Date / /

UNIT = 1

Chapter = 1

[ARRAYS]

The variables that we have till now are capable of storing only one value at a time.
Consider a situation when we want to store and display the age of 100 employes. for this, we have to do the following :-

(i) Declare 100 different variables to store the age of the employees.
(ii) Assign a value to each variable.
(iii) Display the value of each variable.

It would be very difficult to handle so many variables in the program and the program would become very lengthy. The concept of array is useful in these type of situation, where we can group similar type of data items.

Array :-
An array is a collection of similar type of data itmes and each data item is called an element of the array. The datatype of elements may be any valid data type like :- int, Float, char.

The elements of array share the same variable but each element has a different index number known as subscripts.
For about the problems we can take array variable age [100] of type int.

Date / /

The size of this array variable is 100. so it is capable of storing 100 integer value. The individual elements of this array are:-

age[0], age[1], age[2], age[3], ---------- age[99]

In C, the subscripts starts from 0.
so, age[0] is first element and age[1] is second element.

Array can be single dimensional or multi-dimensional. The number of subscripts determine the dimension of the array.

1-D array has one subscripts.
2-D array has two subscripts.
and so on.

- The one dimensional arrays are known as vector.
- The two dimensional arrays are known as Matrices.

V. Imp ★ 1st Sess

One Dimensional Array :→

1. Declaration of 1-D Array :-

Syntax for declaration of an array is :-

datatype Array_name[array size];

Here, Array_name denotes the name of the array and it can be any valid C Identifier.
data-type is the datatype of elements of array.
Here some examples of array declaration :-

```
int age[100];
float sal[15];
char grade[20];
```

datatype, array_name[size];

int → datatype
marks → array_name (variable)
[50] → size

Date / /

When the array is declare the compiler allocate space in memory to hold whole the elements of the array. so, the compiler should know the size of array and at the compile time.

Accessing 1-D Array Element :-

The elements of an array can be access by the specifying the array name followed by sub-scripts in brackets.

In C the array subscript start from 0, hence if based an array of size 5 then the valid subscript will be shown 0 to 4. The last valid subscript 1 less then the size of array.

This last valid subscript in sometime known as the upper bound of the array and '0' is known as the lower bound of the array.

Let us take an array :-

```
int arr[5];
```

Size of array is 5 can hold 5 integer elements
The element of this array :-

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

↓ Lower bound — ↓ Upper bound

Processing 1-D Arrays :-

For processing array, we generally use a for loop and the loop variable is huge at the place of sub-script. The initial value of loop variable is taken '0'.

Since, array subscript start from '0'. The loop variable is increase by 1 each time. so, that we can access and process the next element in the array.

Note :- The total no. of passes in the loop will be equal to the no. of elements in the array. is in each pass, we will process one element.

Example :- Suppose, arr[10] is an array of int type.

① Reading value in arr[10] :-

```
for(i=0; i<10; i++)
scanf("%d", &arr[i]);
```

② Displaying the value of arr[10] :-

```
for(i=0; i<10; i++)
printf("value of array %d", arr[i]);
```

1st Sess

Program 1 :- Write a program to input values into an Array and display them :-

```
#include<stdio.h>
#include<conio.h>
  void main()
  {
    clrscr();
  int arr[5], i;
  for(i=0; i<5; i++)
  {
  printf("Value of array :%d", arr[i]);
```

Date / /

```
printf("Enter the value of array:");
scanf("%d", &arr[i]);
}
for(i=0; i<5; i++)
  {
    printf("Value of array:%d", arr[i]);
    printf("\n");
    }
      getch();
      }
```

Output:-

Enter the value of array: 1
Enter the value of array: 2
Enter the value of array: 3
Enter the value of array: 4
Enter the value of array: 5

Value of array : 1
Value of array : 2
Value of array : 3
Value of array : 4
Value of array : 5

1st scss.

Program 2:- Write a program to add the element of an Array :-

```
#include<stdio.h>
#include<conio.h>
  void main()
  {
    clrscr();
  int i, sum=0, arr[10];
```

Date / /

```
for(i=0; i<5; i++)
{
printf("Enter the value of array:");
scanf("%d", &arr[i]);
}
sum+=arr[i];
printf("The sum of an Array:%d \n", arr[i] sum);
printf("\n");
getch();
}
```

Output :- Enter the value of array: 1
Enter the value of array: 2
Enter the value of array: 3
Enter the value of array: 4
Enter the value of array: 5

Value of Array: 15

* Adding All the Elements of arr[10] :-

```
sum = 0;
for(i=0; i<10; i++)
sum+=arr[i];
```

1st Sess ✓ ★ Two 2 Dimensional Array :→

1. Declaration and associating individual element of 2D array :-

The Syntax of declaration of 2D array is similar to that of 1D Array but here we have 2 subscripts :-

```
datatype array_name [row size] [column size];
```

Date / /

datatype array-name [row size] [column size];

Here, row size specify the number of rows and column size specify the number of column in array.

