UNIT 4

Module – 4: (Arrays & Basic Algorithms)

- Arrays: Array notation and representation
- Manipulating array elements
- Using Multi dimensional arrays.
- Character arrays and strings
- Structure, union, enumerated data types
- Array of structures
- Passing arrays to functions.

Basic Algorithms:

- Searching & Basic Sorting Algorithms (Bubble, Insertion and Selection)
- Finding roots of equations.

Short Questions & Answers

Ques 1. How one dimensional array are initialized? Explain.

Ans : An array elements can be initialized like an ordinary variables when declared. The general syntax is as following : data_type array_name [size] = { list of values separated by comma};

Where [ ] is known as subscript operator.

Example: int arr[5] = { 10, 34, 21,76, 20} ;

float f[4] ={ 0.0 , 1,24, 5.63 ,3.6 };

int a [ ] = { 10 , 30 ,15, 8 , 40 , 60, 50 } ;

char ch[6] = { 'A' , '8' , 'B' , ' &' , 'd', '\0 '};

(i) Array elements contain garbage values if no values are given.
(ii) If number of elements are not known at compile time, then C does not allows to declare that array.
(iii) When all the elements are listed when declaring an array, size is optional.
(iv) Total elements are from 0 to size-1. Size should be a compile time constant expression of integer type.

Ques 2.What are the different ways to store the values in arrays?

Ans : (a) Initialization of array during declaration (b) Input values for the elements from the keyboard.
(c) Assign values to individual elements.

Examples: **(a) Initialization of array during declaration**

int marks [5] { 90, 65, 78, 80 , 47}; // initialization at the time of declaration.

90	65	78	80	47
[0]	[1]	[2]	[3]	[4]

When values are initialized in a 1-D array they are stored consecutively in memory, starting from index 0 to size-1 i.e. in above example from 0 to 4 , because size of array is 5.***If we declare like this :***

int marks [ ] = { 2 , 4, 10}; We have not mentioned the size inside subscript operator, So compiler will allocate space automatically equal to number of elements inside { } braces.

(b) Input values for the elements from the keyboard.

```
int i , marks [15] ;
for ( i=0 , i< 10 ; i++)
scanf ( " %d" , & marks [i] ) ;
```

(c) Assigning Values to individual Elements : This can be done using assignment operator. Any value that matches to the data type of an array can be assigned to the individual array element.

Example : marks [3] = 45 ; // 45 is assigned to 4th index of 1-D array.

Ques 3. Mention some invalid declarations in an array.

Ans : (a) int arr [j] ; // j is a variable not a constant.

(b) int arr [4.7] ; // Size can not be a float value.

(c) int arr [-1] ;

int arr [0] // Size must be >= 1

Ques 4. Define the pointer. How it is declared and initialized ?

Ans : Pointer : A **pointer** is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before using it to store any variable address. The general form of a pointer variable declaration is −

Data type *var-name;

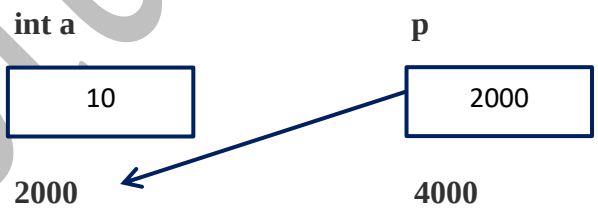
Here, ***data type is the pointer's base type***; it must be a valid C data type and **var-name** is the name of the pointer variable. The asterisk * used to declare a pointer is the same asterisk used for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Valid pointer declarations −

```
int     *ip;        /* pointer to an integer */
double *dp;     /* pointer to a double */
float  *fp;       /* pointer to a float */
 char   *ch       /* pointer to a character */
```

Initialization: **data type * ptr_var = & var name 2 ;** // & is known as address of operator

Example : // * is called value at address or indirection operator

```
int a = 10;
int *p = &a ;   // pointer variable p holds the value of variable a.
```



(i) 2000 is the address of variable a , and pointer p holds this address as a value. Pointer variable p has its own address 4000. 10 is the value stored at location a.

Ques 5. WAP in C to print the values and address with the help of pointers.

Ans :

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

```
Void main()
{
   int x  ;
   int *p;
   x = 10 ; p = &x ;
  printf( "Value of x = %d \n" , x) ;.
  printf(" Variable x has address = %u \n" , &x ) ;

}
```

Ques 6. How are character arrays declared and initialized ?

Ans : Character arrays are also known as strings. Strings are actually one-dimensional array of characters terminated by a **null** character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a **null**.

