MODULE 4

1. Write the syntax for declaring two dimensional arrays with suitable example.

A two dimensional array is declared as:

data_type array_name[row_size][column_size];

Therefore, a two dimensional m x n array is an array that contains m x n data elements and each element is accessed using two subscripts, i and j where i<=m and j<=n.

For example, if we want to store the marks obtained by 3 students in 5 different subjects, then we can declare a two-dimensional array as

int marks[3][5];

A two-dimensional array called 'marks' is declared that has m(3) rows and n(5) columns. The first element of the array is denoted by marks[0][0], second element as marks[0][1], and so on. Here, marks[0][0] stores the marks obtained by the first student in the first subject, marks[1][0] stores the marks obtained by the second student in the first subject, and so on.

2. Demonstrate the representation of 2D array in memory with a suitable example.

For example, if we want to store the marks obtained by 3 students in 5 different subjects, then we can declare a two-dimensional array as

int marks[3][5];

A two-dimensional array called 'marks' is declared that has m(3) rows and n(5) columns. The first element of the array is denoted by marks[0][0], second element as marks[0][1], and so on. Here, marks[0][0] stores the marks obtained by the first student in the first subject, marks[1][0] stores the marks obtained by the second student in the first subject, and so on.

The memory representation of above two dimensional array is shown below:

Row / Column	Column 0	Column 1	Column 2	Column 3	Column 4
Row 0	marks[0][0]	marks[0][1]	marks[0][2]	marks[0][3]	marks[0][4]
Row 1	marks[1][0]	marks[1][1]	marks[1][2]	marks[1][3]	marks[1][4]
Row 2	marks[2][0]	marks[2][1]	marks[2][2]	marks[2][3]	marks[2][4]

Figure: Memory representation of two dimensional array for int marks[3][5];

If we initialize a two-dimensional array 'marks' as:

int marks[2][3] = {90, 87, 78, 68, 62, 71};

Then the memory representation is shown:

Row / Column	Column 0	Column 1	Column 2
Row 0	90	87	78
Row 1	68	62	71

3. Explain how a single row can be passed into a 2D array with an example.

A row of a two-dimensional array can be passed by indexing the array name with the row number. When we send a single row of a two-dimensional array, then the called function receives a one-dimensional array. Figure illustrates how a single row of a two-dimensional array is passed to the called function.

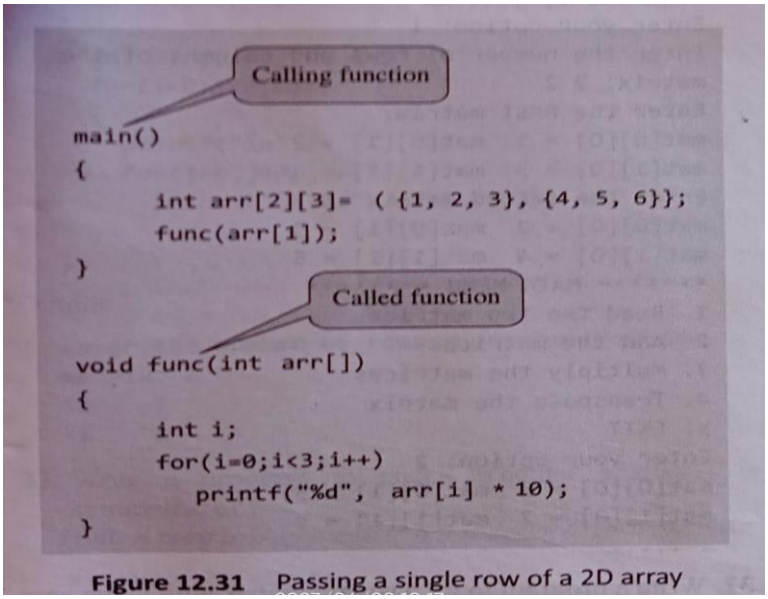

Figure 12.31 Passing a single row of a 2D array

4. With a neat diagram, Explain three dimensional array. Write a C program to read and display 2x2x2 array.

Three dimensional array:

A three dimensional m_1 x m_2 x m_3 array is a collection of m_1 * m_2 * m_3 elements. Figure shows a three dimensional array having three pages, four rows and 2 columns.

