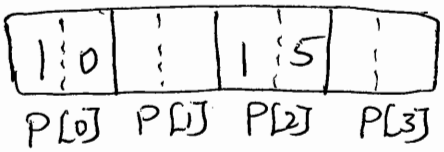
Assume array elements 15, 10, 25, 50 can be inserted into array at positions 2, 0, 3, 1

p[2] = 15;

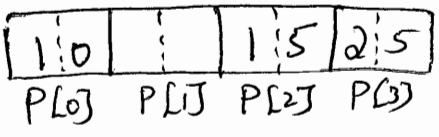


Value 15 is stored at array position 2.

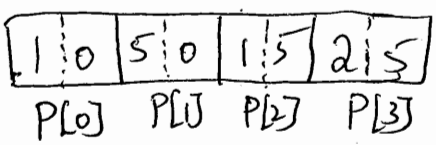
p[0] = 10;



p[3] = 25;



p[1] = 50;



Reading/Writing Single Dimensional Arrays:-

The read, write or Process the array items can be done through for-loop, while-loop, if-Statement, Switch Statement, etc.

To read n data items from keyboard an inbuilt library function is used Scanf()

Example: (i)
```
Scanf("%d", &a[0]);
Scanf("%d", &a[1]);
- - - -
- - - -
Scanf("%d", &a[n-1]);
```

(ii)
```
for(i=0; i<n; i++)
{
  Scanf("%d", &a[i]);
}
```

To display n data items Stored in an array can be done through Printf() library function.

```
for(i=0; i<n; i++)
{
  Printf("%d", a[i]);
}
```

The C-Program to read n items & display n-items on monitor:

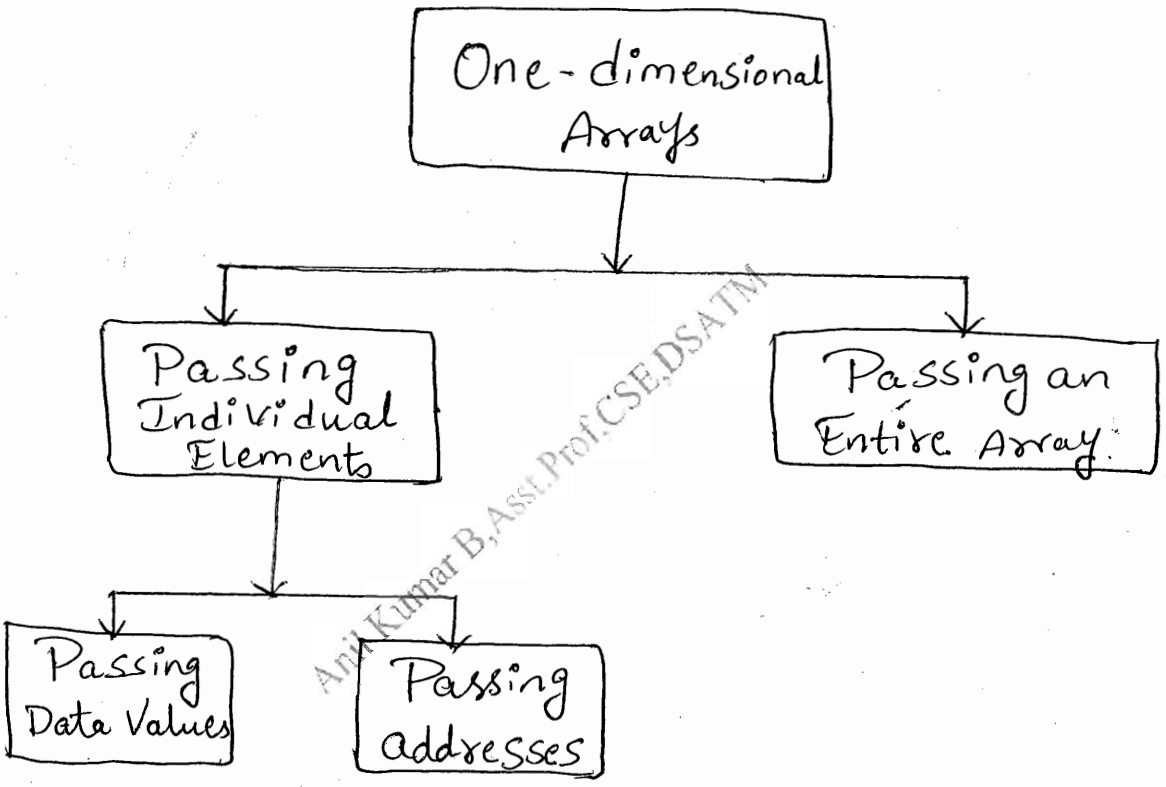
```
#include <stdio.h>
Void main()
{
   int n, a[10], i;

   Printf("Enter no. of elements \n");
   Scanf("%d", &n);
   Printf("Enter elements of array \n");
   for(i=0; i<n; i++)
    {
      Scanf("%d", &a[i]);
    }
   Printf("N elements are \n");
   for(i=0; i<n; i++)
   {
      Printf("%d", a[i]);
   }
}
```


Output:- Enter no. of elements : 4
Enter elements of array:
5 10 15 20
The N elements are:

Arrays and Functions:-

The arrays can be Passed to functions. Using two methods.

* Passing individual elements to array
* Passing the Whole array



Passing Individual Element to an Array:-

The individual elements of an array Can be Passed to a function either Passing their address or their data Values.

Passing data Values:

The individual elements Can be Passed as Variables of any other data type. Here the data type of an array element must match the

datatype of the function Parameter.

Example:

```
Void main()  ← CALLING FUNCTION
{
    int arr[5]={1,2,3,4,5};
    func(arr[3]);
}
```

CALLED FUNCTION →

```
Void func(int num)
{
    Printf("%d", num);
}
```

arr[0]	1
arr[1]	2
arr[2]	3
arr[3]	4
arr[4]	5

Output: 4

Passing Addresses:

Here we can Pass the address of an individual array element by Preceeding the address operator "&" to the element's indexed reference.

In the calling function the Value of an array is Passed through "&" and in the Called function the Value of the array element must be accessed using indirection operator (*)

Example:

```
main() ← CALLING FUNCTION
{
    int arr[5]={10,20,30,40,50};
    func(&arr[4]);
}
Void func(int *num)
{
    Printf("%d", num);
}
```

CALLED FUNCTION

arr[0]	10
arr[1]	20
arr[2]	30
arr[3]	40
arr[4]	50

Output: 50

Passing Entire array to Function:

The function Called by Passing Only the name of the array.

In function definition, the Parameter must be declared as an array of Same datatype as actual Parameter.

Example:

```
main()
{
    int arr[5]={5,10,15,20,25};
    func(arr);
}
```

CALLING FUNCTION

arr[0]	5
arr[1]	10
arr[2]	15
arr[3]	20

```
Void func (int arr[5])
{
    int i;
    for (i = 0; i<=5; i++)
    Printf(" %d", &arr[i]);
}
```

→ CALLED FUNCTION.

Note:

(i) The variable arr in the calling function declared as an array of 5 integers.

(ii) When function is called ie func the array arr is passed as an argument.

Example Program:

```
#include <stdio.h>
Void main()
{
    int n, a[10], i;
    Printf(" Enter Value of n\n");
    Scanf(" %d", &n);
    Printf(" Enter elements \n");
    for (i=0; i<n; i++)
    {
        Scanf(" %d", &a[i]);
    }
```

```
Printf (" Square of n items \n");
  for(i=0; i<n; i++)
  {
      Print_Square(a[i]);
  }
}
Void Print_Square (int x)
{
   Printf("%d", x*x);
}
```

Output:

Enter number of elements.

4

Enter n elements.

2 4 5 6

a[0] a[1] a[2] a[3]

Output Square of n items

4 16 25 36.

