C. Program

Unit 1st

Array

Array :-

The Variable that we have known till Now and on Capable of Storing only one value at a time Consider a situation there we want to ~~state~~ Store and Display the age of Hundred Employes. In and Display the age of Hundred Employes. In this we have to the following

1. Declare two Different variable to store the age of the Employes.
2. Assign a value of Each Variable.
3. Display the value of Each Variable.

It would Be Very Difficult to Handle So Many Variable in the program and the program Become Very lengthy Hence the Concept of array is useful in these type of Situation where we can group similar type of Data Items.

Arrays :-

An Array is a collection of similar type of Data Item and Each Data Items is called an Elements of the Array.

The Data type of Elements May Be any valid Data type like int, char, float

The Element of Array share the same Variable but Each element has a Different index no. known as Sub-Script for the above problem we can take Array Variable age [100] of type int. ~~take Array variable~~ The Size of their array variable is 100. So it is Capable of Storing 100 integer value. The individual element of this Array are Age [0] age [1] age [2] age [3] --- age [99].

In C the Sub Subject Start from 0 so age[0] is 1st element and age[1] is 2nd Element and so an. Array can Be Single Determine the Dimension of the Array. 1 D Array has one Subscript 2D array has 2 Subscript and so on. The one Dimensional arrays are known as Vector and 2D Dimensional array are known as Matrices.

One-Dimensional Array :-

The Syntax for Declaration of one-dimensional array

data type arrayName [size];

Here arrayName Denotes the name of the Array and it can Be only Valid C Identifire data type is the type of Element of Array Here are some Examples of Array Declaration.

```
int age[10];
Float sal[15];
Char Name[18];
```

When the Array is Declared the compiler allocate Space Memory to hold the Elements of the Array So the Compiler should known the Size of array at the Compile time.

1.) Excessing One-dimensional Array :-

The Element of 1-d array can Be excess by Specifying the array ~~Array~~ Name followed By Subscript in Squre Bracket.

In C, the Array Subscript Start from zero, Hence if an Array of Size 5 then the valid Subscript will Be 0 to 4. The last valid Subscript is 1 is less than the Size of the Array this last Valid Subscript is the Sometime known as the upper Bound of the Array and zero is known as the lower Bound of the Array.

```
int age[5];
```

age is an array having Size 5 then the Individual Element of the Array are age[0], age[1], age[2] age[3], age[4] upper Bound lower Band

Processing of 1-d array :-

For processing array we generally use a for loop and the loop variable is used at the place of Subscript The Inital Value of loop variable is 0. Since array Subscript start from zero. the variable is Increases By one each time so we can access and process the Next Element in the Array. The total No. of passes in the loop will Be Equal to the No. of Element in the Array In Each pass we will process in one Element.

for eg - a[10] is an Array of Integer. type

```
int a[10];
for(i=0; i<10; i++)
scanf("%d", &a[i])   // Reading in Array
for(i=0; i<10; i++)
printf("%d", a[i]);  // Writting in Array
```

Q. Write a Program to Input Value into an array and Display them where as the No. of Elements in the array is Entered By User.

```
#Include<stdio.h>
Void Main()
{
  Int n;
printf("Enter the Value");
Scanf("%d",&n);
    Int a[50];
For(i=0;i<n;i++)
printf("%d",&a[i]);
}
```

(IInd method)

```
#include<stdio.h>
Void Main()
{
Int n;c=0,d=0
printf("Enter Any No");
Scanf("%d",&n);
   int a[10];
for(int i=0;i<n;i++){
 Scanf("%d",&a[i]);}
for(int i=0;i<n;i++)
  if(a[i]%2==0)
      c++;
  else
      d++;
printf("%d"is Eventerm,c);
```

```
printf("%d is
Odd term,d);
}
```

Write a Program to Add the Elements of the Array

```
#Include<stdio.h>
Void Main()
{
  Int a, Sum=0;
printf("Enter Any Value")
Scanf("%d",&a);
   Int a[10]
For(i=0;i<a,i++)
Scanf("%d",&a[i]);
For(i=0;i<=a;i++)
   Sum=Sum+a[i]}
printf("%d",&Sum);
}
```

Write a Program to Input an Array and Count the No. of Even and Odd Term in that Array.

(Ist Method)

```
# Include<Stdio.h>
Void Main()
{
printf("Enter Any Value");
  Int n;Count=0 Count1=0;
printf("Enter Any No");
Scanf("%d",&n);
  int a[10];
for(int i=0;i<n;i++)
 Scanf("%d",&a[i]);
for(int i=0;i<n;i++)
  if(a[i]%2==0)
      count++;
else if(a[i]%2!=0)
```

```
    Count++;
printf("%d"=Count);
printf("%d"=Count")
}
```

Searching in One Dimantional Array

One of the Basic operation to be performed on an Array is Searching. Searching an Array means to find a particular element in the Array.

The Search can be used to Return the position of the element or check if it Exist in the Array or Not.

Basically Searching in an Array is of two types

1. Linear Search
2. Binary search

Linear Search

The simplest Search to be Done on an Array is the Linear Search This search start's from one end of the Array keeps Re Iterating until the Element is found or There are No More Elements left.

```
#include <stdio.h>
Void Main
{
int a[100];
int n,i,t;
printf("Enter No. Element in Array")
Scanf("%d", &n);
printf("Enter the Array element");
for(i=0; i<n; i++)
Scanf("%d", &a[i]);
printf("Element the element to search");
Scanf("%d", &t);
for(i=0; i<n; i++)
{
if(a[i]==t)
{
printf("Element is found at position %d", i+1);
break;
}
}
if(i==n)
printf("Element %d is not found in the array.", t);
}
```