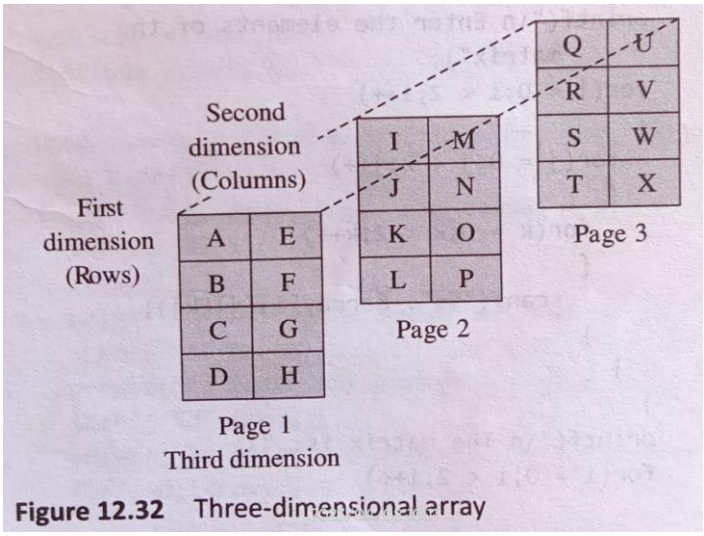

Figure 12.32 Three-dimensional array

Program:

```
#include <stdio.h>
int main ( )
{
int i, j, mat[3][3], t_mat [3][3];
printf ( "Enter the elements of the matrix: \n" );
for ( i=0; i<3; i++ )
{
for ( j=0; j<3; j++ )
{
scanf ( "%d", &mat[ i ][ j ] );
}       }
printf ( "The elements of the matrix are: \n " );
```

```
for ( i=0; i<3; i++)
{
printf("\n");
for ( j=0; j<3; j++ )
printf ( "\t %d", mat[ i ][ j ] );
}
for ( i=0; i<3; i++)
{
for ( j=0; j<3; j++ )
t_mat[ i ][ j ] = mat[ j ][ i ] ;
}
printf ( " The elements of the transposed matrix are: \n " );
for ( i=0; i<3; i++)
{
printf("\n");
for ( j=0; j<3; j++ )
printf ( "\t %d", t_mat[ i ][ j ] );
}
return 0;
}
```

5. What are strings? Mention reading strings & writing strings along with their syntax.

In C programming, a string is a sequence of characters terminated with a null character. This means that, after the last character, a null character ('\0') is stored to signify the end of the character array.

For example, char str[ ] = "Hello"; This will be stored as

H	e	l	l	o	\0

Reading Strings: If we declare a string by writing

char str [100];

Then str can be read by using three ways:

1. using scanf( ) function

2. using gets( ) function

3. using getchar( ), getch( ) or getche( ) function repeatedly

Stings can be read using scanf( ) by writing

scanf("%s", str);

The main drawback with this function is that the function terminates as soon as it finds a blank space. For example, if the user enters 'Hello World', then str will contain only 'Hello'. This is because the moment a blank space is encountered, the string is terminated by the scanf( ) function.

The field width can be specified to indicate the maximum number of characters that can be read in.

The string can be read by using gets( ) function. The string can be read by writing

gets(str);

gets( ) is a simple function that overcomes the drawbacks of the scanf() function. The gets( ) function takes the starting address of the string which will hold the input. The string inputted using gets( ) is automatically terminated with a null character.

The string can also be read by calling the getchar( ) function repeatedly to read a sequence of single characters (unless a terminating character is entered) and simultaneously storing it in a character array.