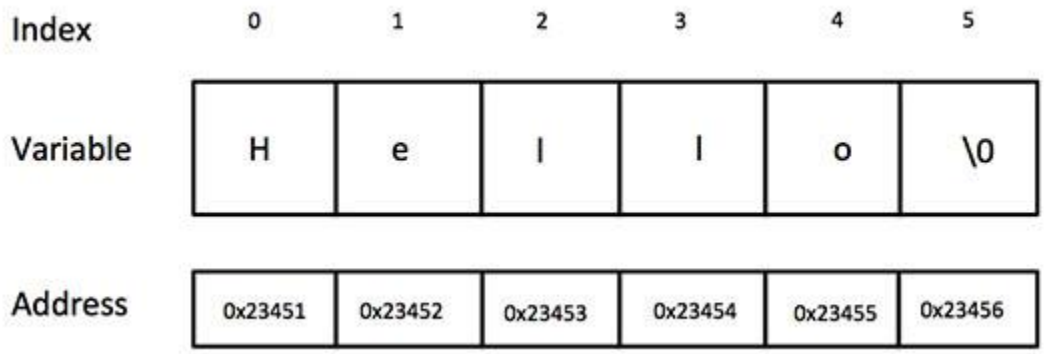
The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

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

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

char greeting[] = "Hello";

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

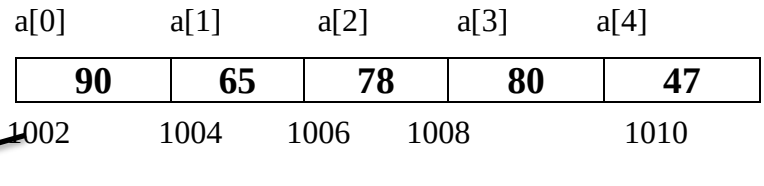


Actually, you do not place the *null* character at the end of a string constant. The C compiler automatically places the '\0' at the end of the string when it initializes the array.

Ques 7. What is the base address of an array?

Ans : Base address of an array refers to the 1st element of a 1-D array at 0th Index position. If we only write the name of an array then also it refers to the base address.

E.g : int a [5] = { 1 , 4 , 7, 9 , 10 };

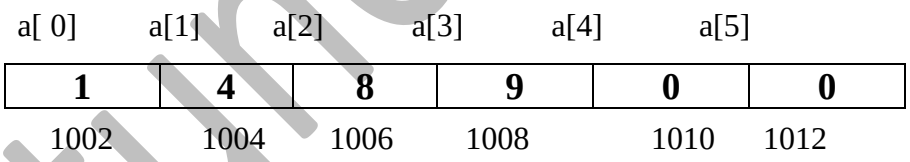


Here 1002 is the base address of array a. Since it is an integer array, so addresses of each elements will be at the difference of 2 bytes. While displaying the elements of array, next adjacent address is automatically called when its previous element is displayed. Using pointers we can access the individual elements.
Increment / decrement of pointers lead to increment /decrement of addresses based on the data type of pointers.

Ques 8. What is the output of following code :

```
void main()
{
int a[6] = { 1 , 4, 8 , 9 } ;
 int j ;
 printf( " Elements of array are : \n ") ;
  for ( j=0 ; j < 6 ; j ++ )
    {
       printf ( " %d " , a [i] ) ;
     }
  getch() ;
}
```

Ans : **Output of above code is : 1 4 8 9 0 0.**



If number of elements stored in an array are less than the maximum size of array , then remaining elements will take 0 value for remaining index. Since array a is initialized with only four elements, and max size is 6 Therefore elements at a[4] & a[5] will take values 0 , 0 by default.

Ques 9. Explain the following string manipulation functions :
(i) gets() (ii) puts() (iii) sprint() (iv strcmp()

Ans : (i) gets() : This is a library function . Definition is stored in string.h header file. This function reads the string as **gets(str)** , where str is the name of character array str.It takes the starting address of an string which will hold the input.This function does not terminates if a blank space occurs between two strings as in scanf().

(ii) **puts()** : The string can be displayed by writing puts(str). This function overcomes the drawback of printf. The puts() function writes a line of output on the screen. It terminates the line by '\n' .

(iii) **sprint() :** This is formatted output is written to memory area rather than directly on the output screen. Syntax is as following :

sprint(char * buffer , const char* format string [arg 1,arg2 ….arg n]) ;

E.g :
```
void main()
{
  char buf [100] ;
   int num = 10 ;
    sprint( buf , " num = %d " , num ) ;
}
```

(iv) **strcmp() :** This function compares the string pointed to by str1 with string pointed by str2.

Syntax strcmp(const char *str1 , const char str2) ;

Function returns zero if the strings are equal. Otherwise it returns a value less than zero or greater than zero if str1 is less than or greater than str2 respectively.

Ques 9. How are 2-Dimensional arrays declared and initialized.

Ans : The 2-D arrays are also known as matrix which are created in row and column form.

Syntax to declare: data_type array_name [i] [j] ; Where two subscripts i and j are rows and columns of a 2-D array. Subscripts i and j must be set at compile time.

E.g : int a [2] [3] ; // 2-D array with 2 rows and 3 columns.

Initialization of 2-D array : int a [2] [3] = { 0, 2, 3, 10, 30, 20 } ;

Matrix can be arranged in Row major and column major forms. In Row major form firstly elements of 1st row , then 2nd row …so on are stored . In column major form 1st column , then 2nd column…son on are stored.

Row major form :

0	2	3
10	20	30

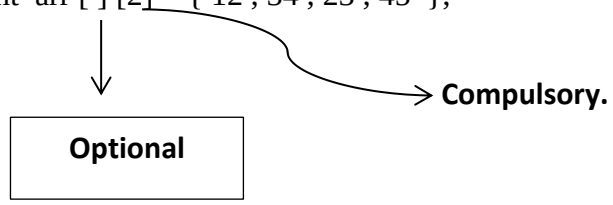
OR int a[2] [3] = { { 0,2,3} , { 10, 20, 30}} ;

Column Major form :

0	3	30
2	10	20

While initializing a 2-D array , it is necessary to mention 2nd dimension(number of columns), whereas 1st dimension is optional.

