❖ Arrays: -

- Definition: - "Array is sequence collection of related data item that share same name and same data type, but store a different value."
- C supports a derived data type known as array that can be used for many applications.
- Three type of array is available in C :

  1. One dimensional array.
  2. Two dimensional arrays.
  3. Multidimensional arrays.

❖ One dimensional array: -

- Definition: - "Array is sequence collection of related data item that share same name and same data type, but store a different value."
- A list of items can be given one variable name using only one subscript and such a variable is called a one–dimensional array.
- Like any other variable, arrays must be declared before they are used.
- The general form of array declaration is:

  data-type variable-name [size];
- The data-type specifies the data type of array.
- Variable is a name of variable which is declared by user.
- Size indicates the maximum number of elements that can be stored inside the array.
- For example, if we want to represent a set of five number,

  say (35,40,20,57,19), by an array
- variable number, then we may declare the variable number as follows: int number[5];

- The computer provides five storage locations as shown below:
  number[0]
  number[1]
  number[2]
  number[3]
  number[4]

- The values to the array elements can be assigned as follows:
  number[0] = 35;
  number[1] = 40;
  number[2] = 20;
  number[3] = 57;
  number[4] = 19;

- This would cause the array number to store the values as shown below:

  number[0]
  number[1]
  number[2]
  number[3]
- These elements may be used in programs just like any other C variable.
- For example, the following are valid statements:
  a = number[0] + 10;
- number[4] = number[0] + number [2];
- number[2] = x[8] + y [10];
- When the compiler sees the character array sting, it terminates it with additional null character.
- The null character of the character is determined by '\0'.
- When declaring character array, we must allow one extra element space for the null terminator.

- ***Initialisation of one dimensional array: -***
- Initialisation means to store the values to an array element.
- An array can be initialized at two ways.
    1. At compile time.
    2. At run time.
- **Compile time initialisation: -**
- We can initialisation the elements of arrays in the same way as the simple variables when they are declared.
- The general form of initialisation of arrays is:
- data-type array-name[size] = {list of values};
- The values in the list are separated by commas.
- For example int num[3] = {10, 23, 43};
- If the number of elements are greater then number of assigned value at that time only that elements will be initialised. The remaining elements will be initialised with zero.
  For example int num[5] = {2, 4, 5};

  [2] num[0]
  [4] num[1]
  [5] num[2]
  [0] num[3]
  [0] num[4]
- In some case the size of array will be omitted.
- In such case, the compiler allocates enough space for all initialized elements.
- For example int num[ ] = {10, 49, 54, 43};
- In this statement the size of array will be four.
- Character array may be initialized in same way.
- For example char name[ ] = {'d', 'o', 'c', 't', 'o', 'r', '\0'};
  char city[ ] = "idar";
- Last character can be initialised with NULL character '\0' .

- Run time initialization: -
- An array can be initialized at run time.
- Run time initialization is used with large array.
- Consider the following example.

```
int i;
int num[50];
for( i = 1; i<=50; i++)
{
        scanf("%d", & num[i]);
}
```

- The 50 elements are initialised with values which are provided by user from keyboard at runtime.

Example:-

```
int main ()
{
   int n[ 10 ]; /* n is an array of 10 integers */
   int i,j;    /* initialize elements of array n to 0 */
   for ( i = 0; i < 10; i++ )
   {
      n[ i ] = i + 100; /* set element at location i to i + 100 */
   }
   /* output each array element's value */
   for (j = 0; j < 10; j++ )
   {
      printf("Element[%d] = %d\n", j, n[j] );
   }
   return 0;
}
```

- When the above code is compiled and executed, it produces the following result:

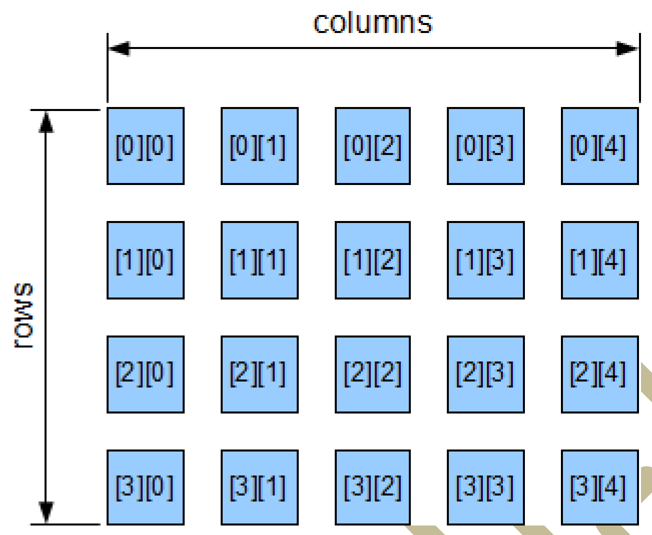
```
Element[0] = 100
Element[1] = 101
Element[2] = 102
Element[3] = 103
Element[4] = 104
Element[5] = 105
Element[6] = 106
Element[7] = 107
Element[8] = 108
Element[9] = 109
```

❖ **Two dimensional arrays: -**

- C allows us to define a tabular form by using two dimensional arrays.
- Two dimensional arrays can be declared as under. data_type array_name [row_size] [column_size];
- This is use the one pair of parentheses with commas to separate array size.
- C places each size in its own set of bracket.
- Each dimension of array is indexed from zero to its maximum size minus one.
- First index select the row and the second index selects the column within that row.

- A table of values is a useful analogy for understanding a two-dimensional array.

- A simple way of identifying a table entry is by its position in the table; that is, by its row and its column. Consider the row and column indices in the figure below.



- To identify an element of a two-dimensional array we use two pairs of brackets. The index in the left pair identifies the row, while the index in the right pair identifies the column:

  array[row][column]

- The definition of a two-dimensional array takes the form

  type identifier[r][c] = init;

- ***Initialisation of two dimension arrays: -***
- Two dimension arrays can be initialised by two ways.
  1. At compile time.
  2. At run time.
- ***Compile time initialisation:-***
- The general form of initialisation of arrays is:
- data-type array-name[row_size] [column_size] = {list of values};
- The values in the list are separated by commas and enclosed with braces.

- For example int num [2][3] = {3, 5, 6, 10, 23, 43};
- In this array first row in initialised with value 3, 5 and 6. And another row is stored with values with 10, 23 and 43.
- An above example also can be initialised as under.
- int num [2] [3] = { { 3, 5, 6}, {10, 23, 43} };

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

  int num [2] [3] = { {3, 5, 6}, {10, 23, 43} };
- We not need to specify the size of the first dimension.

  For example: int num [ ] [3] = { {1, 2, 3}, {4} };
- An above example the size of array is declared automatically. And also the value 4 is initialised on first element and remaining elements are initialised with zero.

- ***Run time initialization: -***
- An array can be initialized at run time.
- Run time initialization is used with large array.
- Consider the following example.

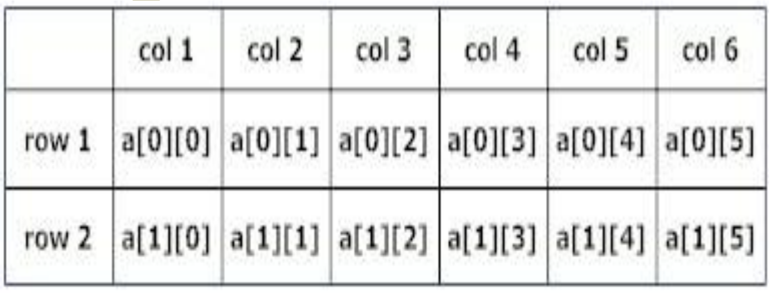
```
int i, j;
int num [10] [10];
for( i = 0; i<=9; i++)
{
        for( j = 0; j<=9; j++)
        {
                scanf("%d", & num[i] [j]);
        }
}
```
- The 100 elements are initialized with values which are provided by user from keyboard at runtime.
- ***Just remember: -***
- When we declare an array we need to specify three things, data_type, array_name and size of an array.

- Always remember that starting number of an array element is zero and last element number is always size minus one.
- Do not forget to initialize the elements. Otherwise compiler stores the "garbage" value.
- When initializing character array at that time we must provide enough space for null character '\0'.