☆ The total no. of element in the array are
row size * column size

For Example :- int arr[4] [5];

Here arr[4] [5] is a 2D array with 4 rows and 5 columns.
first element of this array → arr[0] [0]
last element of this array → arr[3] [4]

	col 0	col 1	col 2	col 3	col 4
row 0	arr [0] [0]	arr [0] [1]	arr [0] [2]	arr [0] [3]	arr [0] [4]
row 1	arr [1] [0]	arr [1] [1]	arr [1] [2]	arr [1] [3]	arr [1] [4]
row 2	arr [2] [0]	arr [2] [1]	arr [2] [2]	arr [2] [3]	arr [2] [4]
row 3	arr [3] [0]	arr [3] [1]	arr [3] [2]	arr [3] [3]	arr [3] [4]

Processing 2-D Array Elements :-
For processing 2-D array we use two for loops the outer for loop correspond to row and inner for loop correspond to columns.
int arr[4] [5];

i) Taking Input in arr :-

```
for(i=0; i<4; i++)
for(j=0; j<5; j++)
```

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

3. Display value of arr :-

```
for(i=0; i<4; i++)
 for(j=0; j<5; j++)
 {
  printf("%d", arr[i][j]);
 }
```

Program based on 1-D Array :-

Program 3:- A Program to input an array and count even and odd numbers :-

```
#include<stdio.h>
#include<conio.h>
 void main()
{
 int arr[10], i, even=0, odd=0;
printf("Enter ten numbers to check:");
 for(i=0; i<10; i++)
  {
  scanf("%d", &arr[i]);
  if(arr[i]%2==0)
  {
  even++;
  }
  else
   {
   odd++;
   }
   }
```

Date / /

```
printf("Number of Even no. are : %d\n Number of odd no. are : %d\n
        getch();
    }
```

Output :- Enter ten numbers to check: 1
2
3
4
5
6
7
8
9
10
Number of Even no. are: 5
Number of Odd no. are: 5

Program based on 2-D Array :-

1st sess.

Program 4 :- A Program to input and display a matrix :-

```
#include<stdio.h>
#include<conio.h>
void main()
{
  clrscr();
 int mat[3][3], i, j;
printf("Enter nine values of matrix:");
    for(i=0; i<3; i++)
    for(j=0; j<3; j++)
   {
     scanf("%d", &mat[i][j]);
   }
```

Date / /

```
printf("Matrix entered by user is:");
    for(i=0; i<3; i++)
     for(j=0; j<3; j++)
{
    if(i==1 && j==0)
    printf("\n\n");
    if(i==2 && j==0)
    printf("\n\n");
    printf("%d", mat[i][j]);
    }
      getch();
       }
```

Output :- Enter elements of 3*3 matrix:

or

Enter nine values of matrix: 1

2

3

4

5

6

7

8

9

Matrix entered by user is: 1 2 3

4 5 6

7 8 9

Date / /

1st sess.

Program 5:- A program for addition of two matrices:-

```
#include<stdio.h>
#include<conio.h>
  void main()
{
  clrscr();
int  i,j, mat1[3][3], mat2[3][3], mat3[3][3];
printf("Enter nine elements of first matrix:");
  for(i=0; i<3; i++)
  for(j=0; j<3 ; j++)
   {
    scanf("%d", &mat1[i][j]);
   }
printf("Enter the nine elements of second matrix:");
  for(i=0; i<3; i++)
  for(j=0; j<3 ; j++)
{
 scanf("%d", &mat2[i][j]);
}
  printf("sum:");
   for(i=0; i<3; i++)
   for(j=0; j<3; j++)
  {
    mat3[i][j] = mat1[i][j] + mat2[i][j];
   if(i==1 && j==0)
   printf("\n\n");
   if(i==2 && j==0)
   printf("\n\n");
printf("\t%d", mat3[i][j]);
   }
      getch();
   }
```

Date / /

Output :- Enter Nine elements of fist matrix : 1
2
3
4
5
6
7
8
9
Enter the nine elements of second Matrix: 2
4
6
8
10
12
14
16
18

Sum: 3 6 9
12 15 18
21 24 27

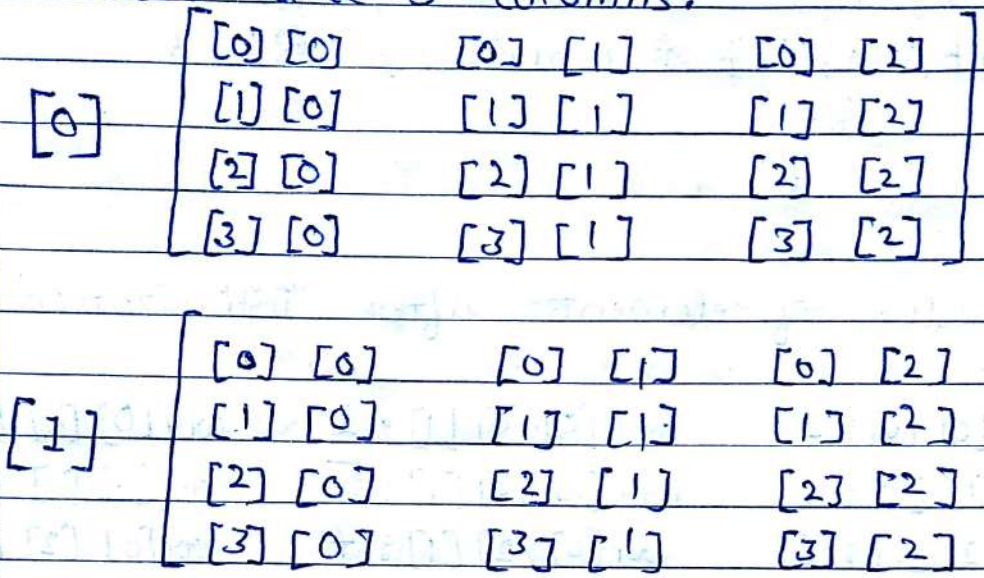
1st sess.