Binary Searching :-

The Linear Search apporch has one Disadvantage Finding Elements in a Large Array will be time consuming As the Array grows the time will Increase Linearly a Binary Search can be used as a Solution to this problem. The principle of Binary search is How we find a page in a Book. We open the Book at a Random page in the Middle and Based on that page we Narrow our Search too the left or Right of the Book. Indeed This is only possible if the page No. are in Order. The Search starts By accessing the Middle of the Array. If the Element is less then the Middle Element it Starts it Search from this Element to the left of the Array. If the Element is More it Starts its Search from this Element to the Right of the Array. This process is Repeated Unless the Middle Element is Equal to the No we are Searching.

```
# Include <stdio.h>
Void Main ()
{
int a[100];
int n, m, i, s, f, l;
printf("Enter the No. Element in Array");
Scanf("%d", &n);
printf("Enter Array Element in Shorted Order")
for(i=0; i<n; i++)
Scanf("%d", &a[i]);
printf("Enter Element Search");
Scanf("%d" &s)
f=0; l=n-1; m=(f+l)/2;
```

```
while (f <= l)
{
if (a[m] < s)
{
f = m+1;
}
else if (a[m] == s)
{
printf ("Element is found at location %d", m+1);
Break;
}
else (l = m-1);
m = (f+l)/2;
}
if (f > l)
printf ("Element is not found");
}
```

Sourting

The process of sourting can be explained as a technique of rearranging the elements in any particular order which can be set ready for further processing by the program logic. In c programming language there are multiple sourting algorithms available. The various types of sourting methods possible in the C language are Bobule sourt, Selection sourt, Quick sourt, Marge sourt, Heap sourt, Insertion sourt

Bouble Sourt

It is the simplest sourting technique which is also

called an Exchange Sourt.

procedure of Bouble Sourt -

1. Compare the First Element with the Remaining Element in the Array and Exchange them if they are Not in Order.
2. Repeat the same for other Element in the list until All the Elements are Sourted. for eg -

for eg - ~~30, 50, 40, 10, 20~~ 30, 50, 40, 10, 20

pass-I 30 > 50 N.E (No Exchange)
30 > 40 N.E
80 > 10 E
10 > 20 E
10, 50, 40, 30, 20

pass II 50 > 40 E (10, 40, 50, 30, 20)
40 > 30 E (10, 30, 50, 40, 20)
80 > 20 E
10, 20, 50, 40, 30

pass III 50 > 40 E (10, 20, 40, 50, 30)
40 > 30
10, 20, 30, 50, 40

pass IV 50 > 40 C
(10, 20, 30, 40, 50

classmate

Date
Page

Q. 1, 7, 8, 6, 2 With the Help of Bubble Sort Solve this

pass I 1 > 7, N.E (1, 7, 8, 6, 2)
1 > 8 (1, 7, 8, 6, 2)
1 > 6 (1, 7, 8, 6, 2)
1 > 2 (1, 7, 8, 6, 2)

pass II 7 > 8 N.E
7 > 6 E (1, 6, 8, 7, 2)
6 > 2 E (1, 2, 8, 7, 6)

pass III 8 > 7 E (1, 2, 7, 8, 6)
7 > 6 E (1, 2, 6, 8, 7)

pass IV 8 > 7 E (1, 2, 6, 7, 8, 6)

Program for Bubble Sort

```
# include <stdio.h>
void Main ()
{
int a[100]
printf ("Enter the No. of Element in Array");
Scanf ("%d", &n);
print ("Enter the Array Element");
for (i=0; i<n, i++)
Scanf ("%d", a[i]);
for (i=0; i<n; i++) {
{
for (j=i+1; j<n; j++)
{
if ( a[i] > a[j])
{
t= a[i]; a[i]= a[j]; a[j] = t;
}
}}
printf ("The Sorted Array is");
For (i=0; i<n; i++)
Printf ("%d /n", a[i]); }
```

Two Dimantional Array

Declaration and Associating Individual Element of two D Array The Syntex of Declration of 2D Array is Similar to the Decleration of 1D Array But Hear We have two Subscripts

Data Type ArrayName [row size] [col. size]

Here Row size Specify the No. of Rows & Coloum size Specify the No of coloums in Array Hence the total No of element in the Array is Row size Multiplyed By Coloum size for eg.

int a [3] [4]

a [0] [0], a [0] [1], [0] [2], a [0] [3]
:
a [3] [0], - - - - a [2] [3]

Processing Two D Array Elements

For processing Two D Array We use Two four loops, the outer four loop correspond to Row & Inner For loop Correspond to Coloums. In the above Example we are taking the elements from user as follows

```
For (i=0, i<3, i++)
{
for (j=0, j<4, j++)
{
```

```
        scanf ("%d", &a[i][j]);
      }
    }
```

For Displaying the value of the above Array the following syntax is used -

```
for (i=0; i<3; i++)
{
  for (i=0; i<4; i++)
  printf ("%d", a[i][j]);
}
```

Q. Write a Program to Input & Display a Matrix where the No. Rows & Columns Are Entered User.

```
#Include <Stdio.h>
  Void Main ( )
  {
    int a[10][10], i, j, m, n;
  printf ("Enter the Number of Rows");
  Scanf ("%d", &m);
  printf ("Enter the no. of Columns");
  Scanf ("%d", &n);
  printf ("Enter the Array Element");
      for (i=0; i<m; i++)
    {
      for (j=0; j<n; j++);
    scanf ("%d", &a[i][j]);
      }
    }
    printf ("The Matrix is");
```

Q. Write a Program for the Addition of Two Matrices where the No. of Rows & Column of Each matrix are Entered By User.

```
#Include <stdio.h>
  Void main ()
  {
int a[10][10], b[10][10], c[10][10], i, j, m, n;
  printf ("Enter the no. of Row & Column of matrix");
  Scanf ("%d %d", &m, &n);
printf ("Enter the Element of Matrix A\n");
for (i=0; i<m; i++) {
    {
      for (j=0; j<n; j++)
  {
    Scanf ("%d", &a[i][j]); }
  }
printf ("Enter the Element of matrix");
 for (i=0; i<m; i++)
  { for (j=0; j<n; j++) {
  Scanf ("%d", &b[i][j]);
}
}

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

Q Write a program to print to Transpose of matrix

```
#Include<stdio.h>
Void Main()
{
    Int a[10][10], i,j, m, n;
printf ("Enter the no. of Rows & Colcumn");
Scanf ("%d %d", &n, &n);
for (i=0; i<n; i++)
{
for(j=0; j<n; j++)
Scanf ("%d", &a[i][j]);
}
for (j=0; j<m; j++)
{
For(i=0;i<m;i++)
printf ("%d", a[i][j]);
}
```

Q Write a program for Multiplication of Matrix.