E.g : int arr [ ] [2] = { 12 , 34 , 23 , 45 };



Long Questions & Answers

Ques 10. What is the relationship between arrays and pointers? Explain with an example. Give the advantages and limitations of array.

Ans : The concept of array is very much dependent on pointers. Elements of array occupy contiguous memory locations depending on the data type of elements stored. Array location is a form of a pointer notation. The name of the array is the starting/base address of an array.

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
1	**4**	**8**	**9**	**0**	**0**
1002	1004	1006	1008	1010	1012

int *ptr ;

ptr = & a [0] ; // ptr is a pointer variable of integer type , that holds the address of 1st element a[0]

if array stores elements with float data type , then addresses will change an pointer to 1st element will be of float type . E.g : float * fptr ;

fptr = & b [0] ;

b[0]	b[1]	b[2]	b[3]	b[4]	b[5]
1.82	**4.67**	**8.00**	**10.23**	**3.67**	**92.78**
1002	1006	1010	1014	1016	1020

Now each element is stored with the difference of four bytes because of floating data type. Similarly in character array difference b/w each element will be of 1 byte.

Use of pointers : We can access the values of array elements using pointers as mentioned below:

(i) Expression a [i] is equivalent to write * (a + i). If a is the name of array, then compiler implicitly takes a = &a[0] . To print the value of third element of the array , we can use the expression *(a + 2). Because a [i] = * (a + i).

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
11	**10**	**80**	**49**	**12**	**34**
1002	1004	1006	1008	1010	1012

Example :
```
# include <stdio.h>
# include <conio.h>
void main ()
{
int a [6] = {11 , 10, 80, 49 , 12, 34 } ;
 printf( " First element is at address %u " , a) ;
 printf( " Third  element is at address %u " , a + 2) ;
printf( " Last  element is at address %u " , a + 5 ) ;
 getch();
}
```

Advantages of Arrays

(a) The direct indexing support is biggest advantage. Time required to read any element in an array of any dimension is almost the same irrespective of memory location.

Limitations of arrays

(a) The memory in array is allocated at compile time.
(b) Arrays are static in nature. The size of an array can not be changed (expanded/squeezed) at run time.
(c) Size of an array has to be kept big enough to occupy worst case .Memory usage is inefficient.

Ques 12. (i) What is referencing and dereferencing in pointers. Give an example.
(ii)What are the rules for pointers and Pointer arithmetic?

Ans : Referencing in pointers : In this a pointer variable is used to refer the address of an object . use of reference ***operator (& , ampersand), also called address of operator***. This is a unary operator on left side of operands. The operand should be a variable of arithmetic type or pointer type.

E.g :
```
float f = 12.5 ;
float * fptr ;
fptr = &f ;
```

Dereferencing a pointer : When we want to access the value stored at a particular address in a variable , then we use dereferencing with the help of pointers. This operator is also known as ***value at address operator , (* , asterisk)*** . Operand of * operator must be a pointer type variable.. This is a unary operator.

E.g : If we write *(p) or *(&a) = value at address of variable a.

```
// Referencing & Dereferencing of pointers
void main()
{
   int a = 12;
    int *i = &a;
   printf (" Value of a is = %d \n " , a) ;
   printf ( " Value by dereferencing a = %d \n " , *i) ;
   printf ( " Value of a  = %d \n" , *(&a))
   printf ( " Address of a =  %u " , & a) ;.
}
```

Rules of pointers

(a) A pointer can be assigned /initialized with the address of a variable. A pointer can't hold a non address value , so it can only be initialized with addresses.

(b) A pointer to a specific data type variable /object can't point to an object of another data type.
E.g : Following is an invalid assignment :

```
void main()
{
  int val = 10 ;
  float * ptr = & val ;
  printf ( " value of variable = %d \n" , val) ;
  printf ( " Pointer ptr holds the address = %u \n " , ptr) ;
 }
```

(c) Increment and decrement operator can be applied to pointer variables which will change the addresses Accordingly.
E.g : float * fptr ;
fptr = ptr ++ // post increment and initially , ptr = 2000 (address).
So address will be incremented by 4 bytes because of float data type .So fptr = 2004.But ptr will remain 2000 because it is post increment. Similarly , If we perform ptr - - , then new address will be 1996.

(d) Subtraction: Subtraction between two pointer variables gives the difference in number of bytes , (subscripts of two array elements)depending on data type of variables. E,g : Consider following float array b

b[0]	b[1]	b[2]	b[3]	b[4]	b[5]
1.82	**4.67**	**8.00**	**10.23**	**3.67**	**92.78**

1002 1006 1010 1014 1016 1020

Let pointer p1 = 1002 and pointer p2 = 1010. So p1 – p2 = 2 , i.e difference of two subscripts , where each subscript element is of 4 bytes (Array is float type)). So , 4 x 2 = 8 bytes.

Subtraction of two pointers is useful only if both the pointers hold the address of variables in same array.

Illegal pointer operations

1. Two pointer variables can't be added.
2. Only integer constants can be added to a pointer variable, but float /double constants can't be added.
   E.g : int * a ;
   int * p ;
   p = a + 3 ; // this is valid , but p = a + 3.67 is invalid.
3. Multiplication and division can't be performed on pointer variables.
4. Bit wise operations are invalid.