☆ Multi-Dimensional Array :→
Arrays with more than two Dimensions :-
We will just give a brief or over view 3-D arrays. We can think of 3D array as an array of 2-D array. for example, if we have an array,

int arr[2] [4] [3];

Date / /

We can think of this array which consist of 2-D array and each of those 2D array have 4 rows and 3 columns.



The individual element are :→

arr[0][0][0], arr[0][0][1], arr[0][0][2], arr[0][1][0] ..... arr[0][3][2]

arr[1][0][0], arr[1][0][1], arr[1][0][2], arr[1][1][0] ..... arr[1][3][2]

Total no. of elements in above array :-

$2 * 4 * 3 = 24$

This array can be initialized as :-

```
int arr[2][4][3]
  {
{1,2,3} * matrix 0, Row 0
{4,5} * matrix 0, Row 1
{6,7,8} * matrix 0, Row 2
{9}   * matrix 0, Row 3
   }

{
```

Date / /

{10, 11} * matrix 1, Row 0
{12, 13, 14} * matrix 1, Row 1
{15, 16} * matrix 1, Row 2
{17, 18, 19} * matrix 1, Row 3
}
}

The value of elements after initialization are as :-

[0]
arr[0][0][0] : 1 arr[0][0][1] : 2 arr[0][0][2] : 3
arr[0][1][0] : 4 arr[0][1][1] : 5 arr[0][1][2] : 0
arr[0][2][0] : 6 arr[0][2][1] : 7 arr[0][2][2] : 8
arr[0][3][0] : 9 arr[0][3][1] : 0 arr[0][3][2] : 0

[1]
arr[1][0][0] : 10 arr[1][0][1] : 11 arr[1][0][2] : 0
arr[1][1][0] : 12 arr[1][1][1] : 13 arr[1][1][2] : 14
arr[1][2][0] : 15 arr[1][2][1] : 16 arr[1][2][2] : 0
arr[1][3][0] : 17 arr[1][3][1] : 18 arr[1][3][2] : 19

1st sess.

1-D Array and Functions :→

1. Passing indivisual array and function :-

We know that array element is treated created as any other simple variable any other program so we can pass indivisual array elements as arguement to a function like other simple variables.

3rd sess

Program 6 :- Write a program to pass array elements to a function :-

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

Date / /

```
int check(int num);
int main()
{
clrscr();
int arr[10], i;
for(i=0; i<10; i++)
{
printf("Enter a number to check:");
scanf("%d", &arr[i]);
check(arr[i]);
}
getch();
}
int check(int num)
{
if(num % 2 == 0)
printf("No. is even\n\n");
else
printf("No. is odd\n\n");
}
```

UNIT :→ 2

Date / /

☆ Pointers :→

1. Pointer Variable :- A pointer is a variable that store memory address like all other variables. It also has a Name, has to be declare and occupie some space in memory. It is called pointer, because it points to a particular location in memory by storing the address of that location.

2. Declaration of pointer variables :-
Like other variables pointer variable should also be declare before being used. The general syntax of declaration of variable is :-

```
data_type *p_name;
```

Here p_name is a name of pointer variable which should be a valid C identifier.
The Asterisk '*' preceding this name informs the compiler that the variable is declare as a pointer.
Here data_type is known as the base type of the pointer.

Let us take some pointer declaration.

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

Here iptr is a pointer that should point to variable of type int.
Similarly fptr and cptr are should point to variables of float and char-type.

Date / /

We have done in the 3rd declaration statement where Ch1 and Ch2 are declare as a variable of type char.

Operators Use with Pointers :→

There are two basic operators used with pointers:-

① Address Operator : & (Ampersand) :-
C provides an address operator which writtens the address of a variable when placed before a.

② Indirection Operator : * (Asterik) :-

☆ Pointer Arithemetic :→

1st sess.

All types of arithemetic operations are not possible with pointers. The only valid operations that can be perform are as :-

1. Addition of an intiger to a pointer & increment operator.
2. Subtraction of a intiger from a pointer and decrement operations.
3. Subtraction of a pointer from another pointer of a same time.