❖ multidimensional arrays

- C programming language allows multidimensional arrays. Here is the general form of a multidimensional array declaration:
  type name[size1][size2]...[sizeN];
- For example, the following declaration creates a three dimensional 5 . 10 . 4 integer array:
  int threedim[5][10][4];

- C programming language allows programmer to create arrays of arrays known as multidimensional arrays. For example:
  float a[2][6];
- Here, a is an array of two dimension, which is an example of multidimensional array.
- For better understanding of multidimensional arrays, array elements of above example can be thinked of as below:



	col 1	col 2	col 3	col 4	col 5	col 6
row 1	a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]	a[0][5]
row 2	a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]	a[1][5]

Figure: Multidimensional Arrays

Initialization of Multidimensional Arrays

- In C, multidimensional arrays can be initialized in different number of ways.

```
int c[2][3]={{1,3,0}, {-1,5,9}};
            OR
int c[][3]={{1,3,0}, {-1,5,9}};
            OR
int c[2][3]={1,3,0,-1,5,9};
```

Initialization Of three-dimensional Array

```
double cprogram[3][2][4]={
{{-0.1, 0.22, 0.3, 4.3}, {2.3, 4.7, -0.9, 2}},
 {{0.9, 3.6, 4.5, 4}, {1.2, 2.4, 0.22, -1}},
 {{8.2, 3.12, 34.2, 0.1}, {2.1, 3.2, 4.3, -2.0}}
};
```

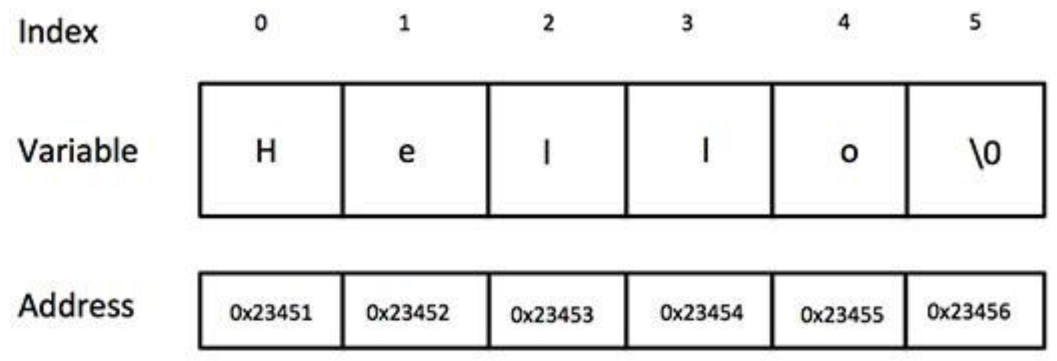
- Suppose there is a multidimensional array arr[i][j][k][m]. Then this array can hold i*j*k*m numbers of data.
- Similarly, the array of any dimension can be initialized in C programming.

❖ **Handling of Character String**

- 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. Let us try to print the above mentioned string –
- In C language, strings are stored in an array of char type along with the null terminating character "\0" at the end. In other words to create a string in C you create an array of chars and set each element in the array to a char value that makes up the string. When sizing the string array you need to add plus one to the actual size of the string to make space for the null terminating character, "\0"
- Syntax to declare a string in C:

  char fname[4];
- The above statement declares a string called fname that can take up to 3 characters. It can be indexed just as a regular array as well.
- fname[] = {'t','w','o'};

Character	T	w	o	\0
ASCII Code	116	119	41	0

❖ Initialization of strings

- In C, string can be initialized in different number of ways.

  char c[]="abcd";

  OR,

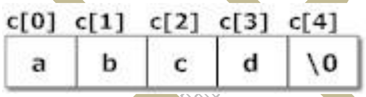
  char c[5]="abcd";

  OR,

  char c[]={'a','b','c','d','\0'};

  OR;

  char c[5]={'a','b','c','d','\0'};



❖ Reading Strings from user.

Reading words from user.

```
char c[20];
scanf("%s",c);
```

- String variable *c* can only take a word. It is because when white space is encountered, the scanf() function terminates.

Write a C program to illustrate how to read string from terminal.

Example 1:-

```
#include <stdio.h>
int main(){
    char name[20];
    printf("Enter name: ");
    scanf("%s",name);
    printf("Your name is %s.",name);
    return 0;
}
```

Example 2:-

```
#include <stdio.h>
void main() {

  char fname[30];
  char lname[30];

  printf("Type first name:\n");
  scanf("%s", fname);

  printf("Type last name:\n");
  scanf("%s", lname);
  printf("Your name is: %s %s\n", fname, lname);
}
```

Output

```
Enter name: Dennis Ritchie
Your name is Dennis.
```

Here, program will ignore Ritchie because, scanf() function takes only string before the white space.

