



Arrays

In C programming, so far we have used only basic data types, namely int, float, char, double, etc. These types of variables can store only one value at a time. To process one value we need one variable in a program. If we want to process five different values, we need five different variables. Now imagine the program where we want to handle very large amount of data. To handle such kind of situations, C supports a special kind of data type called an array. In this chapter we will learn how to use different types of an array available in C language.

Need of Arrays

Imagine some of the following real life situations for which we want to write a C program :

- Maintain Marks of all the students of class
- Maintain Cost of all the products of Super Market
- List of employees and their contact numbers
- Maintain height and weight of each student of class

To handle marks of each student, instead of declaring different variables, such as marks0, marks1, marks2, marks3, marks4, and marks59, we may declare one array variable such as marks[60]. To refer individual student's mark now we can use variable like marks[0], marks[1], marks[2] ... and so on. In computer memory array of marks[59] will look like the one shown in figure 15.1.

First element	Second element	Third element	Last element
marks[0]	marks[1]	marks[2]	marks[3]	marks[59]

Figure 15.1 : Sample Structure of Array in Memory

Note following important points related to an arrays.

- It is a collection of elements having same data type.
- It is a fixed-size sequential collection of elements. An array element occupies contiguous memory locations.
- The element of an array is accessed by an index number. The index number contained within square brackets is also known as subscript.
- A subscript number must be an integer or an integer expression.
- The subscript number starts with zero.

Types of Arrays

The arrays in C language can be classified in to following two categories :

- (i) Single or One dimensional array
- (ii) Multidimensional array

The single dimensional array may have one row or one column while multidimensional array has one or more rows and one or more columns. The number of rows and columns are specified by user as per the program requirement. We will discuss only one and two dimensional arrays offered by the C language.

Declaration of Single Dimensional Array

To declare an array in C language, we specify the type of data which we want to store in the elements and the number of elements to be stored in an array as follows:

datatype arrayname [size];

- **datatype:** It specifies the type of elements that an array stores. If we want to store characters in an array then type of an array is 'char'. In the same way to store integers, type of an array is 'int'. The datatype can be any valid C language data type.
- **arrayname:** This is the name given to an array variable by the programmer. It can be any string but it is usually suggested that some meaningful name should be given to an array. The array name should be in context with what is being stored in that array in a program.
- **size:** The value of size indicates the number of elements the array can store. The size must be an integer constant greater than zero.

For example, to store marks of 60 students, we may use following statement :

int marks[60];

Note that the marks array declared here will be able to contain only 60 integer values. The example of arrays that can store characters, floating point values and very large values (double) is shown in the declarations below:

char strings[20];

float percentages[20];

double numbers[20];

Assigning Values to Single Dimensional Arrays

C language allows array initialization in two different ways:

- (i) Compile time array initialization
- (ii) Runtime array initialization

Compile Time Array Initialization

An array can be assigned values similar to a normal variable at the time of declaration, the general syntax to set the values of various array elements is as follows:

datatype arrayname[size] = { value1, value2, ..., valueN };

Here *value1, value2, ..., valueN* provide the initial values for successive elements of the array. Following is the example where we specify size and set values of all elements of arrays.

```
int marks[5] = {60,65,70,75,80};
```

This will declare an array variable marks that can store five elements with the values set as: marks[0]=60, marks[1]=65, marks[2]=70, marks[3]=75 and marks[4]=80. During such declaration, if list of values are less than the size of an array, then only specified number of elements will be initialized and remaining elements will be initialized to zero. For example, the declaration

```
int marks[5] = {60,65,70};
```