For Ex :-
If we have an integer value PI which contain address 1000, then on incrementing we get,
Incrementing we get 1002 instead of 1001.

Date / /

This is because int data-type is 2.

Similarly on decrementing PI we will get, 998 instead of 999.

The expression (Pi+3) will represent the address, Let us see pointer arthemetic for int, float and char pointers.

```
int   a=5 , *Pi=&a;
float b=2.2, *Pf=&b;
char  c='x', *Pc=&c;
```

Suppose the address of variable a, b, c are 1000, 4000, 5000 so initially values of P1, P2, P3 will be 1000, 4000, 5000

Pi++ or ++Pi	= 1002
Pi = Pi-3	= 996
Pi = Pi+5	= 1006
Pi-- or --Pi	= 1004

Pf++ or ++Pf	= 4004
Pf = Pf-3	= 3992
Pf = Pf+5	= 4012
Pf-- or --Pf	= 4008

Pc++ or ++Pc	= 5001
Pc = Pc-3	= 4998
Pc = Pc+5	= 5003
Pc-- or --Pc	= 5002

Date / /

☆ Array with Pointer :→

```
#include<stdio.h>
#include<conio.h>
 void  main()
{
   int  marks[] = {10, 20, 30};
   int  *point[3], i;
   for(i=0; i<3; i++);
   {
      printf("%d", marks[i]);
      point[i] = &marks[i];
   }for(i = 0; i<3; i++)
    {
       printf("%d", *point[i]);
    }
      getch();
    }
```

Output :- 10 20 30,

10 20 30

Q :→ Write a program for Understanding the concept of Array of Pointer :→

```
#include<stdio.h>
#include<conio.h>
   void  main()
  {
    int  *pa[3];
```

Date / /

```
int i, a=5, b=10, c=15;
  Pa[0] = &a;
  pa[1] = &b;
  pa[2] = &c;

for(i=0; i<3; i++)
{
    printf("%d", pa[i]);
    printf("%d", *pa[i]);

    getch();
}
```

Output :-

5
10
15

Call by Value	Call by Reference
`void disp(int x);`	`void disp(int *x);`
`main()`	`main()`
`{`	`{`
`int a=100;`	`int a=100;`
`disp(a);`	`disp(&a);`
`void disp(int x)`	`void disp(int *x)`
`{`	`{`
`x=x+100;`	`*x=*x+100;`
`}`	`}`

Date / /

Call by Value	Call by Reference
Output :-	Output :-
x = 200	a = 100
a = 100	x = 2156

9m ☆ Dynamic Memory Allocation :→

The memory Allocation that we have done till now was static memory allocation.

The memory that could be used by the pro-gram was fixed we could not increase or the decrease the size of memory during the execution of program.

In many applications it is not possible to predict how much memory would be needed by the program at run time.

Ex :-

```
int emp[200]
```

In an array it is must to specify the size of array while declaring, so the size of this array will be fixed during run time.

Now two types of problem make occur.

① The First case is that the no. of values is to be store is less then the size of array, and hence, there is wastage of memory.

For example :- If we have to store only 50 value then space for 150 value (300 bytes) is wasted.

② In second case, our program face, if want to store more values than the size of array.

For Ex :- If there is need to store 210 value

Date / /

in the above array.

To overcome these problems, we should be able to allocate memory at run time.
The process of Allocating memory at the time of execution is called dynamic memory allocation.
The allocation and release of this memory space can be done with the help of some built-in-functions, whose protypes are found in alloc.h, stdlib.h, Hearder files.

① Malloc ( )
Declaration :→

```
void *malloc(size_t size);
```

This function is use to allocate memory dynamicly. The argument size specify the no. of bytes to be allocated.
The type_t define in <stdlib.h> on success malloc ( ) returns a pointer to the first byte of allocated memory.

It is generally use as :-

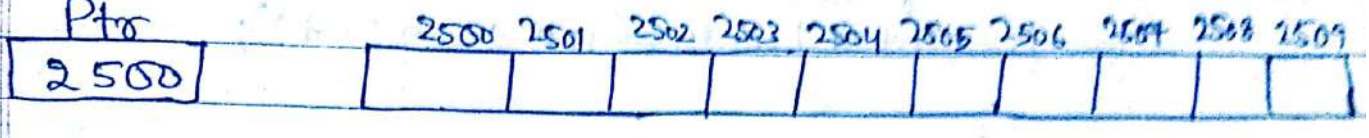
```
Ptr = (datatype *) malloc (Specified size);
```

Where ptr is a pointer of type_datatype and specified size, is the size in bytes required to be reserved in memory.
The expression (datatype *) is used to typecast the pointer written by malloc ( ).

For Example :→

Date / /

```
int *ptr;
ptr = (int *) malloc(10);
```



This allocates 10 continguous bytes of memory space and the address of first byte is stored in the pointer variable. This space can hold 5 intigers. The allocated memory contains garbage ~~memory~~ or valve.

We can use sizeof operator to make a program portable and regdable.

```
ptr = (int *) malloc(5 * sizeof(int));
```

This allocates the memory space to hold 5 intiger values.