Output array

a[0]	4
a[1]	16
a[2]	25
a[3]	36.

Here, in the above example Program We Pass an individual array element to a function. The function receives these values using formal Parameters.

Program-2: C Program to Generate Fibonacci Series / numbers using arrays:

1st Fibonacci num	a[0]	=	0 Initial Value
2nd Fibonacci	a[1]	=	1 Initial Value
3rd Fibonacci	a[2]	=	a[1] + a[0] = 1
4th Fibonacci	a[3]	=	a[2] + a[1] = 2
.....	⋮		
nth Fibonacci	a[n-1]	=	a[n-2] + a[n-3]

In general → a[i] = a[i-1] + a[i-2]


```
#include <stdio.h>
Void Fibonacci (int a[], int n)
{
    int i;
    a[0] = 0;   /* Initial Fibonacci numbers */
    a[1] = 1;
    for (i=2; i<=n ; i++)
    {
        a[i] = a[i-1] + a[i-2];
    }
}
```

```
Void main()
{
    int n, i, a[50];
    Printf(" enter value for n \n");
    Scanf("%d", &n);
    Fibonacci (a, n);
    Printf(" Fibonacci Numbers are \n");
    for(i=0; i<=n; i++)
    {
      Printf("%d \n", a[i]);
    }
}
```


Output:

	Output
a[0]	0
a[1]	1
a[2]	1
a[3]	2
a[4]	3

Output:

Fibonacci Numbers are

a[0]	a[1]	a[2]	a[3]	a[4]
0	1	1	2	3

Definition of Fibonacci Numbers:

The fibonacci numbers are a Series of numbers such that each number is the sum of previous two numbers except the first and second number.

Two Dimensional Arrays: (Multi Dimensional)

An arrays with two sets of Square Brackets [][] ar called two-dimensional arrays.

A two-dimensional array is used when data items are arranged in row-wise and Column wise.

Declaration of Two dimensional Array:-

Syntax:-

datatype arrayname[exp1][exp2];

Where; exp1 - row size
exp2 - Column size
datatype - int, float, char, etc.

Example:-

int a[3][4];

	Col-0	Col-1	Col-2	Col-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]

The array a declared as two dimensional array with two square brackets with row size as 3 and Column Size as 4

Initialization of Two-dimensional Arrays:-

Assigning a required Value to a Variable before Processing is initialization.

Syntax:

```
datatype arrayname[exp1][exp2] =
        {
            {a1, a2 ... an},
            {b1, b2 ... bn},
            .....
            {z1, z2 ... zn}.
        };
```

Here, a_1 a_2 ... a_n are Values assigned to 1st row.

b_1, b_2 ... b_n are Values assigned to 2nd row. and so on.

Two Ways Of initialization are.

* Initializing all Specified memory Locations
* Partial array initialization.

Initializing Specified memory locations:

```
int arr[4][3]={
```

Columns →

rows ↓	0	1	2
0	5	10	15
1	1	2	3
2	5	6	7
3	8	9	10

```
{ 5, 10, 15},
{ 1, 2, 3},
{5, 6, 7},
{ 8, 9 ,10}
};
```


Partial Array initialization :-

If number of values to be initialized are less than the size of the array, then the elements are initialized from left to right one after the other.

Example:-

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

Columns →

rows ↓	0	1	2
0	5	10	0
1	6	8	0
2	4	3	0
3	9	11	0

Reading and Writing two Dimensional Arrays:

To read 2-d matrix Z of size m x n. and to Print a 2-d matrix Z of size m x n we have the following steps.

Step 1: Identify Parameters to function.

Given the size of matrix m x n we have to read elements into matrix Z or Print elements of matrix Z..

Parameters : int Z[][10], m, n

Step 2: Return type: After reading matrix or Printing a matrix we are not returning any value and hence,

return type : Void

Step 3: Designing body of the function:

Any item can be accessed by specifying the row index and column index, we Access the items row by row.

Element accessed by Z[i][j]

index Variable i = 0, 1, 2 .... m-1.

index Variable j = 0, 1, 2 .... n-1

```
for(i=0;i<=m-1;i++)
{
  for(j=0;j<=n-1;j++)
    {
      Z[i][j];
    }
}
```

⇨

```
for(i=0;i<m;i++) /* m-rows */
{
  for(j=0;j<n;j++) /* n-columns */
    {
      Z[i][j]; /* access */
    }
}
```

C - function to read matrix of size m x n

```
Void readmatrix(int Z[][10], int m, int n)
{
    int i;
    int j;
    for(i=0;i<m;i++)
    {
        for(j=0;j<m;j++)
        {
            scanf("%d", &Z[i][j]);
        }
    }
}
```


Here, when we Pass 2-dimensional array row size is optional, but Column size must be specified

C - function to Print a matrix of size m x n.

```
Void Printmatrix(int Z[][10], int m, int n)
{
    int i, j;
    for(i=0; i<m; i++)
    {
        for(j=0; j<n; j++)
        {
            Printf("%d", Z[i][j]);
        }
        Printf("\n");
    }
}
```


Addition of two matrices:-

The Various Steps to be followed to add two matrices as.

Step 1: Identify Parameters to function.

```
int A[][10], int B[][10], int C[][10],
    int m, int n;
```

Step 2: Return type

Returntype: Void.

Step 3: Body of function.

```
C[i][j] = a[i][j] + b[i][j].
```

C - Function to add two Matrices:

```
Void addmatrix(int a[][10], int b[][10], int c[][10],
               int m, int n)
{  int i, j;
   for(i=0; i<m; i++)
   {
      for(j=0; j<n; j++)
      {
         c[i][j]= a[i][j]+b[i][j]; /* Add matrix
                                      A, B */
      }
   }
}
```


Steps Performed to add two matrices:

Step 1: Read the Size of matrices m x n

Step 2: Read matrix A

Step 3: Read matrix B

Step 4: Add matrixes Matrix C = Matrix A + Matrix B

Step 5: Output/Print Matrix C

Step 6: Finished.

Programming Examples :-

Single dimensional array (One dimensional):

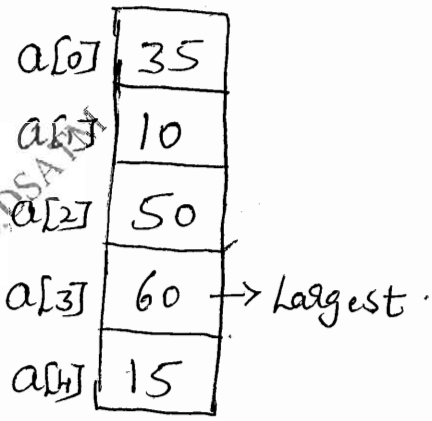
Program 1: C Program to Print Largest and its Position Using Arrays.

```
#include <Stdio.h>
int Largest (int a[], int n)
{
    int i, Pos;
    Pos=0;
    for(i=1; i<n; i++)
    {
      if (a[i] > a[Pos]) Pos=i;
    }
    return Pos;
}
Void main()
{
    int a[10], n, i, Pos;
    Printf (" Enter number of elements \n");
    Scanf(" %d", &n);
    Printf(" Enter %d elements ", n);
    for(i=0; i<n; i++)
    {
      Scanf ("%d", &a[i]);
    }
```

```
Pos = Largest (a, n);
 Printf(" Largest = %d ", a[Pos]);
 Printf(" Position = %d", Pos+1);
}
```

OutPut:

```
Enter number of elements:
 5
Enter 5 elements
  35  10  50  60  15
Pos = 3
Largest = 60
Position = 4
```



Searching:

The Process of finding a Particular item is Present in Large amount of data called Searching.

The two Searching techniques are.

- Linear Search.
- Binary Search.