```
#Include<stdio.h>
    void main()
{
Int a[10][10] i,j, m, n
    Int a[10][10] b[10][10] c[10][10], i,
                j, r1, r2, c1, c2;
printf ("Enter the No. of Rows for first Matrix")
Scanf ("%d", &r1);

printf ("Enter the No. of Colcum for first Matrix
Scanf ("%d", &c1);

printf ("Enter the No. of Rows for Second Matrix")
Scanf ("%d", &r2);

printf (Enter the No. of Colcums for Second
                Matrix)
Scanf ("%d", &c2);

If (c1==r2)
{
printf ("Enter the Elements for first Matrix")
For (i=0; i<r1; i++)
{
    For (j=0; j<c1; j++)
{
Scanf ("%d", &a[i][j]);
For (i=0; i<r2; i++)
{
```

Unit-2

Pointer's

Pointer Variable

A pointer is a variable that store memory Address like all other variable. It also as a Name, has to be Declare and occupy some space in memory. It is called pointer because it points to a particular location in Memory By storing the Address of that location.

Declaration of pointer variables

Like other variables pointer variables should also be Declare Before being used. The General Syntax of Declration of pointer variable is.

Datatype *pname ;

Here P Name is the Name of pointer variable which Should Be a valid C Identifier. The ~~Astrid~~ * preceding this Name informs the Compiler that the variable is Declare as a pointer. where Data Type is known as the Base Type of the pointer.

Let us take some pointer Declaration

```
int *ptr;
float *fptr;
char *cptr, ch1, ch2;
```

Here ptr is a pointer that should point to variable of type int similarly fptr, cptr. Should point to variable of float and Character Type. whereas ch1 and ch2 are Normal character Type variables.

Operators use with pointers

There are two Basic operators used with pointers

1. Address operator

C provide an address operator which Returns Address of the variable for eg.

1. It is also known as Reference operator
2. In Direction operator (*)
3. It is used to get the value stored in an Address. for eg

Pointer Arithmatic

All type of Arithmatic operations are Not possible with pointer the only valid operations that can Be performed on pointers are as follows -

1. Addition of an integer to a pointer and Increemment operations.
2. Subtraction of a integer from an pointer and Decreemment operation.
3. Subtraction of a pointer from Another pointer of same time

for eg. If we have an Integer value pi which contains Address 1000 then on Incrementing we get 1002 instead of 1001 This is Because int Data Type take 2 Bytes. similarly on Decrementing pi we will get 998 instead of 999. The Expression pi Represent the Address

Let us See pointer Arithmatic for int, float, char pointers

```
int a=5   * ptr = &a;
Float b=24 * iptr = &b;
char c='x' * cptr = &c;
```

Suppose the Address of the Variable A, B, C are 1000, 4000, 5000 So Initely Value of iptr, fptr, cptr will Be 1000, 4000, 5000

WAP to print the address of a Variable Along With its Value -

```
#include <stdio.h>
    void main()
{
Char a;
int x;
float p, q;
a='A';
x=7;
p=22;
q=47;
printf("%c is stored at address %u\n", a, &a);
printf("%d is stored at address %u\n", x, &x);
printf("%f is stored at address %u\n", p, &p);
printf("%f is stored at address %u\n", q, &q);
}
```

→ WAP to Illustrate the use of Indirection operation (*) to Access the Value of pointed to By a pointer.

```
#include <stdio.h>
    void main()
{
int x, y;
int * ptr;
x=10;
ptr = &x;
y = *ptr;
printf("Value of x is %d\n", x);
printf("%d is stored at address %u\n", x, &x);
printf("%d is stored at address %u\n" * &x, &x);
printf("%d is stored at address %u\n", *ptr, ptr);
printf("%d is stored at address %u\n", ptr, &ptr);
printf("%d is stored at address %u\n", y, &y);
    *ptr=25;
printf("\n new x = %d\n", x);
}
```

→ WAP to Illustrate the use of pointer in Arithmatic operation.

```
#include<stdio.h>
    void main
{
int a, b, *p1, *p2, x, y, z
    a=12;
    b=4;
    p1=&a;
    p2=&b;
  x = *p1 * p2 - 6
  y = 3* - *p2 / *p1 + 10;
```

```
printf("Address of a = %u\n", p1);
printf("Address of b = %u\n", p2);
printf("Address of a = %d, b = %d\n", a, b);
printf("x = %d, y = %d\n", x, y);
        *p2 = *p2 + 3;
        p1 = *p2 - 5;
        z = *p1 * *p2 - 6;
printf("\n a = %d, b = %d", a, b);
printf("z = %d\n", z);
}
```

→ Pointers & Array's

When an Array is Declare the Compiler Allocates a Base Address and Sufficant Amount of Memory to Contain all the Elements of the Array in Continous Memory location. The Base Address is the location of the first Element of the Array. The Compiler Also Defines the Array Name as the Constant pointer to the first Element for eg. `int x[5] = {1, 2, 3, 4, 5};`

Suppose the Base Address of x is 1000 then the five Elements will Be stored as follows.

Elements	x[0]	x[1]	x[2]	x[3]	x[4]
Value	1	2	3	4	5
Address	1000	1002	1004	1006	1008

The Name x is Defined as a constant pointer pointing to the first Element x[0] and therfore the Value of x is 1000, the location where x[0] is Stored i.e $x = \&x[0] = 1000$;

If we Declare p as an Integer pointer then `p = x` or `p = &x[0];`

Now we Can Access Every Value of x using p++ to Move from one Element to Another. The Relationship B/w p & x is Shown as → `p+1 = &x[1]; p+4 = &x[4];`

Hence Address of x[3] = Base Address + 3 * Scale factor of Int.

When Handling Array's Instead of using Array Indexing we can use pointers to Access Array Elements. So `*p+3` gives the Value of x 3. The pointer Accessing Method is Much faster than Array Indexing.

Q) WAP using pointer to Complete the Sum of all Elements Stored in an Array.