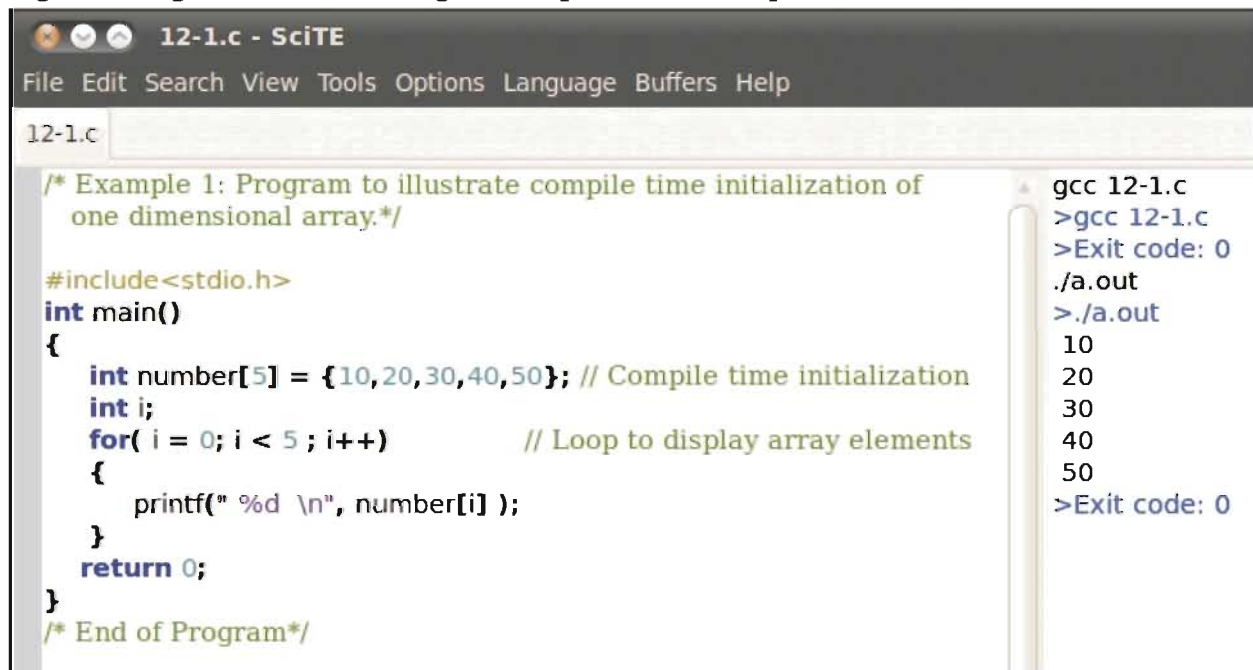
will initialize first three array elements as marks[0] = 60, marks[1] = 65 and marks[2] = 70. The remaining two array elements marks[3] and marks[4] will be initialized to value zero.

If the array elements are assigned value at the time of declaration itself then specifying size is optional. If we don't specify the size, compiler will automatically calculate the size by counting total number of values written inside the curly bracket { }. For example, the following statement

```
int marks[ ] = {60,65,70,75,80};
```

will declare the marks array containing five elements whose values are initialized with the values given in the curly bracket.

Let us learn the concept of compile time initialization of one dimensional array with example 15.1. Figure 15.2 gives the code listing and output of the example 15.1.



```
12-1.c - SciTE
File Edit Search View Tools Options Language Buffers Help
12-1.c
/* Example 1: Program to illustrate compile time initialization of
one dimensional array.*/

#include<stdio.h>
int main()
{
    int number[5] = {10,20,30,40,50}; // Compile time initialization
    int i;
    for( i = 0; i < 5; i++)          // Loop to display array elements
    {
        printf(" %d \n", number[i] );
    }
    return 0;
}
/* End of Program*/

gcc 12-1.c
>gcc 12-1.c
>Exit code: 0
./a.out
>./a.out
10
20
30
40
50
>Exit code: 0
```