Program 2: C Program to implement Linear Search

```
#include <Stdio.h>
int linear(int key, int a[], int n)
{
    int i;
    for(i=0; i<n; i++)
    { if (key == a[i]) return i;
    }
    return -1;
}
Void main()
{
    int n, a[20], key, i, Pos;
    Printf("Enter Value of n");
    Scanf("%d", &n);
    Printf("Enter %d Values", n);
    for(i=0; i<n; i++)
     Scanf("%d", &a[i]);
   Printf("Enter item key to be searched");
    Scanf("%d", &key);
    Pos = Linear(key, a, n);
```

```
if (Pos == -1)
  Printf (" Item not found ");
else
  Printf (" Item found ");
}
```

OutPut:

```
Enter Value of n
5
Enter 5 Values.
5  10  15  12  20
Enter item key to Search.
15
Item found.
```


Advantages Of Linear Search:

* Very Simple approach.
* Suitable for Small arrays
* Element Can be Searched from Unsorted array.

Disadvantages:

* If array size is more then it is Less efficient.
* For an element already Sorted linear Search is not efficient.

Program 3: C-Program to Search an item Using Binary Search technique.

```
#include <Stdio.h>
int binarySearch(int Key, int a[], int n)
{
    int low, high, mid;
     low = 0;   high = n-1;
    While (low <= high)
    {
      mid = (low+high)/2;
      if(Key == a[mid]) return mid;
       if (key < a[mid])
           high = mid -1;     /* Search left Part */
        if (key > a[mid])
             low = mid+1;     /* Right Part */
      }
    return -1;
  }
Void main ( )
{
    int n, a[10], Key, Pos;
    Printf (" Enter number of elements \n");
    Scanf (" %d", &n);
```

```
Printf(" Enter elements in ascending order");
for(i=0; i<n; i++)
  Scanf(" %d", &a[i]);
Printf(" Enter element to Search \n");
 Scanf(" %d", &Key);
 Pos= binarySearch (Key, a, n);
 if( Pos == -1)
      Printf(" item not found ");
    else
     Printf(" Item found at %d Position", Pos);
}
```


OutPut:

```
Enter number of elements
 4
Enter elements in ascending order
  10  15  20  25
Enter item to be Searched.
 15
Item found at 3 Position.
```

Disadvantage:

* List of elements to be Sorted array.

Program4: C-Program to implement Bubble Sort technique to sort the elements of an array.

```
#include<Stdio.h>
Void bubbleSort(int a[], int n)
{
    int i, j, temp;
    for(i=1; i<n; i++)
    {
        for(j=0; j<n-i; j++)
        {
            if (a[j] >= a[j+1])
            {
                temp = a[j];
                a[j] = a[j+1];
                a[j+1] = temp;
            }
        }
    }
}
Void main()
{
    int a[20], i, n;
    Printf(" Enter number of items \n");
    Scanf("%d", &n);
    Printf(" enter items to sort \n");
    for(i=0; i<n; i++)
```

```
{
    Scanf("%d", &a[i]);
}
Printf("before Sort \n");
for(i=0; i<n; i++)
{
  Printf("%d", a[i];
}
bubbleSort(a, n);
Printf("After Sort \n");
for(i=0; i<n; i++)
{
    Printf("%d", a[i]);
}
}
```


OutPut:

```
Enter number of items:
5
Enter elements to Sort
10  5  8  15  25

Before Sort
10  5  8  15  25

After Sort
5  8  10  15  25.
```

Programming Examples:

Two Dimensional or Multi Dimensional Arrays:

C Program to add two matrices:

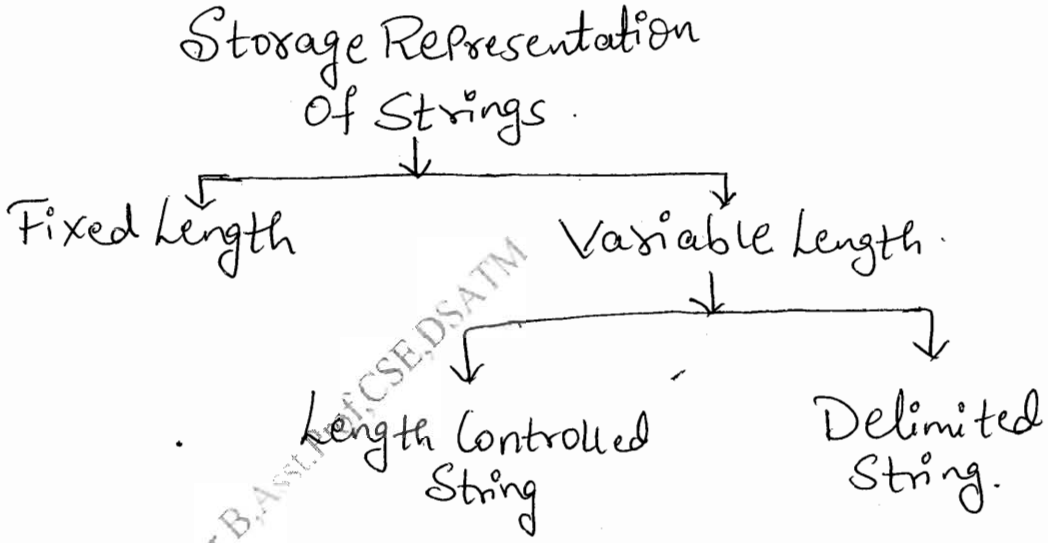
```
#include <stdio.h>

Void read (int Z[][10], int m, int n)
{
    int i, j;
    for (i=0; i<m; i++)
    {
        for (j=0; j<n; j++)
        {
            Scanf("%d", &Z[i][j]);
        }
    }
}

Void display (int Z[][10], int m, int n)
{
    int i, j;
    for (i=0; i<m; i++)
    {
        for (j=0; j<n; j++)
        {
            Printf(" %d", Z[i][j]);
        }
    }
}

Void addmatrix (int a[][10], int b[][10], int c[][10],
                int m, int n)
{
```

```
    int i, j;
    for (i=0; i<m; i++)
    {
      for( j=0; j<n; j++)
      {
        c[i][j] = a[i][j] + b[i][j];
      }
    }
}.
Void main()
{
    int m, n, a[10][10], b[10][10], c[10][10];
    Printf(" Enter the Size of matrix ");
    Slanf(" %d %d", &m, &n);
    Printf(" Enter elements of matrix A ");
      read (a, m, n);
     Printf(" Enter elements of matrix B ");
      read (b, m, n);
      add matrix (a, b, c, m, n);
      Printf(" Resultant matrix is C \n");
       display (c, m, n);
}
```

STRINGS

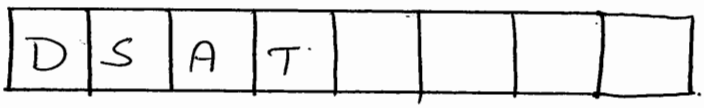
Definition:

A Sequence of Characters is called Strings, The Strings are Stored in Sequence in the memory locations.



Fixed Length:

A String whose Storage requirement is known when String is Created



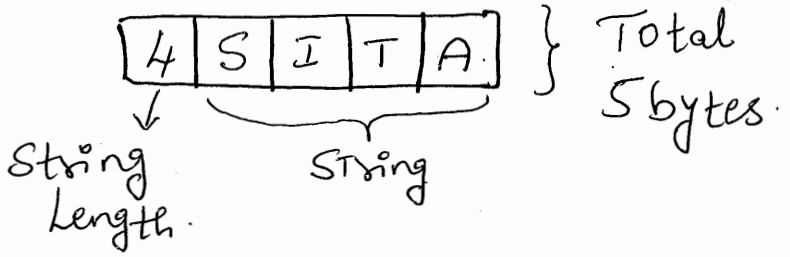
For example DSAT is Stored in 8 bytes fixed format; here Size is fixed to 8 bytes. The remaining four (4) bytes are blank.