```
#include <stdio.h>
void main()
{
    int *p, sum = 0, i;
    int x[5] = {5, 9, 6, 3, 7};
    i = 0;
    p = x;
printf("Element Value address\n");
    while (i < 5)
printf("x[%d] %d %u\n", i, *p, p);
        sum = sum + *p;
        i++;
        p++;
}
printf("Sum = %d\n", sum);
printf("&x[0] = %u\n", &x[0]);
```

printf ("p = %u \n", p);
}

Array of pointers

In C language a pointer Array is a Homogeneous collection of indexed pointer variable that are Referenced to a memory location. It is generally used in C programming when we want to point at multiple memory location of a similar Data Type in our C program. We can access the Data By dereferencing the pointer pointing to it. The Syntax of this is

pointer_type * array_name [array_Size]

Here pointer Type is the Type of Data Type the pointer is pointing to Array of pointer and Array Size so the Size of Array.

Foreg- # Include <stdio.h>
void main()
{
int ~~void~~ var1 = 10, var2 = 20, var3 = 30;
int * ptr_arr [3] = {&var1, &var2, &var3};
for (int i=0; i<3; i++)
{
printf ("Value of var %d : %d \t address : %u \n", i++, * ptr_arr [i], ptr = arr[i]);
}
}

Pointer at Function argument

When an Array is passed to a function as an argument only the address of the first element of the array is passed but not the actual values of the array elements. If x is an array, when we call sort(x), the address of x[0] is passed to the uses this address for manipulating the array elements.

When we pass addresses to a function the parameter recieving the addresses should be pointers. The process of calling a function using pointers to pass the address of variables is known as call by reference where as the process of passing the actual variable is known as call by value.

Functions Turning pointer

Since pointer are a data type in C we can also force a function to return a pointer to the call function.

```
#include <stdio.h>
int * large (int *, int *);
void main()
{
int a = 10;
int b = 20;
int *p;
p = large (&a, &b);
printf ("%d", *p);
}
int * large (int *x, int *y)
```

```
{
  if (*x > *y)
    return x)
else
return y;
}
```

✱ Dynamic Memory allocation

C language requires the no. of element in an array to be specify at compile time but we may not be able to do so always. our initial judgement of size, if it is wrong, may cause failure of the program or wastage of memory space.

Many languages permit a programmer to specify an array size at run time.
Such languages have the ability to calculate and assign Dusing Execution the memory. The process of allocating memory at Run time is known as Dynamic memory allocation.

In C there are four library function known as memory managment functions that can Be used for allocating and freeing memory During program execution these function are as follows -

→ malloc ()
→ calloc ()
→ free ()
→ realloc ()

→ malloc ()
Allocate Requested size of Bytes and Return a pointer to the first Byte of the Allocated space.

→ calloc ()
Calloc () space for an Array of Element initalize them to zero and then Return a pointer to the memory.

→ Free ()

✱ Malloc function [single Block]
A Block of memory may Be allocated using the function malloc.
The Malloc function a Block of memory of and Returns a pointer of Type void this means that we can Assign it to any type of pointer. The general Syntax of malloc function

ptr = (cast-type*) Malloc (Byte Size)

For eg- x = (int*) malloc (100* size of int)

On successful execution of this statment. A memory Space Equivalent to 100 times the size of int Byte is Reserved and the Address of the first

Byte to the memory allocated is assigned to the pointer x of type int.

Note that the storage allocated Dynamic has No Name and Therefore its content can Be Accessed only Through a pointer.

```
#include<stdio.h>
void main()
{
int *ptr;
int n, i;
printf("for Number of Elements");
Scanf("%d", &n);
ptr = (int*) malloc (n*Size of(int));
    If (ptr==null)
printf("Necessary Not Allocated \n");
    exit(0);
}
else
{
printf("memory Successfully allocated using
    malloc \n");
for(i=0; i<n; i++)
ptr[i] = i++;
printf("The Element of the Array are:");
    for(i=0; i<n; i++)
        printf("%d", ptr[int]);
}
}
```

Calloc()

Calloc or Contignous allocation method in C is used to Dynamically Allocate the specified type. It is very much similar to malloc function But has two Different points.

i) It Initialize each Block with a default Value 0.

It has two parameter or Agrement as compared to malloc() function. The general syntax of calloc() function is

ptr (cast Type*) calloc (ni element-Size);

Free method of function

Free Method in C language is called to dynamically to deallocate the memory. The Memory allocated using function Method() and Calloc() is Not Dellocated on their own Hence the free method is used whenever the Dynamic Memory allocation take place. It Make to Reduce Wastage of Memory of freeing it.

The General Syntax of free function is; free(ptr);

```
#include<stdio.h>
    void main()
    {
int *ptr, *ptr1;
  int n, i;
printf("Enter no of Elements:");
    Scanf("%d", &n);
ptr=(int*) malloc (n*Size of(int));
```

```
ptr = (int*) calloc (n, size of (int));
  if (ptr = malloc || ptr != null)
  {
printf ("Memory Not allocated");
      exit (0);
  }
  else
{
printf ("Memory allocated Successfully using
      Malloc \n");
free (ptr);
  printf ("Malloc Memory Successfully freed
          \n");
printf ("Memory Successfully allocated
      using Calloc \n");
    free (ptr1);
printf ("Calloc memory Successfully freed
          \n");
  }
}
```

Realloc () Method

This method in C is used to dynamically changed the memory allocation of a previously allocated memory. Reallocation of Memory maintains the already present value and New Blocks will be initialized with the Default garbage value. The General Syntax includes

ptr = Realloc (ptr, new size);

Program -