Programs on 1-D Arrays

Ques 13. WAP in C to read and display n numbers Using an array.	**Ques 14. WAP in C Find the smallest number of a 1-D array .**

#include<stdio.h>
#include<conio.h>
void main()
{
int i=0 , n , arr [20] ;
clrscr();
printf (" Enter the number of elements : \n") ;
	#include < stdio.h>
	#include <conio.h>
	void main()
	{
	int arr [100] , min, i , n, s , pos ;
	float avg_marks = 0.0 ;
	clrscr() ;

```
scanf( " %d " , &n) ;
printf( " \n Enter the elements \n") ;
for ( i=0 ; i<n ; i++)
 {
   printf( " \n arr [ %d] =  " , i );
  scanf( " %d" , &arr[ i]) ;
}
printf( " \n The array elements are \n ") ;
For ( i=0 ; I <n ; I ++)
 printf ( " arr[%d]  = %d \t ", i , arr[i]) ;
getch();
```

```
  printf ( " Number of elements in array\n") ;
  scanf( " %d " , &n) ;
printf ( " Enter elements of  array\n") ;
for( i = 0; i < n ; i++)
  {
 scanf ( " %d" , &a[ i] ) ;
  }
Small = a[0] ;
Pos =0 ;
for (i=1 ; i<n ; i++)
 { If (small > a[i])
      {
              Small = a[ I ] ;
             Pos =I ;
      }
 }
   printf ( " Smallest number of array is %d at
position %d \n " , small , pos + i);

getch () ;
}
```

Ques 15.WAP in C to enter number of digits. Form a number using these digits.	**Ques 16. WAP in C to calculate the sum and average of marks , secured by students using array named marks[ ] .Number of students must be read through keyboard.**

#include<stdio.h>
#include<conio.h>
#include<math.h>
void main
{
  int num =0, digit [10], n , i ;
clrscr() ;
	#include < stdio.h>
	#include <conio.h>
	void main()
	{
	  int marks[200] , i , sum = 0, studs ;
	  float avg_marks = 0.0 ;
	  clrscr() ;

```
printf( "Enter the number of digits for number \n" ) ;
scanf ( " %d " , &n) ;
for( i=0 ; i < n; i ++)
  {
    printf ( " \n Enter the %d digit : " , i ) ;
    scanf( " %d" , &digit [i]) ;
 }
i=0 ;
while ( i < n)
{
  num = num + digit [ i] + pow (10,i) ;
  i++;
}
printf( " \n The number = %d " , num) ;
getch();
}
```

```
   printf ( " Enter the number of students \n");
  scanf ( " %d" , & studs) ;
printf ( " Enter the marks of all students \n") ;
for( i = 0; i < studs ; i++)
  {
  printf ( "Enter the marks of students %d
\n\t",marks[i]);
 scanf ( " %d" , & marks [i]) ; i)
  }
for (i =0 ; i < studs ; i++ )
 {  sum = sum + marks [i] ; }
   avg_marks = (float) sum / studs ;
  printf ( " Sum of marks = %d" , sum);
  printf( "\n Average marks of %d students =
%f ", studs , avg_marks ) ;
getch () ;
}
```

Ques 17. WAP in C to print the elements of an array using pointers.

```
#include<stdio.h>
#include<conio.h>
{
  int b [3] = { 10 , 20 , 30 } ;
 printf( "Elements are %d  %d  %d \n", b[0], b[1] ,
b[2]) ;
printf(" Elements are %d  %d  %d \n" , *(b +0),
*(b+1) , *(b + 2)) ;
getch();
}
```

Ques 18.WAP in C to check whether an array of integers contain a duplicate number or not.

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int i , n ,j ,  arr [20] , flag=0  ;
 clrscr();
printf ( " Enter the number  of elements : \n" ) ;
scanf( " %d " , &n) ;
printf( " \n Enter the elements \n") ;
for ( i=0 ; i<n ; i++)
 {
  printf( " \n arr [ %d] =  " , i );
 scanf( " %d" , &arr[ i]) ;
}
for ( i=0 ; i < n ; i ++)
 {
```

	for(j= i +1 ; j < n ; j++)
	{
	if (arr [i] = = arr [j] && i!= j)
	{
	flag =1;
	printf (" Duplicate numbers found at location %d and %d ", i , j) ;
	}
	}
	if (flag=0)
	Printf(" \n No duplicate numbers found ")
	getch(); }

Ques 19. What operations can be performed on 1-D arrays? Explain linear searching. WAP to demonstrate linear searching in an array.

Ans : Operations on arrays :

(a) Searching an element in an array.
(b) Sorting an array.
(c) Finding smallest and largest number in the given array.
(d) Reading and printing the elements of an array.
(e) Finding Kth Largest element in an array.
(f) Comparison of two arrays.
(g) Deletion of an element at specific position in an array.

SEARCHING: The technique for finding the desired data element that has been stored in an array is known as searching .After searching is completed , there may be two cases w.r.t data element : Case 1: Search is successful , if it locates an element at required position
Case 2 : Search is unsuccessful , if it fails to locate an element .

Types of Searching: (A) Linear /Sequential search (B) Binary Search.

LINEAR SEARCH : In this method , array is searched for a particular data item by comparing every element of the array one by one until a match is found .An element to be searched is generally called KEY element is used to compare with another values. Linear search is generally applied to unordered list (unsorted) of elements.

Example :