Figure 15.2 : Code listing and output of Example 15.1

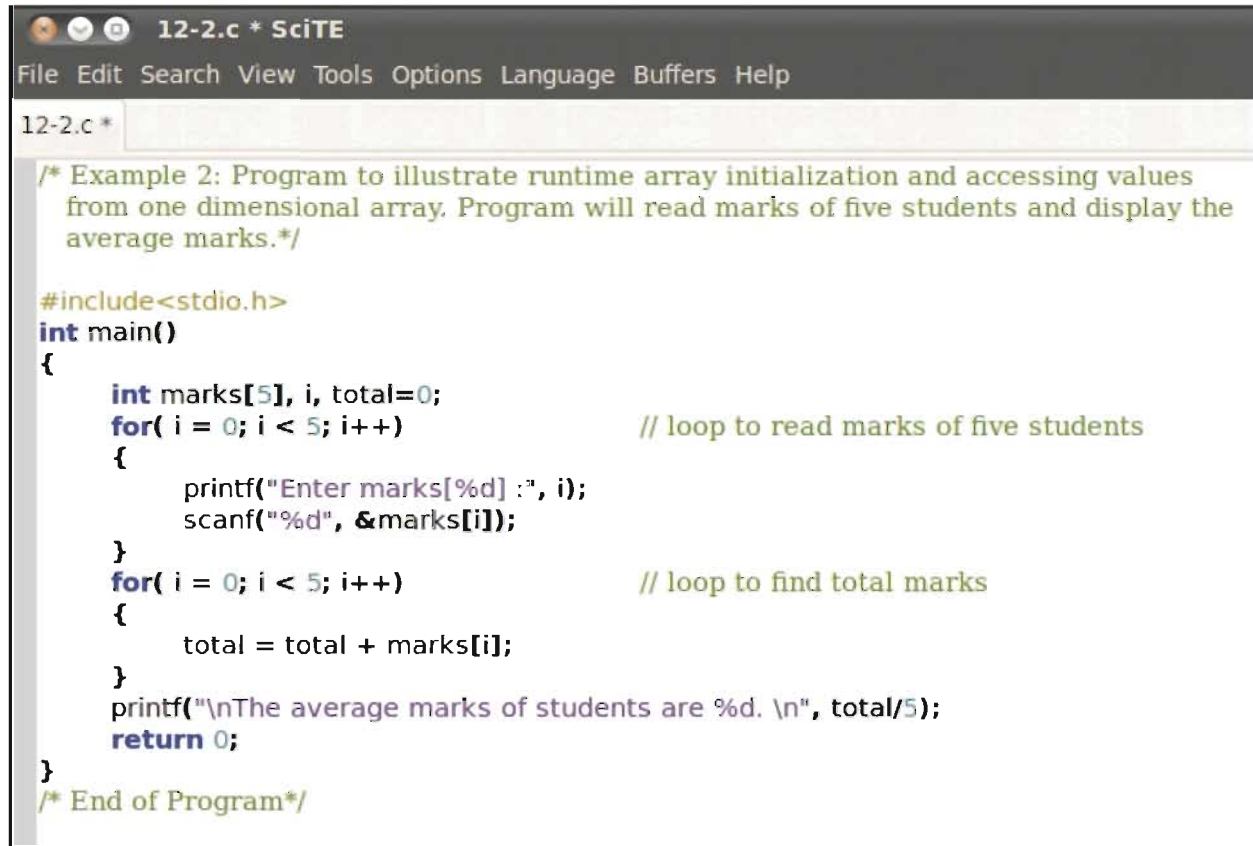
Explanation

The first line after main function declares an array called number and initializes all array elements with list of values given in curly bracket at compile time. The for loop will display the individual elements of a number array.

Runtime Array Initialization

When we need to read data from user while executing our program, we can use runtime array initialization. The code given in example 15.2 will allow user to enter marks of five different students at runtime of the program. The program also demonstrates how to use stored values from the array.