Variable Length:

As the name implies, Variable-Length Strings do not have a Pre-defined Length.

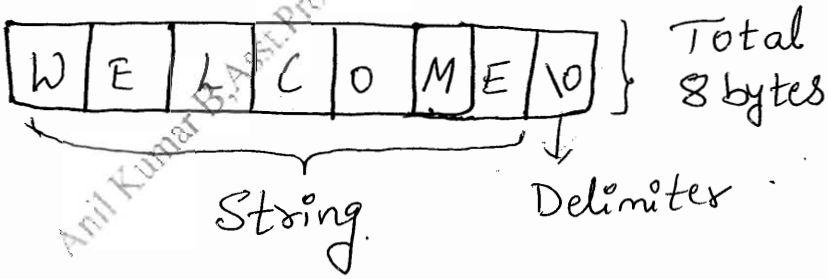
Length Controlled:

A String whose length is Stored as Part of the String itself.



Delimited String:

In a Variable Length the String is ended with a delimited NULL (denoted by \0)



Declaring a Strings:

The String Variable is always declared as an array of Characters.

Syntax:

```
Char String_name[size];
```

The Parameter Size determines total Number of characters in the String name.

Example:

```
Char city[10];
Char name[25];
```

Initializing a Strings:

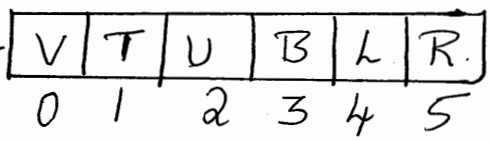
Initialization is a Process of assigning Values to a variable before doing manipulation.

* Initializing Character by character.
* Partial initialization.
* Initialization without Size Specified.
* Initialization with a String.

Initializing Character by character:

```
Char a[6] = {'V','T','U','B','L','R'};
```

Here 6 memory locations are allocated ranging from 0 to 5

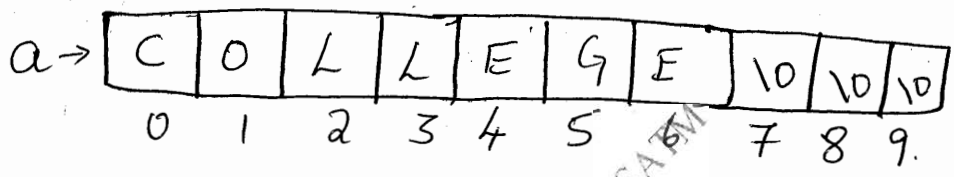


The ASCII Values of the characters will be Stored in the memory Location.

Partial Initialization:

If the number of characters to be initialized is less than the size of array, then Characters Stored Sequentially from Left to Right.

```
Char a[10] = {'C','O','L','L','E','G','E'};
```

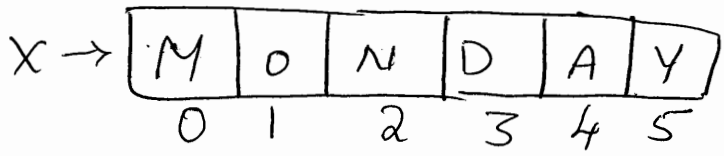


The blank Locations automatically filled with NULL Characters '\0'

Initialize without Size:

```
Char x[] = {'M','O','N','D','A','Y'};
```

For the above declaration the array size will be the total number of initial values.

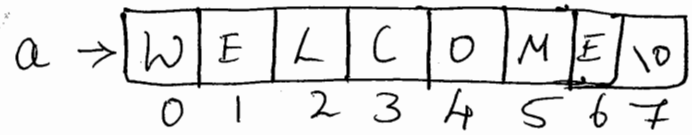


The total size allocated for the variable X will be 6 bytes '\0' is not inserted at end of String.

Array initialization with a String:

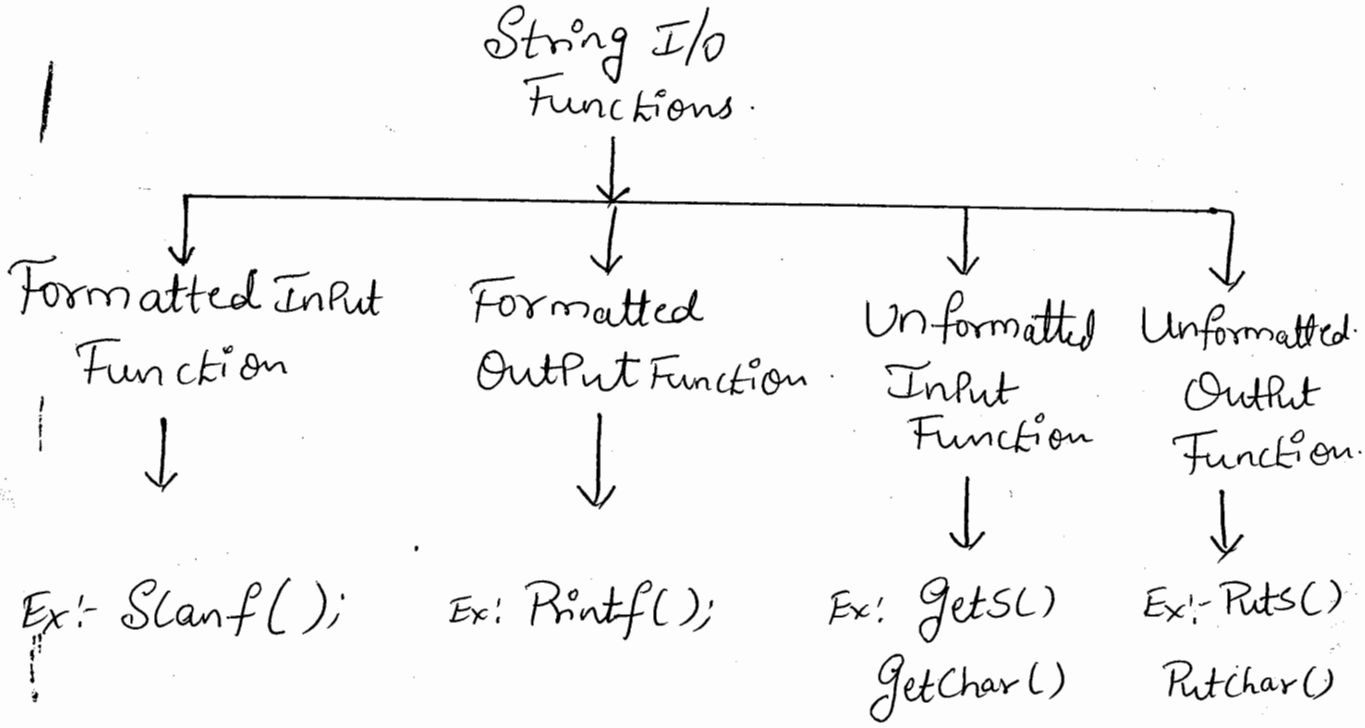
```
Char a[] = "WELCOME";
```

The String length is 7 bytes, but total size of String is 8 bytes; The String always terminated by '\0' NULL Character.



String Input/Output Functions:

The reading the Strings from KeyBoard and Printing the String on to monitor. we have Various input/output functions.



Formatted input function:

The input function Scanf() is used to read the String. The Conversion code used is %s to read a String of Characters.

Example:- Char addr[15];

Scanf("%s", addr);

The "&" operator should not be used to read the String Variables.

Formatted Output function:

The output function Printf() is used to display/Print the String. The Conversion Code/Format Specifier used is %s to Print the String of Characters.

Example:

Char a[10];

Scanf("%s", a);

Printf("%s", a);

Prints the characters is Group of Characters on to the monitor.