```
#include<stdio.h>
#include<conio.h>
void main()
{ int arr[ 10 ] = { 15, 20 , 30 , 9 , 14 , 56 , 40 , 100 , 67, 7 };
  int i , num ;
printf( " Enter the key element to be searched in array\n");
scanf ( " %d" , & num);
for(I = 0 ; i <10 ; i ++ )
   { if (num = = arr [ i ])
      {    printf ( " %d number is found \n ", num) ;
           break ;
       }
else
    {
       printf( " %d number not found " , num ) ;
     } }
getch () ; } // End of main()
```

Ques 20 : What is sorting ? WAP in C to sort the elements of an array using bubble sort.

Ans : Term sorting means arranging the elements of an array in some relevant order which may be either ascending / descending. If A is an array then , the elements of an array arranged in ascending order will be such that : A[0] < A[1] < A[2] < ……A[N-1]. Example : if we have elements of array as :

A[ ] = { 21 , 34 ,11, 9 , 1 , 0 , 22} ; Then the sorted array in ascending order will be as given below :
A[ ] = { 0 , 1 , 9 , 11 , 21 , 22 , 34}.Efficient sorting algorithms are used to optimize the use of other Algorithms like search and merge algorithms which require sorted lists to work properly.
Two types of Sorting exist : (i)Internal Sorting (ii) External Sorting.

Internal sorting deals with sorting the data stored in computer's memory.***External sorting*** deals with the data stored in files . It is used when large data can not be stored in memory of computer.

Bubble Sort: In this method of sorting array elements is done by repeatedly moving the largest elements to the highest index position of the array(if using ascending order). In this consecutive adjacent pairs of elements are compared with each other. If element at lower index is greater than the element at higher index, the two elements interchange their positions. This process continues till the list if unsorted elements are finished.

Program of Bubble Sort Algorithm in C.

```
#include <stdio.h>
#include <conio.h>
void main ()
 {
    int i,j, a, n, number[30];
   clrscr();
   printf ("Enter the value of  N\n");
    scanf ("%d", &n);
     printf ("Enter the numbers \n");
     for (i=0; i<n; ++i)
     scanf ("%d",& number[i]) ;
       for (i=0; i<n ; ++i)
           {    for (j=i+1; j<n; ++j)
                  if (number[i] > number[j])
                     {
                        a= number[i];
                      number[i] = number[j];
                       number[j] = a; } }
```

```
 printf ("The numbers arranged in ascending
          order are given below\n");

for (i=0; i<n; ++i)
 printf ("%d",number[i]);
getch();
 }                    /* End of main() */
```

……

Output

```
Enter the value of N
5
Enter the numbers 80   20  67  10  45
The numbers arranged in ascending
order are given below
10    20   45    67   80
```

Ques 21. Write a C program to accept a matrix of order M x N and find the sum of each row and each column

```
#include <stdio.h>
 #include<conio.h>
 void main ()
{
   int m1[10][10];
   int i, j, m, n, sum=0;
   clrscr() ;
   printf ("Enter the order of the matrix\n");
   scanf ("%d %d", &m, &n);
printf ("Enter the co-efficients of the matrix\n");
     for (i=0; i<m ; ++i)
      {    for (j=0 ; j<n ; ++j)
             {  scanf ("%d", &m1[i][j]) ; }
      }

 for (i=0;i<m;++i)
      {   for (j=0;j<n;++j)
             {  sum = sum + m1[i][j] ; }
                printf ("Sum of the %d row is =
```

Ques 22. Write a C program to accept a matrix of order m x n and find its transpose .

```
#include <stdio.h>
 #include <conio.h>
 void main ()
{
   int mat [10][10];
   int i, j, m, n ;
   printf ("Enter the order of the matrix \n");
   scanf ("%d %d", &m ,&n);
   printf ("Enter the elements  of the matrix\n");
   for (i=0; i<m;++i)
    {
      for (j=0;j<n;++j)
       {
         scanf ("%d",& mat[i][j]);
       }
    }
printf ("The given matrix is \n");
 for (i=0;i<m;++i)      // Here outer loop is now for
counter variable i
   {
```

```
%d\n", i, sum) ;
                    sum = 0;   }

        for (j=0; j<n ;++j )
        {   for (i=0 ; i< m ; ++i)
                {   sum = sum+m1[i][j] ; }
                   printf ("Sum of the %d column is =
%d\n", j, sum);

        } getch () ;
}       /*End of main() */
/*-------------------------------------------------
Output
Enter the order of the matrix
3 3
Enter the co-efficients of the matrix
1 2 3
4 5 6
7 8 9
Sum of the 0 row is = 6
Sum of the 1 row is = 15
Sum of the 2 row is = 24
Sum of the 0 column is = 12
Sum of the 1 column is = 15
Sum of the 2 column is = 18
```

```
      for (j=0;j<n;++j)
        {
          printf (" %d", mat[i][j]);
        }
          printf ("\n");
 }
printf ("Transpose of matrix is \n");
for (j=0; j<n; ++j)                    // Here outer
loop is now for counter variable j
            {
                  for (i=0 ;i<m ;++i)
                  {
                        printf (" %d",mat[i][j]);
                  }
                  printf ("\n");
            }
      }     /* End of main() */

