Unit-02
Arrays

Arrays

- An array is a collection of data that holds fixed number of values of same type.
- An array is a derived data type.
- The size and type of arrays cannot be changed after its declaration.
- Array variable can store more than one value at a time where other variable can store one value at a time.
- For Example:
  - If you want to store marks of 100 students you can create an array for it.
  - float marks[100];

Marks[0]	Marks[1]	Marks[2]	……	Marks[99]

- For Example:
  - List of employees in an organization.
  - Test scores of a class of students.
  - List of customers and their telephone numbers.
  - List of students in the college.
  - For Example, to represent 100 students in college, can be written as → student [100]
  - Here student is a array name and [100] is called index or subscript.

Types of Arrays –

1. One-dimensional arrays
2. Two-dimensional arrays
3. Multi-dimensional arrays

One-dimensional arrays

- A variable which represents the list of items using only one index (subscript) is called one-dimensional array.
- For Example –
  - If we want to represent a set of five numbers say (35, 40, 20, 57, 19), by an array variable number, then number is declared as → int number [5];
- Computer store (35, 40, 20, 57, 19) as shown below:

  The values can be assigned to the array as:
  - number [0] = 35;
  - number [1] = 40;

- number [2] = 20;
- number [3] = 57;
- number [4] = 19;

- **Declaration of One-dimensional array –**
  - The general form of array declaration is:

    type array-name[size];
  - Here the type specifies the data type of elements contained in the array, such as int, float, or char.
  - The size indicates the maximum numbers of elements that can be stored inside the array.
  - For example:
    - int rollno [10] ;
  - here int is type, rollno is a variable name, 10 is a size of array and the subscripts (index) is start from 0 to 9.
- **Initialization of One-dimensional array**
  - After an array is declared, its elements must be initialized.
  - In C programming an array can be initialized at either of the following stages:
    - At compile time
    - At run time
  - **Compile time initialization –**
    - In compile time initialization, the array is initialized when they are declared.
    - The general form of initialization of array is:

      type array-name[size] = { list of values };
    - The list of values is separated by commas.
    - Example:
      - int number[3] = {4, 5, 9};
      - Here declare the variable 'number' as an array of size 3 and will assign the values to each elements.
        - 4 is assign to first element(number[0]),
        - 5 is assign with second element(number[1])
        - 9 is assign with third element(number[2]).
    - If the number of values in the list is less than the length (size), then only that many elements will be initialized.
    - The remaining elements will be set to zero automatically.
    - If we have more initializers than the declared size, the compiler will produce an error.
  - **Runtime Array initialization –**
    - n array can also be initialized at runtime using scanf() function.
    - This approach is usually used for initializing large arrays, or to initialize arrays with user specified values.

- To input elements in an array, we can use a for loop or insert elements at a specific index.
- For Example:

```
#include<stdio.h>
void main()
{
    int array[5];
    printf("Enter 5 numbers to store them in array \n");
    for(i=0;i<5;i++)
    {
            scanf("%d", &array[i]);
    }
    printf("Element in the array are: \n");
    for(i=0;i<5;i++)
    {
            printf("Element stored at a[%d]=%d \n", i, array[i]);
    }
getch();
}
```

Two-dimensional arrays

- A variable which represents the list of items using two index (subscript) is called two-dimensional array.
- Two-dimensional array is known as matrix.
- In Two dimensional arrays, the data is stored in rows and columns format.
- 2-d array is a collection of 1-D array placed one below the other.
- Its syntax is:

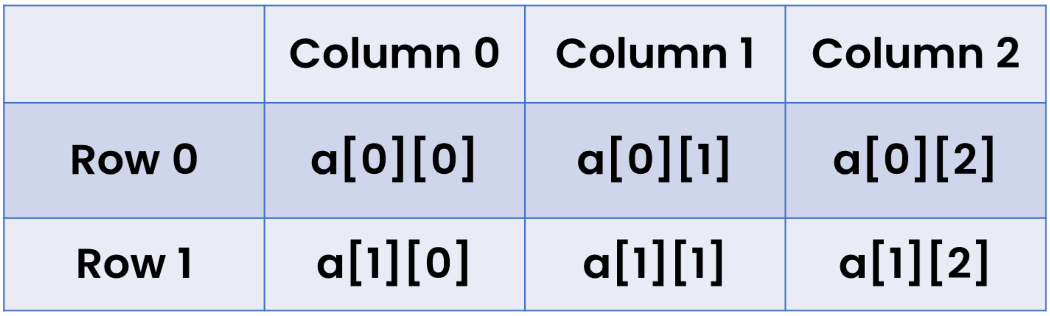
  Data-type array name[row][column];