Example Program:

```
#include<stdio.h>
#include<string.h>
Void main()
{
    Char word1[10], word2[15];
    Printf("Enter text:\n");
    Scanf("%s %s", word1, word2);
    Printf("\n word1 = %s", word1);
    Printf("\n word2 = %s", word2);
    Getch();
}
```

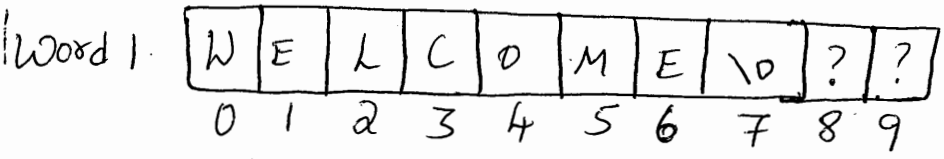

OutPut:

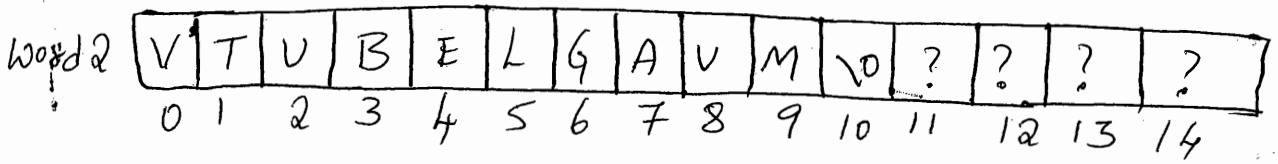
Enter text

WELCOME
VTUBELGAUM

word1 = WELCOME

word2 = VTUBELGAUM.





UnFormatted I/O Functions:

Gets() :- (unformatted inPut function).

To read Sequence of Characters from KeyBoard with Spaces in between and Store them in memory locations. gets() function is used.

Syntax:- gets(str);
↓
String Variable.

Reads String of Characters from Keyboard till User Presses "Enter Key".

Puts() :- (unformatted outPut function)

To display the String of Characters on the outPut Screen Puts() function is used.

Syntax: Puts(str);
↓
String Variable.

The function displays all the Characters Stored in Variable "str" till it encounters NULL Character '\0'.

Example Program:

```
#include <stdio.h>
#include <string.h>
Void main()
{
   Char  Str[25];

   gets (Str);     //Input  A B C ↵ (Press enter)

   Puts(Str);     //output  A BC
}
```


Advantages:-

* Easy to use Unformatted I/O functions.
* A single sequence of Characters can be read easily or displayed easily.

Disadvantages:

* It is not Possible to read/Print any other data except Characters only, ie, not Possible to Print integers, floating numbers.

* To over come the above Problem the formatted I/o functions used Printf and Scanf().

String Manipulation Functions:-

The Various String manipulation functions Supported in C Language are.

* Strlen (Str) – Returns the length of String Str.
* StrCpy (dest, src) – Copies the Source String Src to destination String dest.
* Strncpy (dest, Src, n) – Copies at most n Characters of Source String Src to destination String dest.
* Strcat (Str1, Str2) – APPend String Str2 to Str1
* Strncat (Str1, Str2, n) – APPend first n Characters of String Str2 to Str1
* StrCmp (Str1, Str2) – Compare two Strings Str1 & Str2
* Strstr (Str1, Str2) – Finds the first OCcurence Of String 2 in String Str1.
* Strchr (Str, c) – Scans the String Str for the first OCcurence of Character C.
* Strlwr (Str) – Convert the String Str to Lower case
* Strupr (Str) – Convert the String str to Upper case
* Strrev (Str) – Reverses the String Str
* Strrchr (Str, c) – Finds Last OCcurence of character C in String Str.

Strlen (str) – String Length.

The function returns the length of the String str, it counts all the characters until '\0' and returns the total length.

Syntax:

```
int Strlen (char str[]);
```

Example:

```
#include <stdio.h>
#include <string.h>
Void main()
{
    Char S[] = "LAKSHMANA";
  Printf(" Length = %d", Strlen(S));
}
```


Output:

```
Length = 9.
```

Program: C-Program Using Userdefined function

my_Strlen():-

```
#include <stdio.h>
#include <string.h>
int my_Strlen(char S[])
{
    int i=0;
```

```
    while (s[i] != '\0') i++;
    return i;
}
Void main()
{
    Char s[25];
    int i;
    Printf("Enter the String \n");
    gets(s);
    i = my_Strlen(s);
    Printf("Length = %d", i);
}
```

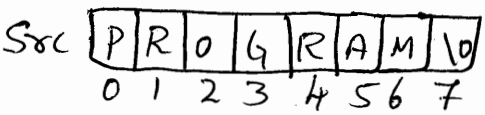

Strcpy (dest, src) – String Copy:-

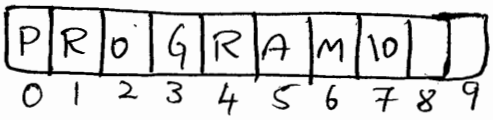
The function Strcpy copies the contents of source string src to destination string dest including '\0', Here the size of destination string should be greater or equal to the source string.

SYNTAX:

```
Strcpy (Char dest[], Char src[]);
```

Example:
```
#include<stdio.h>
#include<string.h>
Void main()
{
  Char src[]="PROGRAM";
  Char dest[10];
  Strcpy(dest,src);
  Printf(" Destination= %s", dest);
}
```





Program:

C Program to implement String Copy using User defined function my_strcpy

```
#include<stdio.h>
Void my_strcpy (Char dest[], Char src[])
{
    int i=0;
    while (src[i] != '\0')
    {
        dest[i] = src[i];
        i++;
    }
dest[i] = '\0';
}
```

```
Void main()
{
    Char src[25], dest[25];
    Printf("Enter String \n");
    Scanf("%.s", src);   OR   gets(src);
    my_Strcpy(dest, src);
    Printf("Dest = %.s", dest);
}
```


Strncpy (dest, src, n) :-

Syntax :

```
Strncpy (Char dest[ ], Char src[ ], int n);
```

n → number of bytes to be copied to destination string

src → source string

dest → destination string.

The function Strncpy copies n characters from source string to destination string.

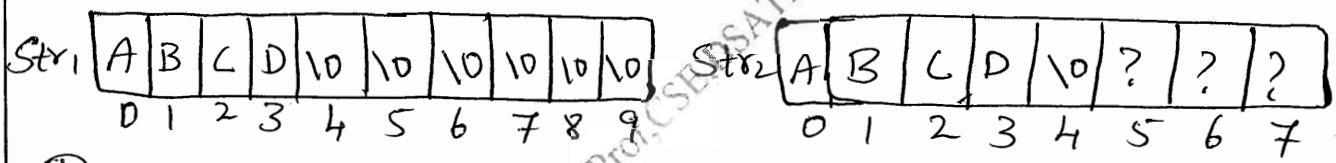
* If source string is less than n, entire string is copied to destination.
* If source string is more than n, then only n characters copied to destination.

Example:

```
Char Str1[10];
Char Str2[8] = "ABCD";
Strncpy (Str1, Str2, sizeof(Str1));
```

Before execution.

After execution Strncpy



Program:

```
#include<stdio.h>
#include<string.h>
Void main()
{
  Char S[15] = " WELCOME TO";
  Char S1[10] = "YES";
  int len;
  len = Strlen (S1);
  Strcpy (&S[6] + len, &S[6]);
  Strncpy (&S[6], S1, len);
Printf(" Result = %s", S);
}
```

Strcat (S_1, S_2) – String Concatenate:-

Syntax:

Strcat (Char S_1[ ], Char S_2[ ]);

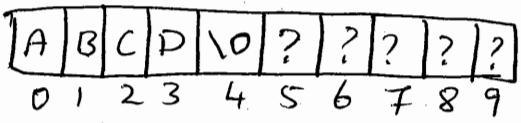
S_1 - first String

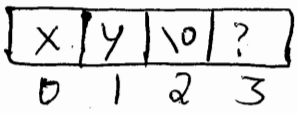
S_2 - Second String.