If there is not sufficient memory available in heap then malloc() returns 'NULL'. so, we should always check the valve return by malloc().

```
ptr = (float *) malloc(10 * sizeof (float));

if(ptr == NULL)
    printf("Sufficient Memory not Available");
```

Q:- Write a program to understand Dynamic allocation of a memory ?

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

```
void main()
{
  int *p, n, i;
printf("Enter the number");
scanf("%d", &n);
p=(int *) malloc(n * sizeof(int));

 if(p == NULL)
 {
 printf("Memory not Available");
    exist(1);
  }
    for(i=0; i<n; i++)
     {
       printf("Enter an integer");
       scanf("%d", p+i);
      }

   for(i=0; i<n; i++)
    {
      printf("        Memory");
      scanf("%d", &*p+i);
     }
       getch();
      }
```

① Declaration :-

① alloc() :-

```
void *calloc (size_t n, size_t size);
```

Date / /

The called Alloc() is used to allocat multiple blocks of memory. It is somewhat smiliar to the malloc() except for two difference.

(i) It takes two arguement the :-
1. Specify the no. of blocks.
2. Specify size of each blocks.

(ii) The other difference between alloc() & malloc() is that the memory allocate by malloc() contain garbage value. While the memory allocated by alloc() is 0.

② realloc() Declaration :→

void * realloc (void * ptr, size_t newsize);

If we want to increase or decrease the memory allocated by malloc() and alloc().
The function realloc() is use to change the size of the memory block. It alter the size of memory block without losing the all data. This is known as reallocation of memory.

UNIT :→ 3

☆ Strings :→

Strings are generally used to store and manipulate data s in text form like words or sentence. There is no seperate datatype for string in 'C'.

They are created as array of typer char. A Character array is string if it end

with a null character.

String Constant or String literal :-
A String constant is a sequence of character inclose in " " (double cots) It is sometimes called a literal.
The double cots are not a part of the string.
Some examples of String constant are :-

"V"
"New Delhi"
"2345"
"Sentence"

Whenever a string constant is written anywhere in program, It is stored somewhere memory as an array of character terminated by null character ('\0').

The String constant itself becomes a pointer to the first character in the array.
For Ex :-
The String "Taj Mahal" will be stored in the memory as,

1000	1001	1002	1003	1004	1005	1006	1007	1008	1009
T	a	j		m	a	h	a	l	\0

Each character occupy 1 byte and compiler automatically insert the \0 null character at the end.
The String constant "Taj Mahal" is actually a pointer to the character to 'T'.

Date / /

If we have a pointer variable of type char* then we can assign the address of the String constant to it as,

```
char *p = "Taj Mahal";
```

① :- W.A.P to print characters of a string and address of each character :-

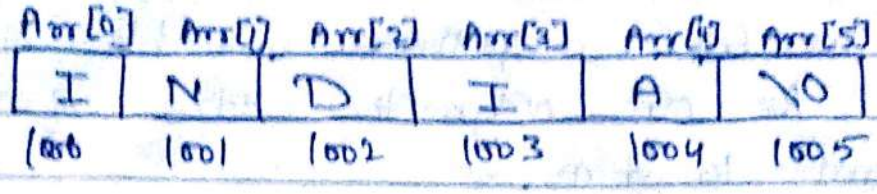
```
#include<stdio.h>
#include<conio.h>
void main()
{
  clrscr();
 int i;

char str[5] = "India";
 for(i=0; i<5; i++)
{
printf("Character is : %c\n", str[i]);
printf("Address is : %u\n", &str[I]);
}
 getch();
}
```

Output :-

I	——	1000
N	——	1001
D	——	1002
I	——	1003
A	——	1004

Date / /



★ String Library functions :→

There are several library functions used to manipulate string. The prototype of these functions are in header file <string.h>. We will discuss some of them below :-

1. Strlen() :→ This function written's the length of the string.
The no. of characters in the string excluding the terminating null character.
For Ex :- Strlen ("India") writtens the value 05. Similarly, if S1 is an array that contain the name "BCA", then Strlen (S1) writtens a value 3.

Q15:- W.A.P to understand the work of Strlen() fun?→

```
#include<stdio.h>
#include<string.h>
    void main()
    {
        clrscr();
    char Str[20];
```

Date / /

```
int length;
printf("Enter the String:");
scanf("%s", str[length]);
length = strlen(str);

printf("Length = %d", length);

getch();
}
```

Output :-

Enter the String: Mayank
length = 6

2. Strcmp() :-

This function is used for comparison of 2 Strings. If the two string match, strcmp() returns a value 0, otherwise return non-zero value.

This function compare the string character by character.