Writing Strings: Strings can be displayed on the screen using three ways:

1. using printf( ) function

2. using puts( ) function

3. using putchar( ) function repeatedly

A string can be displayed using printf( ) by writing

printf("%s", str);

The conversion character 's' is used to output a string. The width and precision specifications along with %s can also be included. The width specifies the minimum output field width. If the string is short, extra space is either left padded or right padded. The precision specifies the maximum number of characters to be displayed. If the string is long, the extra characters are truncated.

For example,

printf("%5.3s", str);

The above statement would print only the first three characters in a total field of five characters. Also these three characters are right justified in the allocated width. To make the string left justified, we must use a minus sign. For example,

printf("%-5.3s", str);

The next method is by using puts() function. The string can be displayed by writing

puts(str);

puts( ) is a simple function that overcomes the drawbacks of the printf() function. The puts( ) function writes a line of output on the screen. It terminates the line with a newline character ('\n'). It returns an EOF (-1) if an error occurs and returns a positive number on success.

The strings can also be written by calling the putchar() function repeatedly to print a sequence of single characters.

6. Write a C program to read and display 3 x 3 matrix.

```
#include <stdio.h>
int main ( )
{
int i, j, mat[3][3];
printf ( "Enter the elements of the matrix: \n" );
for ( i=0; i<3; i++ )
{
for ( j=0; j<3; j++ )
{
scanf ( "%d", &mat[ i ][ j ] );
}
}
printf ( "The elements of the matrix are: \n " );
for ( i=0; i<3; i++)
{
printf ("\n");
for ( j=0; j<3; j++ )
```

```
printf ( "\t %d", mat[ i ][ j ] );
}
return 0;
}
```

7. Write a program to transpose the elements of a 3 x 3 matrix.

```
#include <stdio.h>
int main ( )
{
int i, j, mat[3][3], t_mat [3][3];
printf ( "Enter the elements of the matrix: \n" );
for ( i=0; i<3; i++ )
{
for ( j=0; j<3; j++ )
{
scanf ( "%d", &mat[ i ][ j ] );
}
}
printf ( "The elements of the matrix are: \n " );
for ( i=0; i<3; i++)
{
printf("\n");
for ( j=0; j<3; j++ )
printf ( "\t %d", mat[ i ][ j ] );
}
for ( i=0; i<3; i++)
{
for ( j=0; j<3; j++ )
t_mat[ i ][ j ] = mat[ j ][ i ] ;
}
printf ( " The elements of the transposed matrix are: \n " );
```

```
for ( i=0; i<3; i++)
{
printf("\n");
for ( j=0; j<3; j++ )
printf ( "\t %d", t_mat[ i ][ j ] );
}
return 0;
}
```

8. Write a C program to implement matrix multiplication and validate the rules of multiplication.

```
#include<stdio.h>
#include<stdlib.h>
int main( )
{
int r1, c1, r2, c2, row, col, k, a[10][10],b[10][10],m[10][10];
printf ( "Enter the order of matrix A\n" );
scanf ( "%d %d",&r1, &c1 );
printf ( "Enter order of matrix B\n" );
scanf ( "%d %d", &r2, &c2 );
if ( c1 != r2)
{
printf ( "Matrix Multiplication is not possible \n" );
exit (0);
}
printf ( "Enter the elements into matrix A\ n" );
for ( row=0; row<r1; row++)
{
for ( col=0; col<c1; col++ )
scanf ( "%d" , &a[ row ][ col ] );
}
```

```
printf ( "Enter the elements into matrix B \n" );
for ( row=0; row<r2; row++ )
{
for( col=0; col<c2; col++ )
scanf ( "%d", &b[ row ][ col ]);
}
for( row=0; row<r1; row++)
{
for( col=0; col<c2; col++)
{
m[row][col]=0;
for( k=0; k<c1; k++)
{
m[ row ][ col ] = m[ row ][ col ] + a[ row ][ k ] * b[ k ][ col ];
}       }       }
printf ( " The elements of Product of Matrix A and B is: \n");
for( row=0; row<r1; row++ )
{
for (col=0; col<c2; col++ )
printf ( "%3d", m[ row ][ col ] );
}
printf ( "\n" );
}
return 0;
}
```

