TOPICS:

Arrays: Introduction, One-dimensional arrays, Declaring and Initializing arrays, Multidimensional arrays.
Strings: Introduction to Strings, String operations with and without using String handling functions, Array of strings

1. What is an array? How to declare and initialize arrays? Explain with examples

Ans: Array:-

An array is defined as an ordered set of similar data items. All the data items of an array are stored in consecutive memory locations in RAM. The elements of an array are of same data type and each item can be accessed using the same name.

Declaration of an array:- We know that all the variables are declared before they are used in the program. Similarly, an array must be declared before it is used. During declaration, the size of the array has to be specified. The size used during declaration of the array informs the compiler to allocate and reserve the specified memory locations.

Syntax:- data_type array_name[n];

where, n is the number of data items (or) index(or) dimension.

0 to (n-1) is the range of array.

Ex: int a[5];

float x[10];

Initialization of Arrays:-

The different types of initializing arrays:

1. At Compile time
   - (i) Initializing all specified memory locations.
   - (ii) Partial array initialization
   - (iii) Initialization without size.
   - **(iv)** String initialization.
2. At Run Time

1. Compile Time Initialization

We can initialize the elements of arrays in the same way as the ordinary variables when they are declared. The general form of initialization of arrays is

type array-name[size]={ list of values};

(i) Initializing all specified memory locations:- Arrays can be initialized at the time of declaration when their initial values are known in advance. Array elements can be initialized with data items of type int, char etc.

Ex:- int a[5]={10,15,1,3,20};

During compilation, 5 contiguous memory locations are reserved by the compiler for the variable **a** and all these locations are initialized as shown in figure.

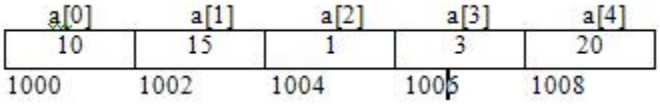


Fig: Initialization of int Arrays

Ex:-

int a[3]={9,2,4,5,6}; //error: no. of initial vales are more than the size of array.

(ii) Partial array initialization:- Partial array initialization is possible in c language. If the number of values to be initialized is less than the size of the array, then the elements will be initialized to zero automatically.

Ex:-

int a[5]={10,15};

Eventhough compiler allocates 5 memory locations, using this declaration statement; the compiler initializes first two locations with 10 and 15, the next set of memory locations are automatically initialized to 0's by compiler as shown in figure.

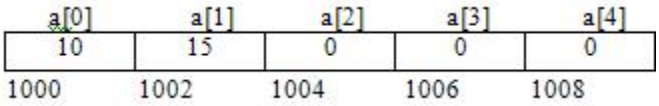
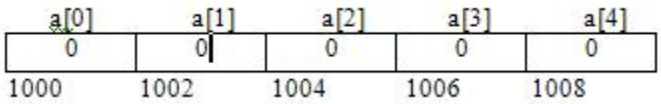


Fig: Partial Array Initialization

Initialization with all zeros:-

Ex:-

int a[5]={0};



(iii) Initialization without size:- Consider the declaration along with the initialization.

Ex:-

char b[]={'C','O','M','P','U','T','E','R'};

In this declaration, eventhough we have not specified exact number of elements to be used in array b, the array size will be set of the total number of initial values specified. So, the array size will be set to 8 automatically. The array b is initialized as shown in figure.

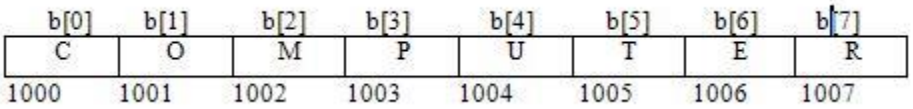


Fig: Initialization without size

Ex:- int ch[]={1,0,3,5} // array size is 4

(iv) Array initialization with a string: -Consider the declaration with string initialization.

Ex:-

char b[]="COMPUTER";

The array b is initialized as shown in figure.

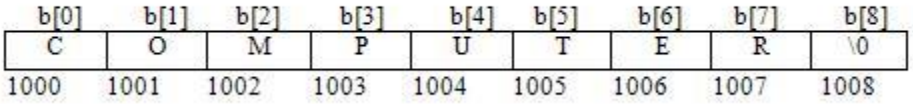


Fig: Array Initialized with a String

Eventhough the string "COMPUTER" contains 8 characters, because it is a string, it always ends with null character. So, the array size is 9 bytes (i.e., string length 1 byte for null character).