 ------------------------------------------------------
      Output
      Enter the order of the matrix
      2 2
      Enter the coefficients of the matrix
      3 -1
      6 0
      The given matrix is
       3 -1
       6 0
      Transpose of matrix is
       3 6
       -1 0
```

Ques 23. Program to Multiply Two 2-D arrays

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int a[10][10],b[10][10],c[10][10],i,j,k,r1,c1,r2,c2;
   int sum=0;
   clrscr();
printf("Enter number of rows and columns of first
            matrix \n");
   scanf("%d%d",&r1,&c1);
   printf("Enter number of rows and columns of sec
            ond matrix \n");
   scanf("%d%d",&r2,&c2);
printf("The First Matrix Is: \n");
                        //print the first matrix
  for(i=0; i<r1; i++)
    {
      for(j=0; j<c1; j++)
      printf(" %d ",a[i][j]);
     printf("\n");
    }
printf("The Second Matrix Is:\n");
                        // print the second matrix
for(i=0; i<r2; i++)
 {
   for(j=0; j<c2; j++)
```

```
  if(r2==c1)
  {  printf("\n Enter First Matrix:");

     for(i=0; i<r1; i++)
     {
          for(j=0; j<c1; j++)
          scanf("%d",&a[i][j]);
     }
 printf("\n Enter Second Matrix: ");
 for(i=0; i<r2; i++)
  {
    for(j=0; j<c2; j++)
     scanf("%d",&b[i][j]);
  }
```

OUTPUT:

```
Enter number of rows and columns of first matrix ( MAX 10)
3
3
Enter number of rows and columns of second matri x MAX 10)
3
3

 Enter First Matrix:2 2 2 2 2 2 2 2 2

 Enter Second Matrix: 3 3 3 3 3 3 3 3 3
The First Matrix Is:
 2  2  2
 2  2  2
 2  2  2
The Second Matrix Is:
 3  3  3
 3  3  3
 3  3  3
Multiplication of the Matrices:
18  18  18
18  18  18
18  18  18
```

```
   printf(" %d ",b[i][j]);
   printf("\n");
 }
printf("Multiplication of the Matrices:\n");

for(i=0; i<r1; i++)
 {
    for(j=0; j<c2; j++)
       {
          c[i][j]=0;
          for(k=0; k<r1; k++)
             c[i][j]+=a[i][k]*b[k][j];
          printf("%d  ",c[i][j]);
       }
       printf("\n");
    }

   }



else
   {
     printf("Matrix Multiplication is Not Possibl
e");
   }
   getch();
}
```

Ques 24.WAP to find the length of a string without using library function.

```
 #include<stdio.h>
#include<conio.h>
 void main()
{
  char string[50];
   int  i , length = 0;
  printf ("Enter a string\n");
  gets (string);
   for (i=0; string[i] != '\0'; i++)
       {     length++  ;
        }
   printf("The length of a string is the number of characters in it\n");
   printf("So, the length of %s =%d\n", string, length);
}
```

/*----------

Output
Enter a string
hello
The length of a string is the number of characters in it
So, the length of hello = 5

Ques 25. Program to compare two strings using user defined function. If strings are identical display "The Two Strings are Identical" otherwise the strings are different.

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int count1=0,count2=0,flag=0,i;
   char str1[10],str2[10];
   clrscr();
   puts("Enter a string:");
   gets(str1);
   puts("Enter another string:");
   gets(str2);
   /*Count the number of characters in str1*/
   while (str1[count1]!='\0')
          count1++;
   /*Count the number of characters in str2*/
   while (str2[count2]!='\0')
          count2++;
   i=0 ;
  while ( (i < count1) && (i < count2))
   {    if (str1[i] == str2[i])
            {   i++;
               continue ;
            }
        if (str1[i]<str2[i])
            {      flag = -1;
                   break;
             }
   if (flag==0)
      printf("Both strings are equal\n");
   if (flag==1)
      printf("String1 is greater than string2\n", str1, str2);
   if (flag == -1)
      printf("String1 is less than string2\n", str1, str2);
   getch(); }
```

/*----------

Output
Enter a string:
happy
Enter another string:
HAPPY
String1 is greater than string2

RUN2
Enter a string:
Hello
Enter another string:
Hello
Both strings are equal

RUN3
Enter a string:
gold

```
        if (str1[i] > str2[i])
            {   flag = 1;
                break;
            }
    }
```

```
Enter another string:
silver
String1 is less than string2
---------------------------------------*/
```

Ques 26. WAP in C to reverse a string input through the keyboard.

```
#include<stdio.h>
#include<conio.h>
int main()
{
   int i= -1;
   char str[100];
   char rev[100];
   char *strptr = str;
   char *revptr = rev;
  clrscr();
   printf("Enter the string:n");
   scanf("%s",str);
   while(*strptr)
   {
      strptr++;
      i++;
   }
   while(i >=0) {
      strptr--;
      *strptr = *revptr;
        revptr++;
      --i;
```

Ques 27. WAP in C to reverse a string using a user defined function.