Q16: W.A.P to Understand the work of strcmp() :→

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
void main()
{
clrscr();
Char Str1[10], Str2[10];

printf("Enter the first String");
```

```
scanf("%s", str1);
printf("Enter the second string");
scanf("%s", str2);

if((strcmp(str1, str2)==0)

printf("String are same");
else
printf("String are not same");

getch();
}
```

Output :→

Enter the first String : Mayank
Enter the second string: rana
String are not same.

Enter the first String : Mayank
Enter the second String : Mayank
String are same.

★ Strcpy( ):-

This function is use for copying another string
Here, Strcpy (Str 1, str2) ,

Here Str 2 is the source String & Str 1 is a destinition string
If Str 2 = "BCA", then this function copies BCA into Str 1, this function take pointer two strings as arguements and writtens the pointer to first String.

Date / /

Q1. W.A.P to understand the work of Strcpy():

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
  clrscr();
char Str1[10], Str2[10];

printf("Enter the first String:\n");
scanf("%s", Str1);

printf("Enter the Second String:\n");
scanf("%s", str2);

strcpy(str1, str2);

printf("Str1 = %s \n str2 = %s", str1, str2);

 getch();

}
```

Output:→

```
Enter the first String:
mayank

Enter the second String:
rana
Str1 = rana
Str2 = rana
```

Date / /

★ Strcat( ) :→

This function is used for concatenation If first string is "xyz" hence second string is "abc".

After using this function, the first string becomes "xyz abc".

Strcat(Str1, Str2)

The null character from the first string is removed and the second string is added at the end of first string.

The second string remains unaffected.

Q18 :- W.A.P to understand the work of Strcat():

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
  void main()
{
    clrscr();
char str1[10], str2[10];
printf("Enter the first string: \n");
scanf("%s", str1);

printf("Enter the second String:\n");
scanf("%s", str2);

strcat(str1, str2);

printf("str1 = %s\n str2 = %s", str1, str2);
```

```
    printf("\n\n");

    getch();
}
```

Output :→

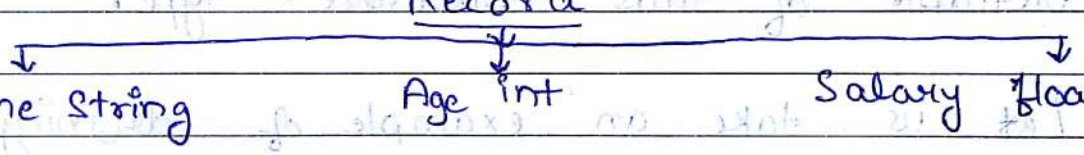
```
Enter the first String:
mayank
Enter the second String:
rana
Str 1 = mayankrana
Str 2 = rana
```

UNIT :→ 4

★ Structure :→

Array is a collection of same type of elements, but In many real life applications, we may need to group different types of logically related data.

For ex:→ If we want to create a record of a person that contains name, age, sallery of that person then can't use array because all the three data elements are of different types.



To store these related things of different datatypes we can use data structure which is capable of storing hetrogenious data.

★ Defining a Structure :→

Create a template or format that describe the characteristic of its member. All the variables that would be declare of this structure type.

Date / /

The General Syntax :- The general syntax of structure definiation is :-

```
Struct tagname
{
    datatype member1;
    datatype member2;
    ---------
    ---------
    datatype membern;
};
```

Here Struct is a keyword which tells the compiler that Structure is being define, member1 member 2 and soon member n, are known as members of the structure. And are declare inside curly braces {}.

Tagname is name of the structure and it is used forther in the program to declare variable of this structure type.

Let us take an example of defining a structure template :→

```
Struct Student
{
    char name[20];
    int age;
    float marks;
};
```

Date / /

★ Declaring Structure Variable :→

By defining a structure we have only created a format. The actual use of structures will be when we declare variables based on this format. We can declare structure variable in two ways :→

1. Structure Defination.
2. Using the Structure tag.

① With Structure Defination :-

```
Struct Student {
        char name[20];
        int Roll no;
        float marks;
    } Stu1, Stu2, Stu3;
```

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 Roll no;
        float marks;
    } Stu1, Stu2, Stu3;
```

If we declare variables in this way, then we will not able to declare other variable of this structure type anywhere in the program.

② Using the Structure tag :-

Date / /

② Using the Structure tag :-

```
Struct Student {
        char name[20];
        int Roll no;
        float marks;
        };
Struct Student Stu1, Stu2, Stu3;
```

Here, Stu1, Stu2, Stu3 are stomructure variable that are declare using the structure tag student.

Initialization of Structure Variable :→

The syntax of initializing structure variable is similar to that of arrays.
All the values are given in curly braces {} and the number order and type of these values should be same as in the structure template definition.

```
Struct Student {
        char name[20];
        int roll no;
        float marks;
} Stu1 = {"John", 25, 65.5};
Struct Student Stu2 = {"Sinha", 26, 76};
```

Here value of members of Stu1 will be "John" for name, 25 for roll no; 65.5 for marks.
The value of number of Stu2 will be "sinha" for name, 26 for roll no; 76 for marks.

Date / /

☆ Accessing Members of Structure :→

For accessing any member of a structure variable we use dot (.) operator. which is also known as membership operator.

The format or accessing a structure member

Structvariable.member

Q:→ W.A.P to accept the information of your teacher is the following Details (field):→ Nam, department qualification, designation ?