```
#include <stdio.h>
#include <stdlib.h>
void main ()
{
  int *ptr;
  int n, i;
  n = 5;
printf ("Number of Element %d \n", n);
ptr = (int*) calloc (n, size of (int));
  if (ptr == NULL)
  {
printf ("Memory Not allocated \n");
    exit (0)
  }
  else
{
printf ("Memory Successfully allocated using Calloc
          \n");
ptr (i=0; i<n; i++)
  ptr [i] = i+1;
printf ("The Element of the Array are");
  for (i=0; i<n; i++)
printf ("%d", ptr [i]);
      n=10;
  ptr = realloc (ptr n * size of (int));
printf ("Memory Successfully Reallocated using
      Realloc \n");
  for (i=5; i<n; i++)
  ptr [i] = i+1;
printf ("The Element of the Array are \n");
    for (i=0; i<n; i++)
printf ("%d", ptr [i]);
      free (ptr);
}
```

Unit III cont

String

A string is a sequence of character that is located as a single Data item. Any group of characters defined between double quotation mark is a string contant.

for eg - "Good morning"

If we want to include a double quote in the string to be printed then we may use it with a '\'
for eg printf ("\" Good morning \"")

Character string are often used to build meaning full and , usable program The common operation performed on character string are ;

- Reading & writing string
- Combining string together
- Copying one string to another
- Comparing string for equality
- Extracting a portion of a string.

Declaring and initialing string variable
C Does not support string as a Data Type. However it allows us to represent string as character array. Therfore a string variable in any valid c variable name and is always declared as an Array of characters The general syntax of declaration of string variable is

Char Array Name [size]

for eg Char city [10] = "Grb"
Char city [10] = {'G', 'r', 'b', '\0'}

When the compiler assign a characters string to a character Array it automatically supplies a Null character at the end of the string.
Therfore the size should be equal to the maximum Number of character in the string +1 (plus one).

In the above example, the string "Grb" is stored in the character array 'city' But in the second Example, the individual elements are stored in character array, Hence we have to assign the last character as Null character.

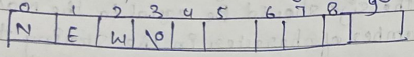
Reading Strings from Terminal
- Using Scanf function
The familiar input function Scanf can be used with %s format specification to Read in a string of character.

for eg- char a[10];
Scanf ("%s", a);

The problem with the scanf function is that it terminates its input on the first white space it finds. A white space includes Blanks, tabs form feeds and New lines. Therefore if the following line of text is typed at the Terminal NEW YORK

then only the string NEW will be read into the array 0. Since the blank space after the word NEW will terminate the reading of string.

Note that unlike previous scanf calls in the case of character arrays. The & (impersand) not required before the variable names. The array a is created in the memory as shown in given figure.



We can also specify the field width using the form % ws in scanf statment for reading a specified number of character from the input string char a [10];

for eg = scanf ("%.ws", a);

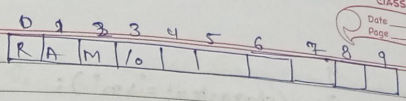
Here two things may happen.

i. The width w is equal or greater then the number of character typed in. The entire string will be stored in the string variable.

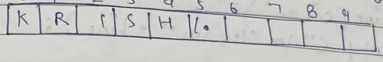
ii. The width(w) is less than the number of character in the string The access character will be truncated and left unread.

for eg = char a[10];
scanf ("%.5s", a);

In the above example the input string RAM will be stored as



Where as the input string KRISHNA will be stored as



Using get char and gets function

We can use get char function repeatedly to read successive using characters from the input and place them into a character array. Thus an entire line of text can be read and stored in an array. The reading is terminated when the new line character is entered and then the null character is inserted at the end of the string. The syntax for get char function.

```
char ch;
ch = getchar();
```

for eg Write a program to read line of text containing a series of words from the terminal

```
# include <stdio.h>
main()
{
char line[81], character;
int c;
c=0;
printf("Enter text press <Return> at end \n");
do
{
char = getchar();
int[c] = character;
c++;
```

```
}
While (character != '\n');
    C=C-1;
line[C] = '\0';
printf("\n%s\n", line);
}
```

Another and more convinent method of reading a string of text containing white spaces is to use the library function gets available in the Header file <stdio.h> The syntax of this function is as follows -

```
gets(ch);
```

For eg The code segment

```
char line[80];
gets(line);
printf("%s", line);
```

Reads a line of text from the keyboard and Display it on the screen.

C Does not provides operators that work on string Directly For instance we can not Assign one String to Another Directly

for eg The Assignment statment are not valid

```
String = "ABC"
String = String;
```

Q) Write a Program to copy one string into Another, Count the Number of character copies

```
#Include <Stdio.h>
void main()
{
char line[80], line1[80];
int c;
printf("Enter text press <Return> at end \n");
scanf("%s", line1)
for (c=0; line1[c] != '\0'; c++)
    line[c] = line1[c];
    line[c] = '\0';
printf("No of character copied = %d", c);
}
```

Writing strings to screen

- Using Printf function

=> We have used the printf function with "%s format to print strings to the screen The format "%s can be used to display an Array of character that is terminated By the Null character \0.

For eg - The statment printf("%s", Name); this can be used to Display the Entire content of the Array Name.

- Using put char and puts functions.

=> Like getchar C suppoarts Another character handling function. Function to output the value of character variables It takes the following form -

char ch = 'N';
putchar(ch);

This statment is equivalent to printf("%c", ch);

We can use this function repeatedly to output a string of character stored in an Array using a loop.

for eg
```
char a[6] = "GOODS"
for (int i=0; i<5; i++)
    putchar(a[i]);
```

Another and more convinent Way of printing string values is to use the function puts Declared in the Header file <stdio.h> The general syntax is as follows
puts(str);

Arithmatic Operation on characters
Computers are to Multipulate characters the same way we do with Numbers. Whereever a character Constant or character Variable is used in an Expression it is automatically converted into an Integer value By the System.

For eg - If the Machine uses the Asci representation then
```
char = 'a'
printf("%d", ch);
```
with Display the Number 97 on the Screen. It is Also possible to perform Arithmetic operation on the character constant and Variable.
x = 'D' - 1; is a valid
Statment as ascii the value of D is 68 and therfore the statment will Assign the value 67 to the Variable x.

Putting string Together
Just as we cannot Assign one string to another Directly we cannot Join their string Together By the simple arithmatic addition.

i.e
```
string = string1 + string2
string = string1 + "Hello";
```
are not valid. The character from string1 and string2 should Be copied into the string one after the other. The size of the array string should Be large enough to Hold the total character of string1 and string2 The process of combining two string1 together is called # Concatenation

For eg Write a Program in which the Name of Employee of an organization are stored in three Array Namely First Name, Second Name and last Name, Finally Concatenate the Three parts into one string to Be called Name.