Ex:-

char b[9]="COMPUTER"; // correct

char b[8]="COMPUTER"; // wrong

2. Run Time Initialization

An array can be explicitly initialized at run time. This approach is usually applied for initializing large arrays.

Ex:- scanf can be used to initialize an array.

```
int x[3];
scanf("%d%d%d",&x[0],&x[1],&x[2]);
```

The above statements will initialize array elements with the values entered through the key board. (Or)

```
for(i=0;i<100;i=i+1)
{
if(i<50)
sum[i]=0.0;
else
sum[i]=1.0;
}
```

The first 50 elements of the array sum are initialized to 0 while the remaining 50 are initialized to 1.0 at run time.

2. How can we declare and initialize 2D arrays? Explain with examples.

Ans: An array consisting of two subscripts is known as two-dimensional array. These are often known as array of the array. In two dimensional arrays the array is divided into rows and columns. These are well suited to handle a table of data. In 2-D array we can declare an array as :

Declaration:-

Syntax: **d**ata_type array_name[row_size][column_size];

Ex:- int arr[3][3];

where first index value shows the number of the rows and second index value shows the number of the columns in the array.

Initializing two-dimensional arrays:

Like the one-dimensional arrays, two-dimensional arrays may be initialized by following their declaration with a list of initial values enclosed in braces.

Ex: int a[2][3]={0,0,0,1,1,1}; initializes the elements of the first row to zero and the second row to one. The initialization is done row by row.

The above statement can also be written as

int a[2][3] = {{ 0,0,0},{1,1,1}};

by surrounding the elements of each row by braces.

We can also initialize a two-dimensional array in the form of a matrix as shown below

```
int a[2][3]={
                {0,0,0},
                {1,1,1}
            };
```

When the array is completely initialized with all values, explicitly we need not specify the size of the first dimension.

```
Ex:  int  a[][3]={
                {0,2,3},
                {2,1,2}
            };
```

If the values are missing in an initializer, they are automatically set to zero.

```
Ex:  int  a[2][3]={
                {1,1},
                 {2}
             };
```

Will initialize the first two elements of the first row to one, the first element of the second row to two and all other elements to zero.

3. Explain how two dimensional arrays can be used to represent matrices. (or)

Define an array and how the memory is allocated for a 2D array?

Ans: These are stored in the memory as given below.

- **Row-Major order Implementation**
- **Column-Major order Implementation**

In Row-Major Implementation of the arrays, the arrays are stored in the memory in terms of the row design, i.e. first the first row of the array is stored in the memory then second and so on. Suppose we have an array named **arr** having 3 rows and 3 columns then it can be stored in the memory in the following manner :

```
int    arr[3][3];
```

arr[0][0]	arr[0][1]	arr[0][2]
arr[1][0]	arr[1][1]	arr[1][2]
arr[2][0]	arr[2][1]	arr[2][2]

Thus an array of 3*3 can be declared as follows :

```
arr[3][3] = { 1, 2, 3,

              4, 5, 6,

              7,8,9};
```

and it will be represented in the memory with row major implementation as follows :

1	2	3	4	5	6	7	8	9

In Column-Major Implementationof the arrays, the arrays are stored in the memory in the term of the column design, i.e. the first column of the array is stored in the memory then the second and so on. By taking above eg. we can show it as follows :

arr[3][3] = { 1, 2, 3,

4, 5, 6,

7,8,9};

and it will be represented in the memory with column major implementation as follows :

1	4	7	2	5	8	3	6	9

Two-dimensional arrays of variable length

An array consisting of two subscripts is known as two-dimensional array. These are often known as array of the array. In two dimensional arrays the array is divided into rows and columns,. These are well suited to handle the table of data. In 2-D array we can declare an array as :

Declaration:-

Syntax: data_type array_name[row_size][column_size];

Ex: int arr[3][3] ;

Where first index value shows the number of the rows and second index value shows the no. of the columns in the array.
These are stored in the memory as given below.

arr[0][0]	arr[0][1]	arr[0][2]
arr[1][0]	arr[1][1]	arr[1][2]
arr[2][0]	arr[2][1]	arr[2][2]

Initialization :-

To initialize values for variable length arrays we can use scanf statement and loop constructs.