9. Develop a C program to sort the given set of N numbers using bubble sort.

```
#include<stdio.h>
int main( )
{
int i, j, N, temp, a[20];
```

```
printf ( "Enter the value of N: \n" );
scanf ( "%d", &N );
printf ("Enter the numbers in unsorted order:\n" );
for ( i=0; i<N; i++ )
scanf ( " %d ", &a[i] );
// bubble sort logic
for ( i=0; i<N; i++ )
{
for ( j=0; j<(N-i)-1; j++ )
{
if ( a[j]>a[j+1] )
{
temp = a[j];
a[j] = a[j+1];
a[j+1] = temp;
}
}
}
printf ( "The sorted array is \n" );
for ( i=0; i<N; i++ )
{
printf ("%d\n", a[i] );
}
return 0;
}
```

10. Write a program to count vowels and consonants in a string.

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

```
{
char s[100];
int i, vowels=0, consonants=0, len=0;
printf ( "Enter the string: " );
gets ( s );
len = strlen(s);
for ( i=0; i<len; i++ )
{
if ( s[ i ] >= 'A' && s[ i ] <= 'Z' ) || ( s[ i ] >= 'a' && s[ i ] <= 'z' )
{
if ( s[ i ] == 'a' || s[ i ] == 'e' || s[ i ] == 'i' || s[ i ] == 'o' || s[ i ] == 'u' || s[ i ] == 'A' || s[ i ]
== 'E' || s[ i ] == 'I' || s[ i ] == 'O' || s[ i ] == 'U' )
vowels++;
else
consonants++;
}
}
printf (" Vowels in the string is = %d", vowels);
printf (" Consonants in the string is = %d", consonants);
return 0;
}
```

11. Write a C program to find the sum of diagonal elements of a matrix.

```
##include<stdio.h>
int main( )
{
int r, c, i, j, a[10][10], sum;
printf ( "Enter the order of the matrix \n" );
scanf ( "%d %d", &r, &c );
if ( r != c)
{
```

```
printf ( "Not a square matrix \n" );
exit (0);
}
printf ( "Enter the elements of matrix \ n" );
for ( i=0; i<r; i++)
{
for ( j=0; j<c; j++ )
scanf ( "%d" , &a[ i ][ j ] );
}
}
for ( i=0; i<r; i++)
{
sum = sum + a[ i ][ i ];
}
printf ( " The sum of diagonal elements is %d", sum);
return 0;
}
```

12. Write program to generate Pascal's triangle.

```
#include <stdio.h>
#include <conio.h>
int main ( )
{
int arr[7][7] = { 0 };
int row=2, col, i, j;
arr[0][0] = arr[1][0] = arr[1][1] = 1;
while (row < 7)
{
arr[row][0] = 1;
for (col = 1; col <= row; col++)
arr[row][col] = arr[row-1][col-1] + arr[row-1][col];
```

```
row++;
}
for(i=0;i<7;i++)
{
printf("\n");
for (j=0; j<=i; j++)
printf ( "\t %d", arr[i][j]);
}
return 0;
}
```

13. Write a program to print following pattern.

```
#include <stdio.h>
int main ( )
{
int i, p;
char str[ ] = "HELLO";
printf ( "\n" );
for ( i=0; i<5; i++)
{
p = i+1;
printf ( "\n % -5 . * s", p, str );
}
printf ( "\n" );
for ( i=4; i>=0; i-- )
{
p = i+1;
printf ( "\n  % -5 . * s", p, str );
}
return 0;
}
```

```
H
H E
H E L
H E L L
H E L L O
H E L L O
H E L L
H E L
H E
H
```