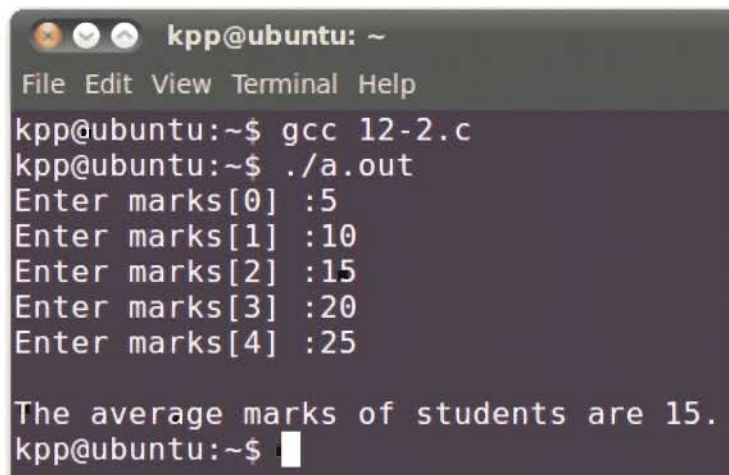
Figure 15.3 gives the code listing of the example 15.2 while figure 15.4 shows its output.



```
12-2.c * SciTE
File Edit Search View Tools Options Language Buffers Help
12-2.c *
/* Example 2: Program to illustrate runtime array initialization and accessing values
from one dimensional array. Program will read marks of five students and display the
average marks.*/

#include<stdio.h>
int main()
{
    int marks[5], i, total=0;
    for( i = 0; i < 5; i++)           // loop to read marks of five students
    {
        printf("Enter marks[%d] :", i);
        scanf("%d", &marks[i]);
    }
    for( i = 0; i < 5; i++)           // loop to find total marks
    {
        total = total + marks[i];
    }
    printf("\nThe average marks of students are %d. \n", total/5);
    return 0;
}
/* End of Program*/
```

Figure 15.3 : Code listing of Example 15.2



```
kpp@ubuntu: ~
File Edit View Terminal Help
kpp@ubuntu:~$ gcc 12-2.c
kpp@ubuntu:~$ ./a.out
Enter marks[0] :5
Enter marks[1] :10
Enter marks[2] :15
Enter marks[3] :20
Enter marks[4] :25

The average marks of students are 15.
kpp@ubuntu:~$
```

Figure 15.4 : Output of Example 15.2

Explanation

The first *for* loop in the program will read marks of five students. The second *for* loop will find the total marks by reading individual elements of marks array. The last *printf* statement will display the average marks of the students.

String as a Character Arrays

A string is a series of characters that is normally treated as a single block. C language does not

provide the *string* as inbuilt data type. However it allows us to represent string as a character array. A string variable is any valid C variable name and it is always declared as the character array. The general form of string declaration is:

```
char stringname[ size ];
```

Here the size indicates the number of characters in the variable called stringname. Followings are some of the valid string declarations:

```
char student_name[20];
```

```
char city[50];
```

```
char state[20];
```

Note that strings in a C language are terminated by the special character called null character ('\0'); this helps the program to identify the end of the string. Since strings are terminated by the null character '\0', we require one extra storage location in the character array to accommodate all the characters of strings.

Similar to numeric arrays, character arrays can also be initialized at compile time and at run time of a program. Character array initialization can be possible in following different manners:

```
char student_name[6] = "PURVA";
```

```
char student_name[6] = {'P', 'U', 'R', 'V', 'A', '\0'};
```

The reason behind keeping 6 elements in student_name array is that the string PURVA contains five characters and one element is used for null character '\0'. Note that when we initialize the character array by listing its elements, it is necessary to specify null character '\0' at the end of string. We can also declare character arrays without specifying size as follow:

```
char state[ ] = {'G', 'U', 'J', 'A', 'R', 'A', 'T', '\0' };
```

In this case the character array state will be declared and initialized with eight elements. Using following statements we can refer previously discussed character arrays.

```
printf("The name of student is %s", student_name);
```

```
printf("The name of state is %s", state);
```

```
printf("The first character of your name is %c", student_name[0]);
```

Multidimensional Array

The single dimensional array keeps track of information in linear order. However, the data associated with certain real life systems (such as digital image, a chess game board, matrix, etc.) are available in two dimensions. To think about this type of system in a program, we need a multi-dimensional data structure.

C language allows us to create multidimensional array. As mentioned earlier, multidimensional arrays have more than one rows and columns. We can create two dimensional, three dimensional, or N-dimensional array in C. The simplest form of the multidimensional array is the two-dimensional array. The two dimensional array have row index as well as column index. We will discuss only the two dimensional arrays in this chapter.