. The function Strcat copies all the characters of S_2 to the end of string S_1. The delimiter of S_1 (\0) is replaced by first character of S_2. The size of S_1 should be large.

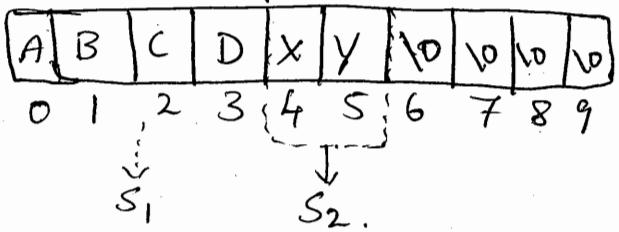
Example:

```
#include <stdio.h>
#include <string.h>
Void main()
{
  Char S1[10]="ABCD";
  Char S2[4]="XY";
 if(Strlen(S1)+Strlen(S2)>sizeof(S1))
   Printf("Error Cannot Concatenate");
   else
    Strcat(S1,S2);
}
```





Concatenated String.



Program:

C Program to Concatenate two Strings using Userdefined function my_strcat:

```
#include <Stdio.h>
Void my_strcat (Char Str1[], Char Str2[])
{
    int i=0, j;
    While (Str1[i] != '\0') i++;
    j=0;
    while (Str2[j] != '\0')
    {
        Str1[i++] = Str2[j++];
    }
    Str1[i++] = '\0';
}
Void main()
{
    Char Str1[20] = " WELCOME ";
    Char Str2[10] = " TO VTU ";
    my_strcat (Str1, Str2);
    Printf (" Concatenated String ");
    Printf (" %.s", Str1);
}
```

Strncat (S_1, S_2, n) :- String Number Concatenate

Syntax:

Strncat(Char S_1[], Char S_2[], int n);

S_1 - first String

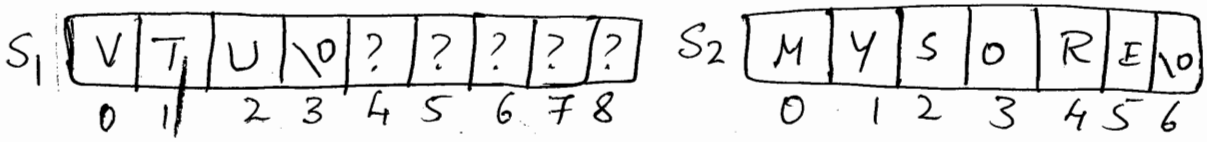
S_2 - Second String

n - number of Characters of String S_2 to be concatenated.

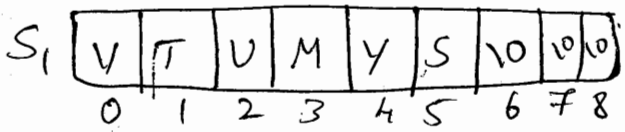
* If the length of String S_2 is less than n then function copies entire String S_2 to end of S_1.

* The delimiter of String S_1 is replaced with String S_2 first Character.

Execution Before Strncat.



After execution Strncat (S_1, S_2, 3);



Here 3 characters are copied from String S_2 to String S_1 at the end of S_1.

StrCmp(S1,S2) - String Compare

SYNTAX:

int StrCmp(Char S1[], Char S2[]);

S_1 - first String

S_2 - Second String.

The function Strcmp Compares two Strings. Comparision Starts from Starting Character of the String.

-> If two Strings are equal returns 0

-> If S_1 is greater than S_2, +ve Value returned.

-> If S_1 is less than S_2, returns Negative Value.

Program:

C - Program to implement String Comparision Using my_strcmp function.

```
#include <Stdio.h>
int my_strcmp(Char S1[], Char S2[])
{
    int i=0;
    while (S1[i]==S2[i])
    {
        if (S1[i]=='\0') break;
        i++;
    }
    return S1[i]-S2[i];
}
```

```
Void main()
{
    Char S1[] = "VTU";
    Char S2[] = "BELGAUM";
    int diff;
    diff = my_strcmp(S1, S2);
    if (diff == 0)
      Printf("%s = %s", S1, S2);
  else if (diff > 0)
      Printf("%s > %s", S1, S2);
    else
      Printf("%s < %s", S1, S2);
}
```

Working of function my_strcmp()

→ Zero if $S_1 = S_2$

→ Positive if $S_1 > S_2$

→ Negative if $S_1 < S_2$

Strrev(Str) - String reverse

SYNTAX:

```
Void Strrev(Char Str[]);
```

-> Str -> InPut String

The function reverses all the characters inthe String Str except '\0'. Original String is Lost.

Example:

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

Void main()
{
   Char Str[]= " WELCOME";

   Strrev (Str);

   Printf (" Reverse = %.S", Str);

}
```


Output:

REVERSE = EMOCLEW

The Strrev() is inbuilt function defined in String.h header file.

Program: C Program to implement String reverse using User defined function.

```
#include <string.h>
#include <stdio.h>
Void myreverse (Char src[], Char dst[])
{
    int i, n;
    n = Strlen (src);
    for (i=0; i<n; i++)
    {
        dst[n-1-i] = src[i];
    }
    dst[n] = '\0';
}
Void main ()
{
    Char S1[] = "VTU", d1[10];
    Char S2[] = "BLR", d2[10];
    myreverse (S1, d1);
    myreverse (S2, d2);
    Printf("Source    Destination\n");
    Printf(" %.s \t %.s \t", S1, d1);
    Printf(" %.s \t %.s \t", S2, d2);
}
```

Source Destination

VTU UTV

BLR RLB

Array of Strings:

Syntax: Char a[row][col];

- a → Name of array
- row → Number of Strings
- col → maximum length of String

Char a[5][20];

-> 5 in the declaration indicate that 5 Student names can be Stored.

-> 20 indicate that each name can have atmost 20 Characters.

Initialization:

```
Char a[5][10]={
                "WELCOME",
                "TO",
                "VTU",
                "BANGALORE",
                "BELGAUM"
              };
```

The elements can be read and accessed through the loop Statement as.

```
for(i=0; i<=4; i++)
 {
   Scanf("%s", a[i]);

 }.
```

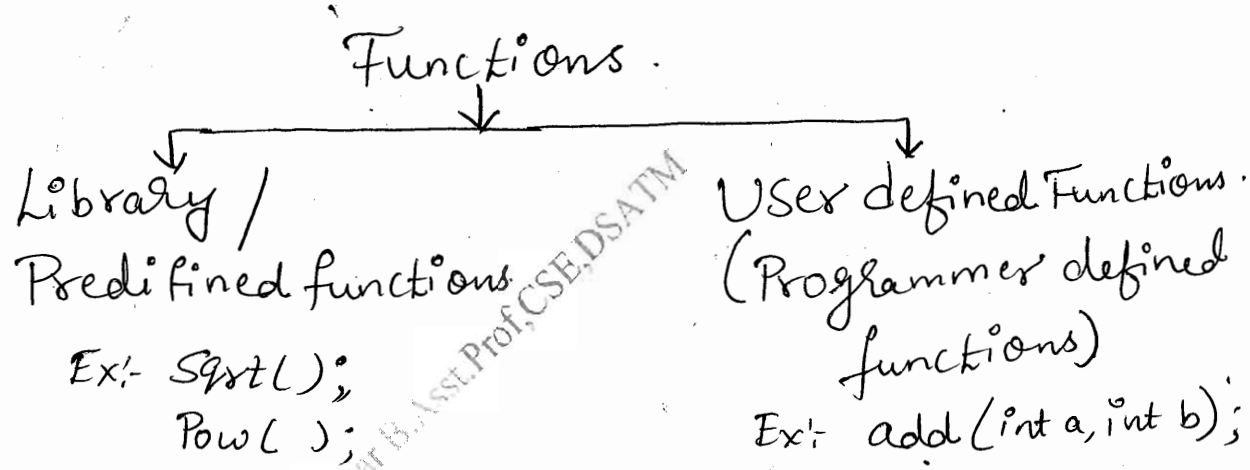
The array of String Values are Printed as