Ex:-

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

for(j=0;j<3;j++)

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

4. Define multi-dimensional arrays? How to declare multi-dimensional arrays?

Ans: Multidimensional arrays are often known as array of the arrays. In multidimensional arrays the array is divided into rows and columns, mainly while considering multidimensional arrays we will be discussing mainly about two dimensional arrays and a bit about three dimensional arrays.

Syntax: data_type array_name[size1][size2][size3]------[sizeN];

In 2-D array we can declare an array as :

```
int   arr[3][3] = { 1, 2, 3,

                    4, 5, 6,

                   7,8,9
                 };
```

where first index value shows the number of the rows and second index value shows the number of the columns in the array. To access the various elements in 2-D array we can use:

```
printf("%d", a[2][3]);

/* output will be 6, as a[2][3] means third element of the second row of the array */
```

In 3-D we can declare the array in the following manner :

```
int  arr[3][3][3] =

              {1, 2, 3,
               4, 5, 6,
               7, 8, 9,
```

```
    10, 11, 12,
    13, 14, 15,
    16, 17, 18,

    19, 20, 21,
    22, 23, 24,
    25, 26, 27
  };
```

/* here we have divided array into grid for sake of convenience as in above declaration we have created 3 different grids, each have rows and columns */

If we want to access the element the in 3-D array we can do it as follows :

```
printf("%d",a[2][2][2]);
```

/* its output will be 26, as a[2][2][2] means first value in [] corresponds to the grid no. i.e. 3 and the second value in [] means third row in the corresponding grid and last [] means third column */

Ex:-

```
int  arr[3][5][12];

float table[5][4][5][3];
```

arr is 3D array declared to contain 180 (3*5*12) int type elements. Similarly table is a 4D array containing 300 elements of float type.

5. Write a program to perform matrix addition.

Ans: /*ADDITION OF TWO MATRICES*/

```
#include<stdio.h>

#include<conio.h>

#include<process.h>

void main()

  {

int a[10][10],b[10][10],c[10][10];
```

```
int i,j,m,n,p,q;
printf("\n Enter the size of Matrix A:");
scanf("%d%d", &m,&n);
printf("\n Enter the size of Matrix B:");
scanf("%d%d", &p,&q);
if(m!=p || n!=q)
    {
        printf("Matrix addition not possible.");
        exit(0);
    }
printf(" Enter the Matrix A values:\n");
for(i=0;i<m;i++)
        for(j=0;j<n;j++)
        scanf("%d",&a[i][j]);
printf(" Enter the Matrix B values:\n");
for(i=0;i<p;i++)
        for(j=0;j<q;j++)
        scanf("%d",&b[i][j]);
for(i=0;i<m;i++)
        for(j=0;j<n;j++)
        c[i][j]=a[i][j]+b[i][j];
```

```
printf("\n The Matrix A is\n");
for(i=0;i<m;i++)
        {
        for(j=0;j<n;j++)
        printf(" %d",a[i][j]);
        printf("\n");
        }
printf("\n The Matrix B is\n");
for(i=0;i<p;i++)
        {
        for(j=0;j<q;j++)
        printf(" %d",b[i][j]);
        printf("\n");
        }
printf("\n The Output Matrix C is\n");
for(i=0;i<m;i++)
        {
        for(j=0;j<n;j++)
        printf(" %d",c[i][j]);
        printf("\n");
```

```
        }
    }
```

OUTPUT:

```
Enter the size of Matrix A:2
3
 Enter the size of Matrix B:2
3
 Enter the Matrix A values:
1
2
3
4
5
6
 Enter the Matrix B values:
6
5
4
3
2
1
```

The Matrix A is

1 2 3

4 5 6

The Matrix B is

6 5 4

3 2 1

The Output Matrix C is

7 7 7

7 7 7

6. Write a program to perform matrix multiplication

Ans: /*MATRIX MULTIPLICATION*/

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
void main()
  {
int a[10][10],b[10][10],c[10][10];
int i,j,k,m,n,p,q;
printf("\ Enter the size of Matrix A:");
scanf("%d%d", &m,&n);
printf("\ Enter the size of Matrix B:");
scanf("%d%d", &p,&q);
if(n!=p)
    {
```