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
  void main()
struct str {
    char name[20];
    char department[20];
    char qualification[20];
    char designation[20];
    };

    void main()
    {
    clrscr();
    str x;

printf("Enter the Name:");
scanf("%s", x.name);

printf("\n Enter the Department:");
scanf("%s", x.department);
```

Date / /

```
printf("\n Enter Qualification: ");
scanf("%s", x.qualification);

printf("\n Enter the Designation: ");
scanf("%s", x.designation);

printf("\n Name is: %s", x.name);
printf("\n Department is: %s", x.department);
printf("\n Qualification is: %s", x.qualification);
printf("\n Designation is: %s", x.designation);

getch();
}
```

Output:- Enter the Name: Mayank

Enter the Department: BCA

Enter the Qualification: 12th

Enter the Designation: DBGI

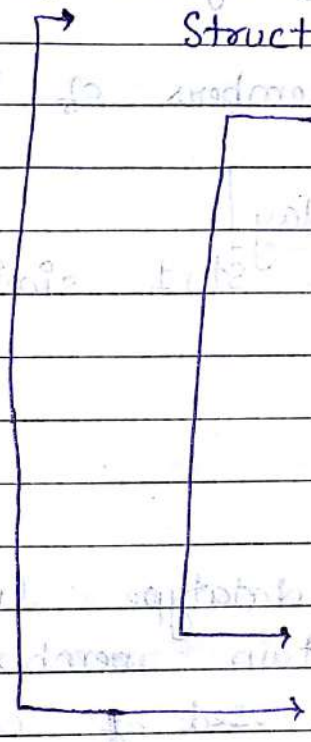
Name is: Mayank
Department is: BCA
Qualification is: 12th
Designation is: DBGI

Date / /

Nested Structure (Structure within Structure):-

The members of a structure can be of any datatype including another structure type, we can include a structure within another structure.

The structure variable can be member of another structure this is called nesting of the structure,



```
Struct tag 1 {
        member 1;
        member 2;
        struct tag 2 {
                member 1';
                member 2';
                ------
                member m;
        } var 1;
        ------
        member n;
} var 2';
```

For accessing member 1 of inner structure, we will write, var 2 . var 1 . member 1 .

Here is an example of nested structure :→

```
struct student 1 {
        char name;
        int Roll no.;
        Struct date {
                int day;
                int month;
                int year;
        } birth date;
```

float marks;
} stu1, stu2;

Here we have define a structure date inside the structure (student). This structure date has three members "day, month, year". Birth date is a variable of type struct data.

We can access the members of inner structure as,

stu1.birthdate.day

Day of birth date of stu1 similarly month similarly year.

v.Imp ☆ Union :→

Union is a drived datatype like structure and it can also contain members of different datatype. The syntax used of definition of an Union, declaration of union variable and accessing members is similar to that used in structures, but here keyword "union" is used instead of "struct".

The main differente b/w union & structure is in the way memory is allocated for the members.

In the structure, each member has its own memory location whereas members of unions share the same memory location.

When a variable of type union is declared, compiler allocates sufficient memory

Date / /

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 syntax of defination of a union is :-

```
union union_name {
        datatype member1;
        datatype member2;
        _ _ _ _ _ _ _ _ _
        _ _ _ _ _ _ _ _ _
    } variable_name;
```

This can also be declare as,

```
union union_name : variable_name;
```

Diff^{r} b/w Structure and Union :→

```
Struct student {
    int roll;
    char name[10];
    } stu;
```

roll → 2 bytes, name → 10 bytes

location → 2156 | 2158

Total 12 bytes.

```
union student {
    int roll;
    char name[10];
    } stu;
```

roll → 2 bytes, name → 10 bytes

Total → 10 bytes

Because it is greater than roll no[2 bytes]

Difference ⇒ Union = 10 bytes
Structure = 12 bytes

Q:→ Write a program to compare the memory allocated for a union and structure variable?

Date / /

```
#include<stdio.h>
#include<conio.h>
 struct stag{
  int i;
  char c;
  float f;
  };

 union utag{
  int i;
  char c;
  float f;
  };