```
for(i=0; i<=4; i++)
{
  Printf("%s", a[i]);
}.
```

FUNCTIONS.

Functions in C.

A large Program can be divided into Small Pieces Called modules, each module also Called a function. The functions are classified into two Ways.



Library Functions:

The Collection of various functions that Performs Standard task and Pre defined tasks, these functions written by the Compilers manufacturers Called library functions.

Examples:

Pow(x,y) – Computes x^y.

Sqrt(x) – Computes Square root of x

Printf() – Used to Print data on output Screen

Scanf() – To read data from keyboard.

User defined Functions:

The functions written by the Programmers to do the specific task called User defined functions. also called Programmer defined functions.

Example:-

```
#include<stdio.h>
Void add()          /* User defined function */
{
   int a, b, Sum;
   Printf(" enter a, b \n");
   Scanf("%d%d", &a, &b);
   Sum = a+b;
   Printf(" %d", Sum);
   return;          /* return control to main */
}
Void main()
{
   add();
   return;          /* return control to OS */
}
```


Note:- i) Execution Starts from main()
ii) Function add() is invoked & control transferred from main() to add().

Elements of User defined functions :-

- Function definition
- Function Call
- Function declaration.

Function definition :

The Program module that is written to achieve a specific task is called function definition.

Syntax:

```
type fname (Parameters)  <- Function Header
{
    Declaration Part;
    Executable Part;
    return Statement;
}                          } Function Body.
```

type -> datatype, int, float, char, void, double, etc
fname -> function name.

Example:

```
double add(double m, double n)
{
    double Sum;
    Sum = m+n;
    return Sum;
}
```

Function Declaration / Prototype:

The Process of functions declaring before they are used (or called) is function. Prototype. It is also called function declaration.

Syntax:

```
type fname(type P1, type P2, ..., type Pn);
```

Example:

```
int  add (int a, int b);
Void Sub (int x, int y);
```

type - data types used int, float, double, Void, etc.
fname - function name
$P_1 P_2 P_3 \ldots P_n$ - list of Parameters Seperated by Comma.

Function Call:

Once the function is defined, it has to be called. to achieve the task. The method of calling function to achieve a specific task called function call.

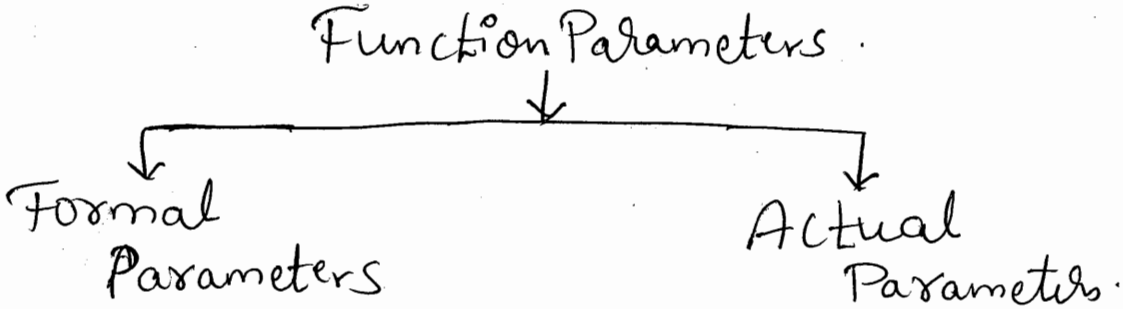
Example:

```
Void main()
{
  int res;
  res = add(10, 20);
}
```

→ Function call.
↳ Arguments

Function Parameters:

The Variables defined in the function and the arguments in the function are Called Parameters.



Formal Parameters:

The Variables defined in the function header of function definition are Called formal Parameters. The formal Parameters are also Called dummy Parameters. The Parameters receives data from actual Parameters.

Actual Parameters:

The Variables that are used when a function is used or invoked are Called actual Parameters. The data transferred to the functions using actual Parameters. The type of data and number of Parameters Should match for formal & actual Parameters.

Example:

```
#include<stdio.h>
double add(double x, double y)
{
    return x+y;
}
Void main()
{
    double a,b, res;
    Printf("enter a,b \n");
    Slanf("%.lf %.lf", &a,&b);
    res = add(a, b);
    return;
}
```

Formal Parameters

Actual Parameters

Categories of Functions:

The functions are categorized based on return value by the function & Parameters accepted.

* Functions with no Parameters & no return Values.
* Functions with no Parameters and with return Values
* Functions with Parameters & no return Values.
* Functions with Parameters & with return Values

Functions with no Parameters & No return Values:

Void functions are with no Parameters. Here there is no data transfer between Calling and Called function, Calling function Cannot Send any Values & Called function Cannot receive any Values.

Example:

```
#include<stdio.h>
Void add();

Void main()
{
   add();
   return;
}
```

```
Void add()
{
   int a,b,c;
   Printf("enter a,b \n");
   Scanf("%d %d",&a,&b);
   C=a+b;
   Printf("Sum=%d",c);
   return;
}
```


Functions with Parameters & no return Values:

The Void functions with Parameters, Here, the data transferred from Calling function to the Called function using Parameters But no transfer of data from Called function to Calling function.

Example:

```
#include <stdio.h>
Void add (int a, int b);
Void main()
{
    int m, n;
    Printf("enter m, n\n");
    Scanf("%d %d", &m, &n);
    add (m, n);
    return;
}
```

```
Void add (int a, int b)
{
    int Sum;
    Sum = a+b;
    Printf("Sum=%d", Sum);
    return;
}
```


Functions with no Parameters & with return Values:-

There is no data transfer from Calling function to Called function, But the data is transferred from Called function to the Calling.

When the Control is transferred to the Called function, the values are read they are added & result is Stored in 'c' & the value of c is returned.

Example :-

```
#include<stdio.h>
int add();
Void main()
{
  int c;
  c = add();
  Printf("Sum=%d", c);
  return;
}
```

```
int add()
{
  int a, b, c;
  Printf("enter a, b \n");
  Scanf("%d%d", &a, &b);
  c = a+b;
  return c;
}
/* returns the
   Value */
```


Functions With Parameters & with return Values :-

Here, the data transfer between the Calling function and the Called function is done. When Parameters are Passed, the Called function receives the Values from the Calling function.

When function returns a Value, the Calling function can receive Value from the Called function.

Example:-

```
#include<stdio.h>
int add(int a, int b);
Void main()
{
  int m,n,c;
  Printf("enter m,n");
  Scanf("%d%d",&m,&n);
  C=add(m,n);
  Printf("Sum=%d",C);
  return;
}
```

```
int add(int a, int b)
{
  int C;
  C=a+b;
  return C;
}
```


Arguments Passing:

The arguments are passed to the functions in two ways.

- Call by Value
- Call by Reference.

Call By Value:-

The Values of actual Parameters are Copied into formal Parameters, here. formal Parameters has only the Copy of actual Parameters. if the Values Changed in the Called function, then the actual Parameters will not be Changed.

Example:

```
#include<stdio.h>          → Formal Parameters
Void Swap(int P, int q)
{
    int temp;
    temp = P;
    P = q;
    q = temp;
}
Void main()
{
    int x, y;
    x=10;  y=25;
    Swap(x, y);  → Actual Parameters
    Printf("x=%d y=%d", x, y);
}
```

{P=25 q=10;}

Output

a=10 b=20.

Call By Reference:

When the function is called, the address Of actual Parameters are sent.

* The formal Parameters Should be declared as Pointers in Called function with Same data type as actual Parameters.

* The address of actual Parameters are Copied into formal Parameters. Using these addresses Values of actual Parameters also Can be Changed.

Example:-

```
#include<stdio.h>                    → Formal Parameters
Void swap(int *P, int *q)
{  int temp;
    temp = *P;
    *P = *q;
    *q = temp;
}
Void main()
{
    int a, b;
    a=10; b=20;
    Swap (&a, &b); → Actual Palametils

  Printf("a=%d b=%d", a, b);
}
```

Output

a=20 b=10.

Location of Functions:

The functions can be defined in various places in same file or different files. The function locations can be done in three different regions.

- Functions immediately after #include
- Functions after main() function.
- Functions in seperate files.

Functions immediately after #include & #define:

→ The entire function definition consisting of function header & function body is placed at begining of file immediately after #include.

→ The function Prototype can be ommited if function is defined before main().

```
#include< >
#define
// Function Prototypes.
// Function Definitions

// main() function.
```

Example:

```
#include <stdio.h>
Void Square (int x);   /* Function Prototype */
Void Square (int x)    /* definition */
{
  Printf("Square = %d", (x*x));
}
Void main()            /* main function */
{
  int a;
  Printf("enter a");
  Scanf("%d", &a);
  Square(a);
}
```


Function After main():-

→ The function Prototype should be written immediately after #includes.

→ The main() function will be written followed by all Sub function definitions.

```
#includes.
// function Prototypes
// main function
// function definitions
```

Example :-

```
#include<stdio.h>
Void Square(int P);  /* Function Prototype */
Void main()          /* main function */.
{
   int x;
   Printf("enter x");
   Scanf("%d",&x);
   Square(x);
}
Void Square(int P)   /* Function definition */
{
   Printf("%d", P*P);
}
```


Functions in Seperate Files:

→ Write all function Prototypes immediately next to #includes &#defines
→ Write main() function.
→ define all user defined functions in seperate files.

File1.c

```
#includes
//functionPrototypes.
//main() function.
```

File2.C

```
//functiondefinitions.
```

Example:

file1.c

```
#include<stdio.h>
Void Sub(int x, int y); /* Function Prototype */
Void main()  /* main function */
{
    int a, b;
    Printf("enter a, b");
    Scanf("%d %d", &a, &b);
    Sub(a, b);
}
```

file2.c

```
Void Sub(int x, int y)  /* function definition */
{
    int c;
    c = x - y;
    Printf("%d", c);
}
```


Note:- Here, the main() Program is in one file and other functions are defined in another file; they are Compiled Seperately, This way Library functions are used.

RECURSSION :-

A Recurssion is a method of solving the Problem where the Solution to a Problem depends on Solutions to Smaller instances of the Same Problem. The recursive function is a function that calls itself during execution.

Two ways of Recurssion.

- Direct recursion
- Indirect recursion.

Direct recursion:

A recursive function that invokes itself is said to have direct recursion.

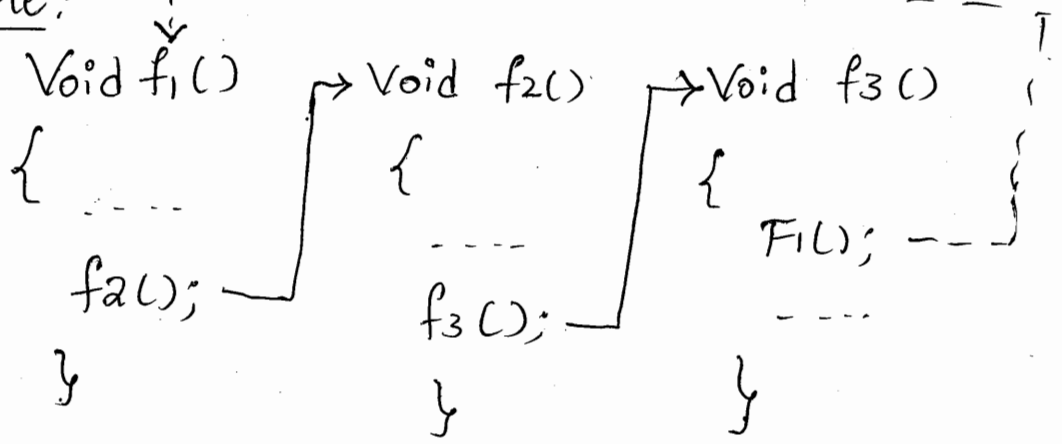
Example:

```
int fact(int n)
{ if (n==0) return 1;
  return n*fact(n-1);
}
```

Indirect recursion:

A function which Contains Call to another function which in turn Calls another function and so on & Calls again first function Called indirect recursion.

Example:-



Programming Examples for Recursion:

(i) Compute Factorial of n.

$$n! = \begin{cases} n! = 1 & \text{if } n = 0 \\ n! = n * (n-1)! & \text{otherwise} \end{cases}$$

For the above definition function for factorial is written as.

$$F(n) = \begin{cases} 1 & \text{if } n = 0 \\ n * F(n-1) & \text{otherwise.} \end{cases}$$

Function:

```
int fact (int n)
{
    if (n == 0) return 1;
    return n*fact(n-1);     /* when n > 0 */
}
```

Program 1:

C - Program to Compute Binomial Co-efficient

$$nCr = n! / ((n-r)! * r!)$$

```
#include <stdio.h>
int fact (int n)
{
    if (n==0) return 1;
    return n*fact(n-1);
}
Void main()
{
    int n,r;
    float result;
    Printf(" enter n");
    Scanf("%d", &n);
    Printf(" enter r ");
    Scanf ("%d",&r);
   result = fact (n) / (fact (n-r) * fact(r));
   Printf(" %d C %d = %d ", n, r, result);
}
```

Output:

```
Enter n : 6
Enter r : 3
6C3 = 20
```

Program-2:

C-Program to find Sum of array elements using Recursion

```
#include<stdio.h>
 float Fun(float a[], int n)
 {  if (n==-1) return 0;
    return Fun(a,n-1)+a[n];
  }
 Void main()
{
   int n,i
   float a[15], Sum;
   Printf("Enter number of elements");
   Scanf("%d", &n);
   Printf("Enter elements");
   for(i=0; i<n; i++)
   Scanf("%d", &a[i]);
   Sum = Fun(a, n-1);
   Printf("Result=%f", Sum);
}
```

Program-3:

C-Program to Compute factorial of number using recursion.

```
#include<stdio.h>
int fact(int n)
{
    if(n==0) return 1;
    return n*fact(n-1);
}
Void main()
{
    int n, result;
    Printf("enter n");
    Scanf("%d", &n);
    result = fact(n);
    Printf("%d", result);
}
```


Limitations of Recursion; (Advantages)

- → Recursive definition can be easily translated into recursive function.
- → Clearer & simpler versions of recursive functions can be created.
- → Many functions are easier to implement recursively & efficient.

Disadvantages of Recursion:

→ Since function is called repeatedly, it takes more time to execute.

→ Execution is very slow compared to other iterative counterparts

→ Consumes lot of memory when function is called.

Program - 4

C-Program to Evaluate Sum of natural numbers.

```
#include <stdio.h>
int Fun(int n)
{
    if (n == -1) return 0;
    return n + Fun(n-1);
}
Void main()
{
    int n, result;
    Printf("enter n");
    Scanf("%d", &n);
    result = Fun(n);
    Printf("Result = %d", result);
}
```