```
    printf("Matrix Multiplication not possible.");
    exit(0);
    }
printf(" Enter the Matrix A values:\n");
for(i=0;i<m;i++)
    for(j=0;j<n;j++)
    scanf("%d",&a[i][j]);
printf(" Enter the Matrix B values:\n");
for(i=0;i<p;i++)
    for(j=0;j<q;j++)
    scanf("%d",&b[i][j]);
for(i=0;i<m;i++)
    for(j=0;j<q;j++)
     {
    c[i][j]=0;
    for(k=0;k<n;k++)
    c[i][j]=c[i][j]+a[i][k]*b[k][j];
     }
printf("\n The Matrix A is\n");
for(i=0;i<m;i++)
    {
    for(j=0;j<n;j++)
    printf(" %d",a[i][j]);
    printf("\n");
    }
printf("\n The Matrix B is\n");
for(i=0;i<p;i++)
    {
    for(j=0;j<q;j++)
    printf(" %d",b[i][j]);
```

```
    printf("\n");
    }
printf("\n The Output Matrix C is\n");
for(i=0;i<m;i++)
    {
    for(j=0;j<q;j++)
    printf(" %d",c[i][j]);
    printf("\n");
    }
  }
```

OUTPUT:

```
Enter the size of Matrix A:2
3
 Enter the size of Matrix B:3
2
 Enter the Matrix A values:
2
2
2
2
2
2
 Enter the Matrix B values:
3
3
3
3
3
3
```

The Matrix A is

2 2 2

2 2 2

The Matrix B is

3 3

3 3

3 3

The Output Matrix C is

15 15

15 5

7. Define C string? How to declare and initialize C strings with an example?

Ans: C Strings:-

In C language a string is group of characters (or) array of characters, which is terminated by delimiter \0 (null). Thus, C uses variable-length delimited strings in programs.

Declaring Strings:-

C does not support string as a data type. It allows us to represent strings as character arrays. In C, a string variable is any valid C variable name and is always declared as an array of characters.

Syntax:- char string_name[size];

The size determines the number of characters in the string name.

Ex:- char city[10];

char name[30];

Initializing strings:-

There are several methods to initialize values for string variables.

Ex:- char str1[6]="HELLO";

H	E	L	L	O	\0

Ex:- char month[]="JANUARY";

J	A	N	U	A	R	Y	\0

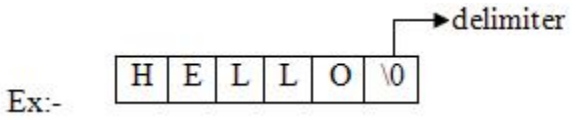
Ex:- char city[8]="NEWYORK";

char city[8]={„N“,“E“,“W“,“Y“,“O“,“R“,“K“,“\0“};

The string city size is 8 but it contains 7 characters and one character space is for NULL terminator.

Storing strings in memory:-

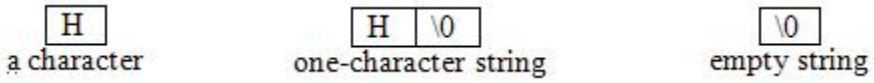
In C a string is stored in an array of characters and terminated by \0 (null).



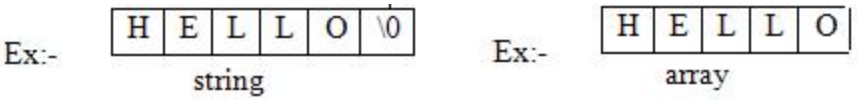
A string is stored in array, the name of the string is a pointer to the beginning of the string.

The character requires only one memory location.

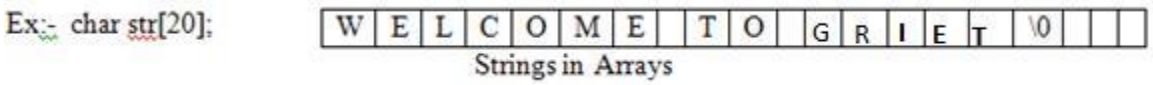
If we use one-character string it requires two locations. The difference is shown below,



The difference between array and string is shown below,



Because strings are variable-length structure, we must provide enough room for maximum-length string to store and one byte for delimiter.



Why do we need null?

A string is not a datatype but a data structure. String implementation is logical not physical. The physical structure is array in which the string is stored. The string is variable-length, so we need to identify logical end of data in that physical structure.

String constant (or) Literal:-

String constant is a sequence of characters enclosed in double quotes. When string constants are used in C program, it automatically initializes a null at end of string.

Ex:- “Hello” “Welcome” “Welcome to C Lab”

8. Explain about the string Input/ Output functions with example? Ans: Reading and Writing strings:-

C language provides several string handling functions for input and output.

String Input/Output Functions:-

C provides two basic methods to read and write strings. Using formatted input/output functions and using a special set of functions.

Reading strings from terminal:-

(a) **formatted input function**:- scanf can be used with %s format specification to read a string.

Ex:- char name[10];

scanf(“%s”,name);

Here don‟t use „&‟ because name of string is a pointer to array. The problem with scanf is that it terminates its input on the first white space it finds.

Ex:- NEW YORK

Reads only NEW (from above example).

(b) **Unformatted input functions**:-

(1) **getchar()**:- It is used to read a single character from keyboard. Using this function repeatedly we may read entire line of text

Ex:- char ch=‟z‟;

ch=getchar();

(2) **gets()**:- It is more convenient method of reading a string of text including blank spaces.

Ex:- char line[100];

gets(line);

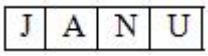
Writing strings on to the screen:-

(1) **Using formatted output functions**:- printf with %s format specifier we can print strings in different formats on to screen.

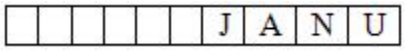
Ex:- char name[10];

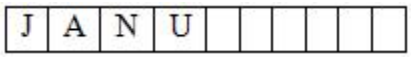
printf(“%s”,name);

Ex:- char name[10];

printf(“%0.4”,name); 

/* If name is JANUARY prints only 4 characters ie., JANU */

Printf(“%10.4s”,name); 

printf(“%-10.4s”,name); 

(2) **Using unformatted output functions**:-

(a) **putchar()**:- It is used to print a character on the screen.
Ex:- putchar(ch);

(b) **puts()**:- It is used to print strings including blank spaces.
Ex:- char line[15]=”Welcome to lab”;

puts(line);

9. Explain about the following string handling functions with example programs.

(i) strlen (ii) strcpy (iii) strcmp (iv) strcat

Ans:

C supports a number of string handling functions. All of these built-in functions are aimed at performing various operations on strings and they are defined in the header file **string.h**.

(i). strlen( )

This function is used to find the length of the string excluding the NULL character. In other words, this function is used to count the number of characters in a string. Its syntax is as follows:

```
Int strlen(string);
```

Example: char str1[ ] = “WELCOME”;

int n;

n = strlen(str1);

/* A program to calculate length of string by using strlen() function*/

#include<stdio.h>

```
#include<string.h>
main()
{
char string1[50];
int length;
printf("\n Enter any string:");
gets(string1);
length=strlen(string1);
printf("\n The length of string=%d",length);
}
```

(ii). strcpy( )

This function is used to copy one string to the other. Its syntax is as follows:

```
strcpy(string1,string2);
```

where string1 and string2 are one-dimensional character arrays.

This function copies the content of string2 to string1.

E.g., string1 contains master and string2 contains madam, then string1 holds madam after execution of the strcpy (string1,string2) function.

Example:

```
char str1[ ] = "WELCOME";
char str2[ ] ="HELLO";
strcpy(str1,str2);
```

/* A program to copy one string to another using strcpy() function */

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

```
{
char string1[30],string2[30];
printf("\n Enter first string:");
gets(string1);
printf("\n Enter second string:");
gets(string2);
strcpy(string1,string2);
printf("\n First string=%s",string1);
printf("\n Second string=%s",string2);
}
```

(iii). strcmp ( )

This function compares two strings character by character (ASCII comparison) and returns one of three values {-1,0,1}. The numeric difference is „0" if strings are equal .If it is negative string1 is alphabetically above string2 .If it is positive string2 is alphabetically above string1.

Its syntax is as follows:

```
Int strcmp(string1,string2);
```

Example:

```
char str1[ ] = "ROM";
char str2[ ] ="RAM";
strcmp(str1,str2);
(or)
strcmp("ROM","RAM");
```

/* A program to compare two strings <u>using strcmp()</u> function */

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