```
#include <stdio.h>
void main()
{
int i, j, k;
char first Name[10] = {"VISWANATH"};
char second Name[10] = {"PRATAP"};
char third Name[10] = {"SINGH"};
        char Name[30];
for(i=0; first Name[i] != '\0'; i++)
Name[i] = first Name[i];
        Name[i] = ' ';
for(j=0; second Name[j] != '\0'; j++)
Name[i+j+1] = second_Name[j];
    name[i+j+1] = ' ';
for(k=0; last Name[k] != '\0'; k++)
Name[i+j+k+2] = last Name[k];
    Name[i+j+k+2] = '\0';
printf("\n\n");
printf("%s\n", Name);
}
```

Comparison of Two Strings

C Does Not permit the comparision of Two string Directly i.e the statment:-

if (Name1 == Name2)

if (Name1 == "Hello")

are NOT valid. It is Therefore Necessary to Compare the two String to Be Tested character By Character The comparision is Done until there is a Mismatch os one of the string terminate into a Null character

The syntax for the above is as follows

```
i=0;
while(str1[i]==str2[i] && str[i]!='\0'
        && str2[i]!='\0')
        i++;
if(str[i]=='\0' && str2[i]=='\0')
printf("Strings are Equal\n");
    else
printf("Strings are Not Equal\n");
}
```

Comparision of Two strings -

C Does not permit the comparision of two string Directly i.e the statment if (Name1 == Name2) if (Name1 == "Hello") are Not Valid. It is therefore Necessary to compare the two string to Be tested character By character the comparision is Done until there is ~~Mismatch~~

String Handling Function

The Most commonly used string Handling function are as follows -

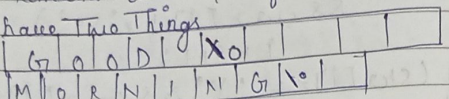
Strcat(); The Strcat() function join two string together A general syntax of this function is as follows -

```
Strcat();
    Strcat(String1, string2);
```

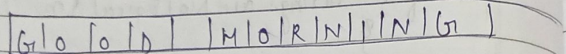
Where string1 & string2 are character arrays. When the function Strcat(); is executed string to is appended to String1. It Does ~~Not~~ Show By

Removing the Null character at end of string 1 and placing string 2 from There. The string at string 2 Remains Unchanged for eg-

We have Two Things



Then after Executing strcat() function Str1 will look like as



are str2 Remains Unchanged.

- Strcmp() - The strcmp() function Compares two things By the Arrangment and as value zero if they are Equal. If they are Not it gives the Numeric Difference Between the first Non-Matching Characters in the string. The General Syntax of this function is as follows

strcmp (string1, string);

Where string1 & string2 May Be string variables or string Character

Our Major Concern is to Determine whether the string are Equal and if Not which is alphabitically above.

For eg strcmp ("Their", "There"); will Return a value of -9 which is the Numeric Difference B/w ascii I and ascii R i.e I-R in ASCII Code is -9. If the value is Negative string1 is alphabatically above string2

o) strlen() This function founds & Return the No. of Character in a string the General Syntax of this function strlen();

n = strlen (string);

Where N is an Integer variable which Recives the value of the len of the string. The argument May Be a string constant. The counting Ends at the first Null character

- Strcpy() The strcpy() function works almost like a string Assignment operators The General syntax as follows

strcpy();

strcpy (string1, string2);

In this we will Assign the contents of string two to string1. May Be a string variable or a string constant for eg- strcpy (string1, "DELHI"); will Assign the string Delhi to the string variable string1

```
#include <stdio.h>
void main()
{ char s1[20] s2[20] s3[20];
int x, l1, l2, l3
printf ("\n\n enter into string Con \n");
printf (" ? ");
scanf (" %s %s", s1, s2);
x = strcmpp (s1, s2);
if (x!=0)
{ printf ("\n\n string one Not Equal \n")
  strcpy (s1, s2);
}
else
printf ("\n\n string are Equal \n");
```

```
L1 = strlen(S1);
L2 = strlen(S2);
L3 = strlen(S3);
printf("\nS1 = %s\t length = %d character\n", S1, L1);
printf("\nS1 = %s\t length = %d character\n",
                                    S2, L2);
printf("S3 = %s\t length = %d character\n",
                                    S3, L3);
}
```

```
l1 = strlen (s1);
l2 = strlen (s2);
l3 = strlen (s3);
printf ("\n s1 = %s \t length = %d a character \n", s1, l1);
printf ("\n s1 = %s \t length = %d character \n",
                                    s2, l2);
printf ("\n s3 = %s \t length = %d character \n",
                                    s3, l3);
}
```

Unit IV

Structure

Array is a collection of some type of Elements But in Many Real life Application we may need to get Different type of logically Related Data for eg- we want to create a Record of a function that contains Name, age, Salary of that person then we can't use Arrays Because all the three Data elements are of Different type.

To store these Related things of Different Data type we use Data structure which is capable of storing heterogeneous Data.

Definition of Structure

A structure is a combination of Different Data Types or Heterogeneous data type its general Syntax of structure is

```
struct tagname
{
  datatype member1;
  datatype member2;
  .
  .
  datatype membern;
};
```

Here struct is a keyword which tells a compiler that Structure is Being Define and member1, member2 membern are known as members of the structure and are declare inside Every Braces. Tagname is the Name of the Structure and it is used further in the Pogram Declare Variables of this structure Type.

Here Stu1, Stu2, Stu3 are variable of Type Struct Student when we Declare a variable while Defining the Structure Template the Tag Name is optional so we can Also Declare them As

```
Struct
{
  Char Name[20];
  Int Rollno;
  Float Marks;
} Str1, Str2, Str3;
```

But If we Declare variables in this way then we will Not Able to Declare other variables of this Structure Type Anywhere in the program

Using the Structure Tag

Let us take an Example of Defining a Structure Template

```
Struct Student
{
    Char Name[20];
    Int Rollno;
    Float Marks;
};
```

Declaring Structure Variable

By Defining a Structure we have only created a formate the Actual use of Structure will Be when we Declare Variables Based on this format you can Declare Structure Variable in two ways

- Structure Definition
- Using a Structure Tag

Structure Definition

```
Struct Student
{
    Char Name[20];
    Int Rollno;
    float Marks;
} Str1, Str2, Str3
```

the program

Using the Structure Tag

```
Struct Student
{
Char Name[20];
Int Rollno;
Float Marks;
};
struct Student Stu1, Stu2, Stu3;
```

Here Stu1, Stu2, Stu3 are Structure Variables that are Declare Using the Structure Tag Student.

Initialization of Structure Variable

The Syntax of Initializing Structure Variable is Similar to Array. All the values Are given in { } and the Number Order and Type of these value should Be same as in

the Structure Template Defination.

for eg.