```
#include<stdio.h>
#include<conio.h>
char* mystrrev(char*s)
int main()
{
    char str [ 20] ;
  puts(“ Enter a string”) ;
  gets(str) ;
  mystrrev (str) ;
puts(“After the reversal string is : ”) ;
 puts(str);
 getch();
}
char* mystrrev (cahr* s)
{
  int i=0 , j=0 ;
  char temp;
  while (s[i] !=’\0’)
    i++;
   i-- ;
 while (i > j)
 {
   temp = s[i] ;
```

```
    }
    printf("nn Reversed string is:%s", rev);
    return 0;
  getch();
}
```

```
   s[i] = s[j] ;
   s[j] = temp ;
  j++ , i-- ;
}
return s ;
}
```

Ques 28. Write short notes on the following concepts:
(i) Passing arrays to functions.
(ii)Enumerated data types.
(iii)Union.

Ans : (i) Passing Arrays to functions : When we need to pass an entire array to a function, we can pass the name of the array. Entire array is always passed by reference to the function. Following are the rules to pass 1-D array in a function :

Rule 1. The actual argument in the function call should only be the name of the array without any subscript, because name of an array represents base address.

Rule 2. Formal parameters in the function definition must be of array type or pointer type (i.e pointer to first element of the array.) If formal parameter is of array type , it will be implicitly converted to pointer type.

Rule 3. Parameter type in the function declaration should be of array type or pointer type.

Example :

```
void main()              // Calling function.
{
   int arr [5] = { 1 ,4 , 10 , 45 , 65 };
   func(arr) ;
}
void func (int arr [ 5])   // Called function
 {
  int i ;
```

Note : When an entire array is to be sent to the called function, the calling function just needs to pass the name of array.

In cases where called function does not makes any changes to the array, the array must be received as a constant array by the called function. It prevents any type of unnecessary modifications by called function to the array

for (i=0 ; i<5 ; i++) **printf(" %d " , arr [i]) ;** **}**	elements.

(ii) **Enumerated Data Types** : This is a user defined data type based on the basic integer type. Enumeration has a set of named integer constants. Each integer value is assigned an identifier, also known as enumeration constant, which can be used as a symbolic name to make program easy to be read.

- Keyword enum is used for this data type.
- Syntax : **enum enumeration_name { identifier1 , identifier 2, ……., identifier n}.** Enumeration name is optional.

  Example : enum COLORS { RED , BLUE , BLACK , GREEN , YELLOW ,PURPLE} ;

  Now COLORS is a new data type. COLORS is the name given to the set of constants. In case we do not assign any value to a constant , default value for the first one in list , RED has value of 0. Next constants will have sequential values after 0 i.e , 1, 2, 3…so on.

  RED = 0 ,BLUE = 1, Black =2 , GREEN = 3 , YELLOW= 4, PURPLE= 5.

  We can also initialize symbolic constants explicitly by specific integer values , which may not be in sequence.

  Example : enum COLORS { RED = 3 , BLUE = 7 , BLACK = 0 , GREEN = 5 ,
  YELLOW = 2, PURPLE = 10} ;

(iii) **Union:** This is a user defined data type. It is a collection of variables of different data types.

In Unions we can only store information in one field at any one time. It is like a chunk of memory used to store variables of different data types. When a new value is assigned to a field, the existing data is replaced with the new data.

Syntax of declaration: **union union-name**
{

```
    data_type var-name1;

    data_type  var-name2 ;
}
```

Ques 29. What are structures? What is the advantage of structures over array? Differentiate between Structures and Unions.

Ans : Structures are the user defined data types that is of heterogeneous nature , that means it can store information of different data types. Keyword struct is used to declare a structure.

Declaration of structure:

```
struct  structure _ name
{
   Data_type var-nmae1;
   Data_type var-nmae1
} ;
```

Example: if we define a structure for a student , then the Related information can be : rollno , name , course , fees. This structure can be declared as :

```
struct student
    {
       int rollno ;
      char name [ 20];
      char course [ 25] ;
      float fees ;
    } ;
```

Example : declaration of a structure named date.

```
struct date
{
   int day ;
   int month ;
   int year ;
}
```

Each variable name declared inside structure is called member of structure. Structure declaration does not consume any storage space. Example: , we can define a variable by writing :

struct student stud1 ;

E.g :

```
struct student
 {
    int rollno ;
    char name [ 20];
  char course [ 25] ;
  float fees ;
} stud1,stud2 ;
```

Here stud1 and stude2 are two variables of structure name student. Variables are separated by commas.

A separate memory is allocated to variables while declaring them.

Initialization of structures: A structure can be initialized in the same way as other data types

are initialized. Initializing a structure means assigning some constants to the members of the structure. If explicitly, then C automatically does that. For int and float members, values are initialized to zero and character and string members are initialized to '\0' by default. Initializers are enclosed in curly braces separated by commas.

Example:

```
struct student
{
  int rollno ;
  char name [ 20 ];
  char course [ 25] ;
  float fees ;
} stud1 = { 001 , Vishal , B.Tech , 76000 } ;
```

OR