```
{
char string1[30],string2[15];
int x;
printf("\n Enter first string:");
gets(string1);
printf("\n Enter second string:");
gets(string2);
x=strcmp(string1,string2);
if(x==0)
printf("\n Both strings are equal");
else if(x>0)
printf("\n First string is bigger");
else
printf("\n Second string is bigger");
}
```

(iv). strcat ( )

This function is used to concatenate two strings. i.e., it appends one string at the end of the specified string. Its syntax as follows:

```
strcat(string1,string2);
```

where string1 and string2 are one-dimensional character arrays.

This function joins two strings together. In other words, it adds the string2 to string1 and the string1 contains the final concatenated string. E.g., string1 contains **prog** and string2 contains **ram**, then string1 holds **program** after execution of the strcat() function.

Example:

```
char str1[10 ] = "VERY";
char str2[ 5] ="GOOD";
```

```
strcat(str1,str2);
```

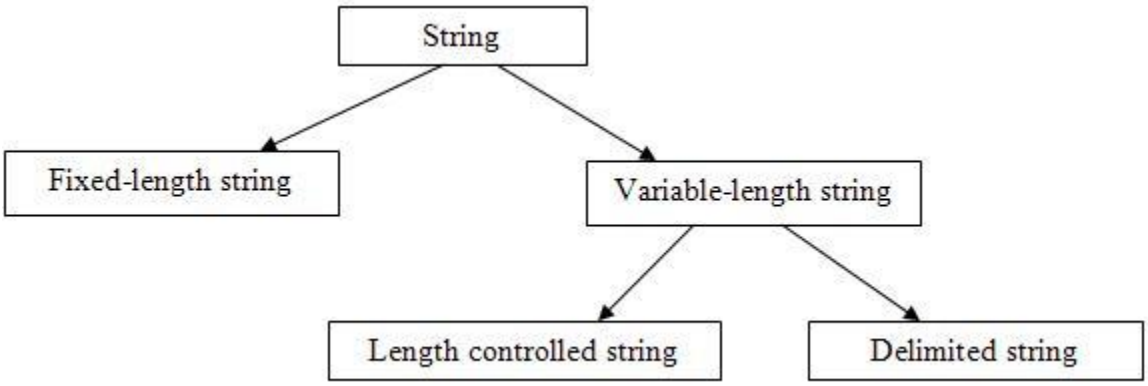
/* A program to concatenate one string with another <u>using strcat() function</u>*/

```
#include<stdio.h>

#include<string.h>

main()

{

char string1[30],string2[15];

printf("\n Enter first string:");

gets(string1);

printf("\n Enter second string:");

gets(string2);

strcat(string1,string2);

printf("\n Concatenated string=%s",string1);

}
```

10. Write about storage representation of fixed and variable length format strings with example?

Ans: <u>String concepts</u>:-

In general a string is a series of characters (or) a group of characters. While implementation of strings, a string created in pascal differs from a string created in C language.



1. **Fixed-length strings:**
   When implementing fixed-length strings, the size of the variable is fixed. If we make it too small we can‟t store, if we make it too big, then waste of memory. And another problem is we can‟t differentiate data (characters) from non-data (spaces, null etc).

2. **Variable-length string:**
   The solution is creating strings in variable size; so that it can expand and contract to accommodate data. Two common techniques used,

(a) Length controlled strings:

These strings added a count which specifies the number of characters in the string.

Ex:-

5	H	E	L	L	O

(b) Delimited strings:

Another technique is using a delimiter at the end of strings. The most common delimiter is the ASCII null character (\0).

Ex:-

H	E	L	L	O	\0

11. How can we declare and initialize Array of strings in C? Write a program to read and display array of strings.

Ans: We have array of integers, array of floating point numbers, etc.. similarly we have array of strings also.

Collection of strings is represented using array of strings.

Declaration:-

Char arr[row][col];

where,

arr - name of the array

row - represents number of strings

col - represents size of each string

Initialization:-

char arr[row][col] = { list of strings };

Example:-

char city[5][10] = { “DELHI”, “CHENNAI”, “BANGALORE”, “HYDERABAD”, “MUMBAI” };

D	E	L	H	I	\0				
C	H	E	N	N	A	I	\0		
B	A	N	G	A	L	O	R	E	\0
H	Y	D	E	R	A	B	A	D	\0
M	U	M	B	A	I	\0			

In the above storage representation memory is wasted due to the fixed length for all strings.