Declaration of Two Dimensional Arrays

So far we had discussed one dimensional array that can store a list of values in it. There are many situations in real life where we may want to store data tables. Consider the case given in the table 15.1. It shows the details about sale of an item by different salesmen from Monday to Friday.

	Monday	Tuesday	Wednesday	Thursday	Friday
Salesman1	100	150	200	250	200
Salesman2	200	250	300	350	300
Salesman3	150	200	250	300	250

Table 15.1 : Sales details

The table contains a total 15 different values. Here use of one dimensional array is not advisable. Better option is to use two dimensional arrays. The table 15.1 can easily be represented by using two dimensional arrays of C. The two dimensional arrays are declared as follows:

datatype arrayname [row size][column size];

The *row size* and *column size* must be an integer constant greater than zero and datatype can be any valid C language data type. Note that in C language, the multidimensional arrays are row major. In other words, the first bracket in the declaration of an array specifies number of rows.

To store the data shown in table 15.1 we can declare a two dimensional array as under:

int sales[3][5];

Here row size=3 which is used to refer salesman (Salesman1 = 0, Salesman2 = 1, and Salesman3 = 2) while column size=5 which is used to refer days. (Monday = 0, Tuesday = 1, Wednesday = 2, Thursday = 3 and Friday = 4) We have assumed that unit sales values are in integer only. Figure 15.5 shows the sample memory layout of the Sales details.

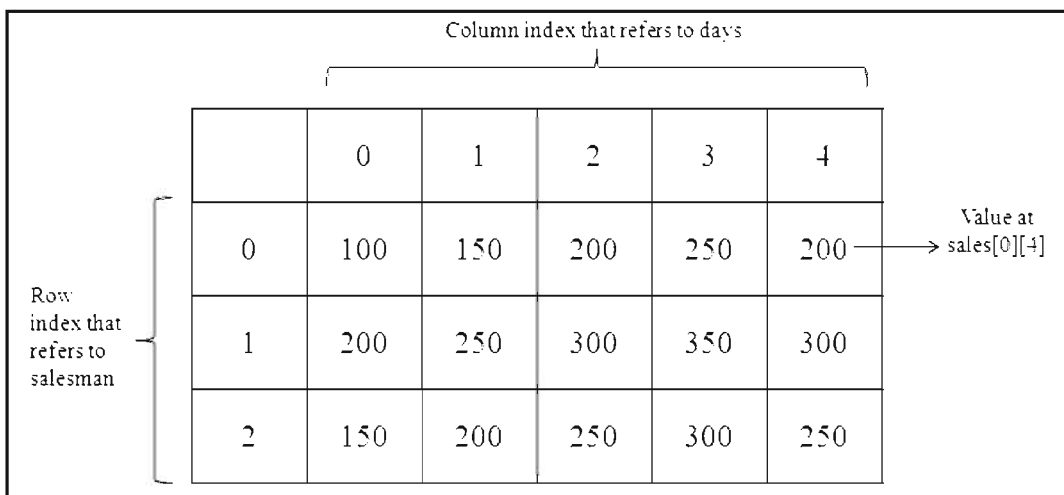


Figure 15.5 : Memory layout of Sales details

Now to check the sales of item done by Salesman1 on Monday we may use array element sales[0][0]. The value stored at sales[0][0] is 100. Similarly to check the sales of item done by Salesman2 and Salesman3 on Monday, we may use array element sales[1][0] and sales[2][0] respectively. As shown in figure 15.5 the value stored at sales[0][4] is 200 and it refers to the sales of item done by Salesman1 on Friday.

Initializing Two Dimensional Arrays

Like single dimensional array, we can also initialize two dimensional arrays at compile time and runtime of a program.