```
Structure Student
{
    Char Name [20]
    Int Roll no.;
    float Marks;
};
Str1 = {"John", 30, 74.5};
Struct Student Str 2 = {"Sema", 25, 76};
```

Here value of Members of Stu1 will Be Jon for Name 30 for Roll no. and 74.5 for Marks. Similarly the value of Member Stu. 2 will Be Sema for Name 25 for Roll no. and 76 for Marks.

Excessing Members of Structure

For Excessing Any Member of a Structure Vaeiable we use Dot operator which is Also known as Membership operator The formate for Accessing a Structure Member is.

⇒ Struct Variable . member ;

for eg- Write a Program to Accept the Information of your Teacher having the following fields i.e. Name Department. Designation and Qulification.

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

Strut Str
{
Char Name [20];
Char department [20];
Char qualification [20];
Char Designation [20];
};

Void main ()
{
clrscr ();
Str n;
printf ("Enter the Name:");
Scanf ("%s", n.name);
printf ("Enter the Department");
Scanf ("%s", n. Department);
printf ("Enter the Qualification:");
Scanf ("%s", x. qualification);
printf ("Enter the Designation");
Scanf ("%s", x. designation);
printf ("Name is : %s", x. name);
```

printf ("Department is : %s ", x.Department);
printf ("Qualification is : %s" x.Qualification)
printf ("Designation is : %s", x.Designation)
getch();
}

Nested Structure

The Members of a Structure can Be of Any Data Type Including Another Structure Type, We can Include A Structure With in Another Structure The Structure Variable can Be Member of Another Structure This is called Nesting of the Structure.

```
Struct Tag1
{
Member1;
Member 2;
Struct Tag2 {
        Member 1;
        member 2;
        :
        member n;
      } var 2;
Member n;
} var1;
```

For Accessing Member 1 of Inner Structure We will Write Var1, Var 2. Member 1

Here is an Example of Nested Structure.

```
Struct Student
{
Char Name [10];
Int Rollno;
Struct Dob;
        Int date;
        Int month;
        Int year;
      } birthdate;
} Student;
} Str1;
```

Here We have Define a Structure Dob Inside the Structure Student. This Structure Dob has 3 members i.e Data month and year. Birth date is a variable of Type Struct Data We can Access the members of Inner Structure has Str1. Birth date. date.

Union

Union is a Derived Data Type like Structure And it Can Also contain members of Diff. Data Type.

The Syntax Defination of an Union. Declaration of an Union And Accessing Member is similar to the Structure But Here keyword union is used Instead of Struct. The Main Difference Between Union & Structure is in the way of Memory is Allocated for the Members.

In the Structure Each Member has its own memory where as Members of union share the same memory location. When a variable of Type union is Declared Compiler Allocates Sufficient Memory to Hold the largest member in the union. Since All Members share the same Memory location Hence we can use only one member at a time. Thus union is used for saving memory.

The general Syntax of an union is as follows -

```
union Tagname
{
    member 1;
    member 2;
    :
    upto member n;
} var 1;
```

Difference B/w Structure & union

Structure	union
struct Student	union Student
{	{
char Name [10];	char Name [10];
int Roll no;	int Roll no;
float marks;	float marks;
} str 1;	} str 1;

Write a program to compare the memory allocated for a union And Structure variable.

```
#include <stdio.h>
#include <conio.h>
struct stag {
    int i;
    char c;
    float f;
};
union tag {
    int i;
    char c;
    float f;
};
void main()
{
    clrscr();
    union tag u var;
    struct tag s var;
printf("Size of svar %d\n", Size of(svar));
printf("Address of svar %u \n", & svar);
printf("Address of member %u, %u, %u, \n\n\n",
        & svar.i, & svar.c, & svar.f);
printf("Address of uvar %u \n", & uvar);
printf("Address of member %u, %u, %u, \n\n\n",
        & uvar.i, & uvar.f);
    getch();
}
```

Unit V

Introduction to C Pre-Processor

C has a special features of pre-processor which makes it Different from other High level language that doesn't have this type of facility. Some advantages of using pre-processor are.

- Readability of the program is increased.
- program Modification Become Easily.
- Makes the program portable of efficient.

As we know that, the Coed you write will Right is Translated in to object Coed By the compiler But Before Being compiler the coed is passed to the C processor. The C pre-processor is Can the hole Source coed & Modify it which is then given to the compiler.

Source Coed
↓
Pre Processor
↓
Object code

The line processing with the # Symbol are known as pre-processor Directives. Some features of their are Each pre-processor Directives start with a # Symbol.

- There Can Be only one Directive on a line.
- There is no semicolon at the end of Directives.
- The pre processor Directive Can Be place anywhere in a program But Usually return at the Begning of the program.

SNo	Directive	Description
1	# Define	Substitute a pre processor Macros
2	# Include	Insert a particular header from another file
3	# undef	Undefine a pre-processor Macro
4	# If	Test if a compile time condition is true
5	# else	The Alternative for # If

Pre define Macros's

SNo	Macros's	Description
1)	DATE	The Current Data has a character lateral in MM/DD/YY
2)	Time	The current time as a character lateral in HH:MM:SS
3)	File	This contain the current file Name as string lateral.
4)	Line	This contain the current line no. as the Decimal Constant
5)	STDC	Define as one when the compiles Compile with the ANSI Standard

```
#Include<stdio.h>
main()
{ Printf("File:%s\n",__FILE__);
Printf("Date:%s\n",__DATE__);
Printf("Time:%s\n",__TIME__);
Printf("Line:%d\n",__LINE__);
Printf("STDC:%d\n",__STDC__);
```