  Total no. of elements in 2-D array is calculated as row*column
- Example:

  int a[2][3];

  Total no of elements= row*column = 2*3 = 6

  It means the matrix consist of 2 rows and 3 columns
- A two-dimensional array a, which contains 2 rows and 3 columns can be shown as follows:



	Column 0	Column 1	Column 2
Row 0	a[0][0]	a[0][1]	a[0][2]
Row 1	a[1][0]	a[1][1]	a[1][2]

- For example: int a[2][3] = {20, 2, 7, 8, 3, 15};

	Column 0	Column 1	Column 2
Row 0	**20**	**2**	**7**
Row 1	**8**	**3**	**15**

- **Declaration of Two-dimensional arrays:**
    - The general form of two-dimensional array declaration is:

      type array-name[row_size][column_size];
    - Here the type specifies the data type of elements contained in the array, such as int, float, or char.
    - The size indicates the size of number of rows and number of columns.
- **Initialization of Two-dimensional arrays:**
    - The general form of initializing two-dimensional array is :

      type array-name[row_size][column_size] = {list of values};
    - Example: int table[2][3] = {0, 0, 0, 1, 1, 1};
    - Here the elements of first row initializes to zero and the elements of second row initializes to one.
    - This above statement can be written as :

      ```
      int table[2][3] =     {
                                  {0,0,0},
                                  {1,1,1}
                            };
      ```
    - In two-dimensional array the row size can be omitted.
    - Example:

      int table[ ][3] = {{0,0,0}, {1,1,1}};
        - If the values are missing in an initializer, they are automatically set to zero.
    - Example:

      int table[2][3] = {1,1,2};
        - Here first row initializes to 1,1 and 2, and second row initialize to 0,0 and 0 automatically.
- **Memory layout of Two-dimensional arrays:**
    - In Two dimensional arrays, the data is stored in rows and columns format.
    - For example:

      int table[2][3] = {1,2,3,4,5,6};
        - The memory layout of above example:
            - table[0][0] = 1;
            - table[0][1] = 2
            - table[0][2] = 3;

  - table[1][0] = 4;
  - table[1][1] = 5;
  - table[1][2] = 6;

- **Accessing Two-dimensional arrays:**
  - For processing 2-d array, we use two nested for loops.
  - The outer for loop corresponds to the row and the inner for loop corresponds to the column.
  - For example: int a[4][5];

    For reading value:

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

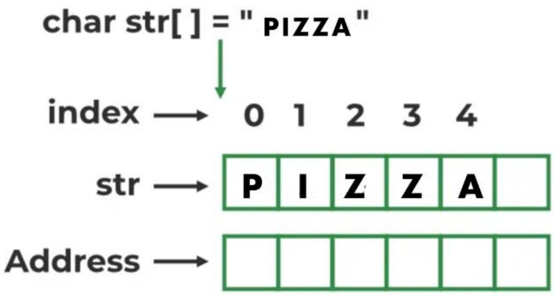
    For displaying value:

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

Strings in C

Strings

- The C String is stored as an array of characters.
- A string is a sequence of characters terminated with a null character: \0
- The difference between a character array and a C string is the string is terminated with a unique character '\0'.



String Declaration Syntax –

- Declaring a string in C is as simple as declaring a one-dimensional array.
  char string_name[size];
  - ▪ str_name is any name given to the string variable
  - ▪ size is used to define the length of the string,
    - i.e the number of characters strings will store
- There is an extra terminating character which is the Null character '\0'
- It indicates the termination of a string that differs strings from normal character arrays.