```
struct student  stud1 = { 001 , Vishal , B.Tech , 76000 } ;
```

Accessing Members of a Structure.

Dot operator is used to access the values of members with the help of variables in structure**.**

Syntax : struct_var. member_name ;

Example :
Stud1.rollno = 001
Stud1.name= Vishal
Stud1.course = B.Tech
Stud1.fees = 76000

Advantage of Structure over Arrays : Structure is advantageous than arrays in the sense that it can store information of variables of different data types where as arrays have information of only same data types**.**

Difference between Structure & Union

	STRUCTURE	UNION
Keyword	The keyword **struct** is used to define a structure	The keyword **union** is used to define a union.
Size	When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is **greater than or equal to the sum of sizes of its members.**	when a variable is associated with a union, the compiler allocates the memory by considering the size of the largest memory. So, size of **union is equal to the size of largest member.**
Memory	Each member within a structure is assigned unique storage area of location.	Memory allocated is shared by individual members of union.
Value Altering	Altering the value of a member will not affect other members of the structure.	Altering the value of any of the member will alter other member values.
Accessing members	Individual member can be accessed at a time.	Only one member can be accessed at a time.
Initialization of Members	Several members of a structure can initialize at once.	Only the first member of a union can be initialized.

Ques 30.WAP in C to create a structure named account to hold information of account number , account name , balance amount. Display the values of each variable declared . Values must be entered through the keyboard.

```
#include<stdio.h>
#include<conio.h>
void main()
{
  struct account
    {
       int acc_no ;
       char acc_name[ 15 ] ;
        float bal ;
    } ;

 struct account a1, a2,a3 ;
```

Ques 31.WAP in C to create a structure named student to hold information of three students .Program must accept the roll number , name , marks obtained in three tests. Display the information of each student also and average marks obtained.

```
#include<stdio.h>
#include<conio.h>
void main()
{
  struct  student
     {
        int  rollno ;
        char  name[ 20 ] ;
        int m1, m2, m3 ;
        float avg ;
     } ; int i,total ;

struct student std[3] ; // Array of structures
printf(" Enter the information of three
students \n ") ;
for( i=0 ; i< 3; i++ )
```

```
printf(" Enter the account no , account name and
balance information \n");

// Reading the values from keyboard
scanf( " %d %s %f " , &a1.acc_no ,&a1.acc_name
        , & a1.bal ) ;
scanf( " %d %s %f " , &a2.acc_no ,&a2.acc_name
       , & a2.bal ) ;
scanf( " %d %s %f " , &a3.acc_no ,&a3.acc_name
        , & a3.bal ) ;

// Displaying the values
printf ( " \n %d %s %f " , a1.acc_no , a1.acc_name
          ,  a1.bal ) ;
printf ( " \n %d %s %f " , a2.acc_no ,a2. acc_name
          ,  a2.bal ) ;
printf ( " \n %d %s %f " , a3.acc_no , a3.acc_name
          , a3.bal ) ;

getch();
}               // End of main
```

```
{
  printf( "Enter the roll no. of student %d
            = " , i+1);

scanf(" %d " , &std[i].rollno ) ;

printf( "Enter the name of student %s
          = " , i+1 );
scanf("% s " , &std[i].name ) ;

printf( "Enter the marks1= " );
scanf("%d " , &std[i].m1 ) ;

printf( "Enter the marks2 = ");
 scanf ( "%d " , &std[i].m2);

printf( "Enter the marks3 = ");
 scanf ( "%d " , &std[i].m3);
}


for(i=0 ; i<3 ; i++)
{

total= std[i].m1 + std[i].m2
       + std[i].m3 ;

std[i].avg = total/3.0 ;
printf( " %d  %s  %d  %d %d  % f\n  ",

        std[i] .rollno,std[i].name,std[i].m1,
 std[i].m2,std[i].m3,std[i].avg) ;
}  // End of for loop

getch() ;
}          // End of main()
```

Ques 32 : How an array of structure is created ? Explain with the help of a program.

Ans : It is possible to create an array whose elements are of structure type. Such an array is called Array of Structures. If we consider a book example where we have to display information of book name , author name , number of pages and its price then it can be stored in a variable of type struct book. We can create two variable book1 , book2 and read and display the information about two books..

But in case if we want to access the information of suppose 100 books , then we will have to create 100 variables like book1 , book2, book3 ,…..book100 . Also we will have to read and display the information separately for each book. This will be a very long , difficult process ,and time consuming also. So array of structures will provides an easy way to store information of 100 books.

General Syntax :

```
struct  struct_name
{
    Data_type1    member_name1 ;
     Data_type2    member_name2 ;
     Data_type3   member_name3 ;
…….
 Data _type n  member_name n ;
} ;    struct    struct_name  struct_var [ index ];
```

/* Program of Array of Structure for Book information */

```
#include<stdio.h>
#include<conio.h>
void main
{
  struct book
      {

          char Name[30];
          char author[20] ;
          int pages ;
           float price ;
       };
           struct  book  b[ 100 ];        //array of structures.
           int i;
          for(i=0;i<100;i++)
          {
          printf("\n Enter details of %d Book ",i+1);

          printf("\n\t Enter Book Name  : ");
         scanf("%s",&b[i].name);

          printf("\n\t Enter Author Name  : ");
         scanf("%s",&b[i].author);
```

```
        printf("\n\t Enter No. of pages  : ");
        scanf("%d",&b[i].pages);

        printf("\n\t Enter Book Price  : ");
        scanf("%f",&b[i].price);
        }                                   // End of loop
        printf("\n Details of  all the  Books ");
        for(i=0;i< 100 ; i++)
        printf("\n%d\t %s\t%s\t%d \t %d ",b[i].name ,b[i].author,b[i].pages, b[i].price );
   getch();
}           // End of main()
```

[END of 4th UNIT]