Compile Time Initialization of Two Dimensional Arrays

The compile time array initialization of table 15.1 can be performed in a program as under:

```
int sales[3][5]={
    {100, 150, 200, 250, 200},
    {200, 250, 300, 350, 300},
    {150, 200, 250, 300, 250}
};
```

The nested inner curly braces, which indicate the intended salesman row, are optional. The following initialization is equivalent to previous example:

```
int sales[3][5]={100, 150, 200, 250, 200, 200, 250, 300, 350, 300, 150, 200, 250, 300, 250};
```

Run Time Initialization of Two Dimensional Arrays

The run time array initialization of data shown in table 15.1 can be performed using program segment given in code listing 15.1.

```
int sales[3][5], row, column;
```

```
for( row = 0; row < 3; row++) {
    for (column = 0; column < 5; column++){
        printf("Enter sales item [%d][%d]:", row, column);
        scanf("%d", &sales[row][column]);
    }
}
```

Code Listing 15.1 : Program segment to add data at runtime

The program segment declares two dimensional array sales with three rows and five columns. The logic given in for loops will read total fifteen different values for sales array. Individual elements of sales array can be displayed on screen using the program segment given in code listing 15.2.

```
for( row = 0; row < 3; row++) {
    for (column = 0; column < 5; column++){
        printf("[%d][%d] := %d \n", row, column, sales[row][column]);
    }
}
```

Code Listing 15.2 : Program segment to show contents of array

Example of Two Dimensional Arrays

Let us now understand a program to illustrate runtime array initialization and accessing values from one and two dimensional arrays. The program will declare two dimensional arrays for maintaining marks of students. The code of this program is given in code listing 15.3, along with its output.

```
/* Example 3: Program to illustrate use of one and two dimensional array. The program will
maintain marks of two students in three quizzes using two dimensional arrays. The total marks
of each student will be displayed.*/

#include<stdio.h>
#define STD 2 /*Number of rows for student data*/
#define QUIZ 3 /*Number of columns for quiz data*/

int main(){
    int marks[STD][QUIZ]; /* Marks array for 2 students and 3 quizzes */
    int i, j, total_quiz[STD] = {0};

    for( i = 0; i < STD; i++) /* Loop for student row */
    {
        for( j = 0; j < QUIZ; j++) /* Loop for quiz column */
        {
            printf("Enter marks of student %d in quiz %d: ", i+1, j+1);
            scanf("%d", &marks[ i ][j ]); /* Reading marks of quiz */
            total_quiz[ i ] = total_quiz[ i ] + marks[ i ][ j ];
        }
    }

    /* Logic to print student no., quiz marks and total... */

    for ( i = 0 ; i < QUIZ; i++) /* Loop for printing quiz number... */
    {
        printf(" \tQuiz %d" , i+1);
    }

    printf(" Total \n");
    for( i = 0; i < STD; i++) /* Loop for student row */
    {
        printf("Student %d:", i+1);
        for( j = 0; j < QUIZ; j++) /* Loop for quiz column */
        {
            printf("%d \t", marks[ i ][ j ]);
        }
        printf("%d \n", total_quiz[ i ]);
    }
    return 0;
}
/* End of Program*/
```



```

/*****
Output:

Enter marks of student 1 in quiz 1: 20
Enter marks of student 1 in quiz 2: 30
Enter marks of student 1 in quiz 3: 40
Enter marks of student 2 in quiz 1: 50
Enter marks of student 2 in quiz 2: 40
Enter marks of student 2 in quiz 3: 20

           Quiz1    Quiz2    Quiz3    Total
Student 1:    20      30      40      90
Student 2:    50      40      20     110
*****/

```

Code Listing 15.3 : Code and output of example 15.3

Explanation

Here we have used two symbolic constants STD and QUIZ in our program. STD refers to the number of students to be maintained in a program. QUIZ refers to the number of quizzes to be maintained in a program. We have declared two dimensional array marks to maintain quiz marks of students. One dimensional array total_quiz is declared for storing total marks of all quizzes by a student. At the time of declaration of total_quiz, all elements are initialized to zero. Two counter variables i and j are declared to repeat *for* loop body statements as per our requirement.

A nested for loop is used to read quizzes marks of students. Within the loop we had calculated total marks of all quizzes by a student. The remaining for loops are used to display output appropriately on the screen. The given program will be able to maintain two students' marks for three quizzes only. But we can make our program more dynamic by changing values of STD and QUIZ variable in our program.