String Initialization –

- We can initialize a C string in 4 different ways.
  - o Assigning a string literal without size
  - o Assigning a string literal with a predefined size
  - o Assigning character by character with size
  - o Assigning character by character without size
- **Assigning a string literal without size**
  - o String literals can be assigned without size. Here, the name of the string str acts as a pointer because it is an array.
    char str[] = "HelloDYPians";
    ** When a Sequence of characters enclosed in the double quotation marks is encountered by the compiler, a null character '\0' is appended at the end of the string by default.
- **Assigning a string literal with a predefined size**
  - o String literals can be assigned with a predefined size.
  - o But we should always account for one extra space which will be assigned to the null character.

  - If we want to store a string of size n then we should always declare a string with a size equal to or greater than n+1.

    char str[50] = " HelloDYPians ";

- **Assigning character by character with size**
  - We can also assign a string character by character.
  - But we should remember to set the end character as ‘\0’ which is a null character.

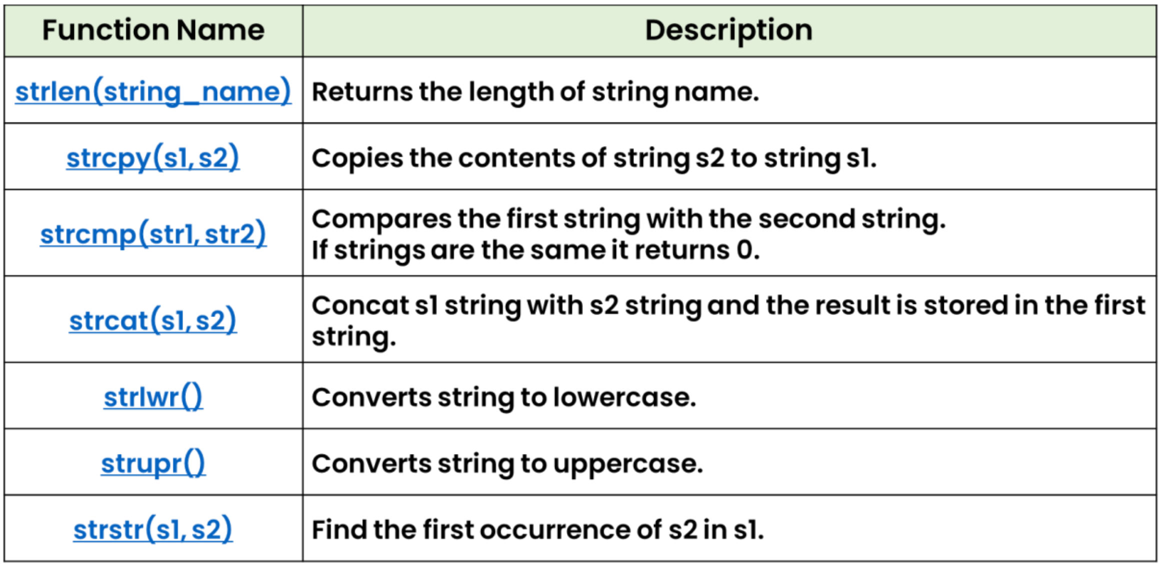
    char str[13] = { ‘H’, 'e’, ’l’, ’l’, ’o’, ’D’, ’Y’, ’P’, ’i’, ’a’, ’n’, 's’, '\0'};

- Assigning character by character without size
  - We can assign character by character without size with the NULL character at the end.
  - The size of the string is determined by the compiler automatically.

    char str[ ] = { ‘H’, 'e’, ’l’, ’l’, ’o’, ’D’, ’Y’, ’P’, ’i’, ’a’, ’n’, 's’, '\0'};

String Handling Functions

- **String Functions**
  - C also has many useful string functions, which can be used to perform certain operations on strings.
  - To use them, you must include the <string.h> header file in your program – #include <string.h>



Function Name	Description
strlen(string_name)	Returns the length of string name.
strcpy(s1, s2)	Copies the contents of string s2 to string s1.
strcmp(str1, str2)	Compares the first string with the second string. If strings are the same it returns 0.
strcat(s1, s2)	Concat s1 string with s2 string and the result is stored in the first string.
strlwr()	Converts string to lowercase.
strupr()	Converts string to uppercase.
strstr(s1, s2)	Find the first occurrence of s2 in s1.