 void main()
  {
   clrscr();
  union utag uvar;
   struct stag svar;
printf("Size of svar %d\n", sizeof(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("Size of uvar %d\n", sizeof(uvar));
printf("Address of uvar %u\n", &uvar);
printf("Address of member %u, %u, %u\n", &uvar.i
                    , &uvar.c, &uvar.f);
  getch();
  }
```

UNIT :→ 5

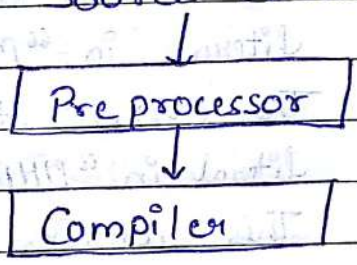
Date 6 / 4 / 18

☆ Introduction to C Preprocessor :→

'C' has a special feature of preprocessor which makes it different from other high level language that don't have his type of fecility.
Some advantage of using preprocessor are :→

1. Readability of the program is increase.
2. Program modifications eassy.
3. Makes the program portable and efficient.

We know that the code will write is translated into object code by the compiler but before being compile, The code is pass to the 'C' preprocessor. The 'C' preprocessor scan the whole source code and modify it, which is then given to the compiler.



The line starting with the symbol # are known as preprocessor directives. His some features of preprocessor directries are :→

1. Each preprocessor drictive start with a # symbol.
2. There can be only one driective on a line.
3. There is no semi-colon (;) at the end of directive.
4. The preprocessor driective can be plane anywhere

Date / /

in a program (inside or outside functions) But they are ~~program~~ usually written at the begining of the program.

☆ Derective & Description :→

Sr. No.	Directive	Discription
01	#define	Substitute a preprocessor macros
02	#include:	Insert a particular hyder from another file
03	#undef	Undefine a processor macros
04	#if	Test if a compile time condition is true
05	#else	The alternative for #if
06	#endef	end preprocessor conditioned

* Predefined Macros :→

Sr. No.	Macros	Discription
01	-DATE-	The current data as a character literal in "MMDD YY" format.
02	-TIME-	The current time as a character literal in "HHMMSS" format
03	-FILE-	This contains the current file name as a string literal.
04	-LINE-	This contains the current line numbers as the decimal constant.
05	-STDC-	Defined as 1, when the compiler compiles with the ANSI Standard.

Let us try the following example :-

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

Date / /

```
printf("File :%s \n", __FILE__);
printf("Date :%s \n", __DATE__);
printf("Time :%s \n", __TIME__);
printf("Line :%d \n", __LINE__);
printf("ANSI :%d \n", __STDC__);
```

When the above code in a file test.c is compile and executed it produce the following result.

Output :→ File : Test.c
Date : Apr. 10 2018
Time : 10:20:15
Line : 8
ANSI : 1

☆ Preprocessor Operator :→

① Macro Continuation (\) Operator :→ This Operator is generally used to continue a macro that is too long for a single line.

② STRINGIZE (#) Operator :→ It is use with in a macro difination, converts a macro parameter into a string constant.

③ TOKEN Passing (##) Operator :→ It is use with in a macro definition & combine two arguements it permits two separate token in the macro definition to be join into a single token.

④ Including files :→ The preprocessor directive #include is use to include a file into the saurce code the

Date / /

file name should be within (< >) brackets or (" ") double quotes.

The Syntax is :→ #include <file name>
#include "file name"

★ Bitwise Operator :→

① Bitwise AND Operator (&) :→ The Output of bitwise AND is one if the corresponding bits of two operants is one (1). If either bit of an operant is 0 ; the result of corresponding bit is evaluated to 0 (zero).

Let us suppose bitwise and operation of two integer 12 and 25.

Ex:→ 12 — 0 0 0 0 1 1 0 0 (in Binary)
25 — 0 0 0 1 1 0 0 1 (in Binary)

Bitwise operation of 12 & 25 :-

```
  0 0 0 0 1 1 0 0
& 0 0 0 1 1 0 0 1
-----------------
  0 0 0 1 1 0 0 0   = 8 (in Decimal)
```

Example of Program :→

```
#include<stdio.h>
#include<conio.h>
  void main()
  {
    int a=12, b=25;
  printf("Output = %d", a&b);
    return 0;
    getch();
  }
```

Date / /

Output :→ 8

② Bitwise OR Operator :→ The Output of bitwise OR is 1 (one), if atleast one corresponding bits of two operants is 1. In C programming bitwise OR Operator is denoted by bileline (|).

12 - 0 0 0 0 1 1 0 0 (in Binary)
25 - 0 0 0 1 1 0 0 1 (in Binary)

Bitwise OR Operation of 12 and 25

```
  0 0 0 0 1 1 0 0
  0 0 0 1 1 0 0 1
  ---------------
  0 0 0 1 1 1 0 1  = 29 (in Decimal)
```

Program Example :→

```
#include<stdio.h>
#include<conio.h>
  void main()
 {
  clrscr();
 int a=12, b=25;
printf("Output = %d", a|b);
  getch();
 }
```

Output :→ 29

③ Bitwise Compliment Operator (~) :→ Bitwise compliment Operator is an unary operator (work only one operant) its change 1 to 0 & 0 to 1. It is denoted by (~).

35 = 100011 (in Binary)

Date / /

Bitwise Compliment Operation of 35.

(~) 100011 Complement 011100

IInd Complement +1

011101

(4) Bitwise XOR (Exclusive OR) Operator (^) :→

The result of Bitwise XOR operator is one if the corresponding bit of two operants are opposite it is denoted by (^) sign.

12 - 00001100 (in Binary)

25 - 00011001 (in Binary)

Bitwise XOR Operation of 12 and 25

00001100

^ 00011001

00010101 = 21 (in decimal)

Program Example :→

```
#include <stdio.h>
#include<conio.h>
void main()
{
clrscr();
int a=12, b=25;

printf("Output = %d", a^b);
getch();
}
```

Output :→ 21