No Array Bound Check

Arrays are one of the strength of C language but it is important to note that there is no array bound check in C language. Array bound check means checking the boundaries of array declared in a program. Consider the following array declaration statement:

```
int number[5];
```

This will declare an array variable number capable of storing five integer values. We can access individual array elements by using number[0], number[1]....number[4]; C provides power to the programmer to write any index value within the [] of an array. This means we can access number[-1] and also number[6] or any other illegal index location. As there is no bound check of array index, we will not get any error but the program will give us the garbage data or it will crash.

Summary

In this chapter we had learnt important characteristics related to use of one dimensional and two dimensional arrays in C. We need to remember that the array datatype can be any valid C language data type. In array declaration the subscript number must be an integer or integer expression. An array index number starts with zero. Character arrays (string) are terminated by the special null character '\0'. There is no array boundary check in C language. It is programmer's responsibility to refer correct array index number in a program.

EXERCISE

1. Give the advantages of using one dimensional array in a program.
2. State the difference between one and two dimensional arrays.
3. What do you mean by runtime and compile time array initialization of an array?
4. Find errors, if any in the following code, correct it and write the output.

(a) `#include <stdio.h>`

```
int main( ) {  
    int num[3][3] = {{1, 2}, {3, 4}};  
    printf("%d \n", number[1][1]);  
    return 0;  
}
```

(b) `#include<stdio.h>`

```
void main()  
{  
    char str[ ] = 'INDIA'  
    printf("%c %c %c %c c", str[0], str[1], str[2], str[0], str[4]);  
}
```

5. Fill in blanks :
 - (a) The two-dimensional arrays have rows and _____.
 - (b) An array element occupies _____ memory locations.
 - (c) Index number contained within square brackets of an array is also known as _____.
 - (d) A subscript number must be an integer or _____ expression.
 - (e) Strings in a C language are terminated by the special character _____.

6. State True or False :

- (a) An array can be initialized only at runtime of a program.
- (b) The C array index number starts with one.
- (c) The simplest form of multidimensional array is one-dimensional array.
- (d) C language provides the string as inbuilt data type.
- (e) First bracket in the declaration of two dimensional arrays specifies number of rows.
- (f) The statement: `int temperature[6] = {35, 32, 42};` is valid array initialization.
- (g) The statement: `int num[] = { {1,1}, {2,2}};` is valid initialization.

7. Choose the correct option from the following :

(1) What happens if a C program tries to access a value from an array element whose subscript exceeds the size of array ?

- (a) The element will be set to 0.
- (b) The compiler would report an error.
- (c) The program may crash or gives garbage data.
- (d) The array size increases automatically.

(2) What will be the output of following program segment?

```
int num[5] = {1, 2, 3, 4, 5};
```

```
int i, j;
```

```
i = num[1];
```

```
j = num[2];
```

```
printf("%d, %d, %d", i, j, num[0]);
```

- (a) 1, 2, 3 (b) 2, 3, 1 (c) 1, 2, 0 (d) 3,4,5

(3) What will be the output of following program segment ?

```
int num[5] = {1, 2, 3};
```

```
printf("%d, %d", num[0], num[3]);
```

- (a) 1, 2 (b) 1, 0 (c) 1, 3 (d) 2,3

LABORATORY EXERCISE

Write a C program for performing followings tasks :

1. Read two five by five matrices from the user. Add these two matrices into third matrix and display the answer on screen.
2. Read marks of 20 students using an array in a program as shown below:

Roll No	Marks
1	60
2	70
...
20	75

- Display the Roll No. of students having highest marks amongst all.
 - Display total number of students having marks greater than 50.
 - Display total number of students having marks less than 50.
3. Read 10 integer numbers in one dimensional array. Display the array elements into reverse order. For example, if user input is 10, 20, 30, 40 then out output is 40, 30, 20, 10.
 4. Find maximum, minimum and average of N elements of an array.
 5. Read integer values for 3 X 3 Matrix from the user. Calculate and display sum of all elements of a Matrix.