Types of Directives

These Directives can be Divided into 3 catogries
1) Macro's Substitution Directive
2) File Inclusion Directive
3) Compiler Control Directives

→ Macro S.D - It is a process where an Identifier in a Program is Replaced By a Pre-Define String composed of one or More tokens The Pre Processor accomplishes this task under the Direction of # Define Statment This statment usually known as Macro's Definition which takes the following general forms

Define Identifier String

If this statment is included in the Program at the Beginning Then the pre-processor Replace every occurance of the Identifier in the Source code By the String. The String May Be any text while the Identifier Must Be a Valid Name There are Different forms of Macro's Substitution the Most common form are -

i) Simple Macro Substitution
define 3.14
Define COUNT 5
Define CAPITAL DELHI

ii) Macro's with Arguments
Define CUBE (x) [x * x * x]

iii) Nesting of Macro's
Define M 5
Define N M+1

Undefining a Macro

A Defined Macro Can Be Undefined By Using the following Statments

#Undefin Identifier

This is useful when we want to Restrict the Defination only to a particular part of the program.

File Inclusion

An External file containing function or Macro Definations Can Be Included as a part of a program so that We Need Not Rewrite these functions or Macro Definitation this is Achived By a Pore-processor Directive

#Include file Name

Where File Name Is the Name of the file containing the Required Defination.

let us Assum that We have Created a following three files.

Syntax.C - Contains Syntax Defination.
Stat.C - Contains Statical function
Test.C Contains Testfunction

Now we Can Make Use of a Defination or Function Contained In Any of these files By Including them as

```
#include <stdio.h>
#include "SYNTAX.C"
#include "STAT.C"
#include "test.C"
#define m 50
main()
{
```

Compiler Control Directives

The C pre processor offer a feature known as Conditional Compilation which can be used to switch on or off a particular line or group of lines in a program

#ifdef, #else, #if, #endif are some examples of these types of directives

Bitwise Operator

C has a list of supporting of special operators known as Bitwise Operator for manipulation of Data at Bit level

These Operator are used for Testing a Bites or Shifting them Right or left.

Bitwise operator may not be applied on float or Double. The following Table list the Bitwise operator & their meanings

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise Exclusive OR
>>	Right Shift
<<	left shift

⇒ Bitwise AND Operator (&)

The output of Bitwise AND is one if the corresponding Bites of two operants is 1. If either Bite of an Operant is 0 the Result of corresponding Bite is Evaluated to zero.

let us take An Example of Bitwise AND operator of Two Integers 12 AND 25

```
12 = 0001100
25 = 0001100 1
& = 0000 0100  = 8
```

Write a Program to Elestruate the use of logical & operator.

```
#Include <Stdio.h>
#Include <Conio.h>
void Main()
{
int a=12, b=25;
printf("output = %d", a & b);
getch();
}
```

→ Bitwise OR Operator

The output of Bitwise OR is one (1) If Atleast one corresponding Bite of two operants is 1. In C programming Bitwise OR Operator is Denoted By single pipeline for eg 25

```
35 = 00010011
17 = 00001001
or   00011011  — 27
```

Q) Write a Program

```
# include <stdio.h>
# include <conio.h>
Void main()
{
int a=35, b=17;
printf("output = %d", a&b);
getch();
}
```

Bitwise Compliment Operator

The Bitwise Compliment operator is an unary Operator it changes 1 to 0 and 0 to 1 it is Denoted By the Symbol ~

for eg 35 = 00100011
~35 = 11000100

Bitwise X or Operator

The Reset of Bitwise X or Operator is 1 If the corresponding Bite of two operants are opposite of is Denoted By ^

```
35 = 00100011
12 = 00001100
^  = 00101111
```

Bitwise Shift Operator

The Shift Operator are used to move Bite patterns either to the left or to the Right. The Shift Operators are Represented By the Symbols << >> and are used to the following form

left shift = op << n
Right shift = op >> n

Q) 10010110 << 3
10010000 Left Shift

10010110 >> 3
00010110 Right shift

Masking

It refers to the process of Extracting Desired Bites from OR Transforming Desired Bites in a Variable By using Bitwise operations. The operant that is used to perform masking is called the Mask

for eg
```
y = x & mask;
y = x | mask;
```

It is used in Many Diff Way

1. To Decide Bite pattern of an Integer Variable
2. To copy a partion of given Bite pattern to a New variable while the reminder of the New Variable is filed with Zeroes By Using Bitwise END
3. To copy a portion of a given Bite pattern to a New Variable where the Reminder

of the New Variable is field with once By Using Bitwise OR Operator.

4. To copy a portion of a given Bite pattern to a New Variable while the Remender of the Original Bite patter in Inverted with in the New Variable By Using Bitwise Exclusive OR Operator.

```
# include < stdio.h >
void main ()
{
  int a;
print f ("output = %d", a << 1);
}
```

WAP to find out the Sum of Two Intogers Entered By User Through Structure

```
#include < stdio.h >
~~void main()~~
  Struct Sum
{
  int a, b;
};
  void Main ()
{
  Struct Sum x;
printf ("Enter the value a);
Scanf ("%d", &x.a);
printf (" enter the value b);
Scanf ("%d", &x.b);
x.sum = x.a + x.b;
printf ("%d", x.sum);
}
```

Unit - VI

Write a Program to Record Data of Books Using Structure having the member Element, Title Author, Subj. And Book id.

```
# include < string >{
  ~~Struct Book~~
{
  char, Title, ~~Author, Subject, Book~~ id.
};
  void main ()
{
  Cluscon ();
```

```
# include < string.h >
# include < stdio.h >
  Struct books
{
 char title [20];
 char author [20];
 Char subject [20];
 ~~int~~ book_id;
};
 Void main()
 {
  Struct book n;
 printf ("Enter the title");
 scanf ("%s", &n.title);
 printf ("Enter the author");
 Scanf ("%s", &n.author);
 printf("Enter the Subject);
  Scanf ("%s", &n.subject);
 printf ("Enter the book id");
  Scanf ("%d", &n.book id);
  printf ("title is : %s", n.title);
```

```
printf("author is:%s", n.author);
printf("Subject is:%s", n.subject);
printf("book id is:%d", n.bookid);
}.
```