Reading a line of text

C program to read line of text manually.

```
#include <stdio.h>
int main(){
    char name[30],ch;
    int i=0;
    printf("Enter name: ");
    while(ch!='\n')    // terminates if user hit enter
    {
        ch=getchar();
        name[i]=ch;
        i++;
    }
    name[i]='\0';      // inserting null character at end
    printf("Name: %s",name);
    return 0;
}
```

This process to take string is tedious. There are predefined functions gets() and puts in C language to read and display string respectively.

```
int main(){
    char name[30];
    printf("Enter name: ");
    gets(name);     //Function to read string from user.
    printf("Name: ");
    puts(name);    //Function to display string.
    return 0;
}
```

Both, the above program has same output below:

Output

```
Enter name: Tom Hanks
Name: Tom Hanks
```

❖ **Arithmetic Operations on Strings**

- Characters in C can be used just like integers when used with arithmetic operators.
- This is nice, for example, in low memory applications because unsigned chars take up less memory than do regular integers as long as your value does not exceed the rather limited range of an unsigned char.

```
#include<stdio.h>
   void main() {
   unsigned char val1 = 20;
   unsigned char val2 = 30;
   int answer;
   printf("%d\n", val1);
   printf("%d\n", val2);
   answer = val1 + val2;
   printf("%d + %d = %d\n", val1, val2, answer);
   val1 = 'a';
   answer = val1 + val2;
   printf("%d + %d = %d\n", val1, val2, answer);
  }
```

Arithmetic Operations On Character :

- C Programming Allows you to Manipulate on String
- Whenever the Character is variable is used in the expression then it is automatically Converted into Integer Value called ASCII value
- All Characters can be Manipulated with that Integer Value.(Addition,Subtraction)

Examples :

- ASCII value of : 'a' is 97
- ASCII value of : 'z' is 121

Possible Ways of Manipulation :

Way 1: Displays ASCII value[Note that %d in Printf]

```
char x = 'a';
printf("%d",x); // Display Result = 97
```

Way 2 : Displays Character value[Note that %c in Printf]

```
char x = 'a';
printf("%c",x); // Display Result = a
```

Way 3 : Displays Next ASCII value[Note that %d in Printf]

```
char x = 'a' + 1 ;
printf("%d",x);
// Display Result = 98 ( ascii of 'b' )
```

Way 4 Displays Next Character value[Note that %c in Printf]

```
char x = 'a' + 1;
printf("%c",x); // Display Result = 'b'
```

Way 5 : Displays Difference between 2 ASCII in Integer[Note %d in Printf]

```
char x = 'z' - 'a';
printf("%d",x);
/* Display Result = 25
   (difference between ASCII of z and a ) */
```

Way 6 : Displays Difference between 2 ASCII in Char [Note that %c in Printf]

```
char x = 'z' - 'a';
printf("%c",x);
/* Display Result = ↓
     ( difference between ASCII of z and a ) */
```

❖ String Operations

C supports a wide range of functions that manipulate null-terminated strings –

S.N.	Function & Purpose
1	**strcpy(s1, s2);** Copies string s2 into string s1.
2	**strcat(s1, s2);** Concatenates string s2 onto the end of string s1.
3	**strlen(s1);** Returns the length of string s1.
4	**strcmp(s1, s2);** Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
5	**strchr(s1, ch);** Returns a pointer to the first occurrence of character ch in string s1.
6	**strstr(s1, s2);** Returns a pointer to the first occurrence of string s2 in string s1.

The following example uses some of the above-mentioned functions –

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

int main () {

   char str1[12] = "Hello";
   char str2[12] = "World";
   char str3[12];
   int  len ;

   /* copy str1 into str3 */
   strcpy(str3, str1);
   printf("strcpy( str3, str1) :  %s\n", str3 );

   /* concatenates str1 and str2 */
   strcat( str1, str2);
   printf("strcat( str1, str2):   %s\n", str1 );
   /* total lenghth of str1 after concatenation */
   len = strlen(str1);
   printf("strlen(str1) :  %d\n", len );

   return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
strcpy( str3, str1) :  Hello
strcat( str1, str2):   HelloWorld
strlen(str1) :  10
```

strlen()

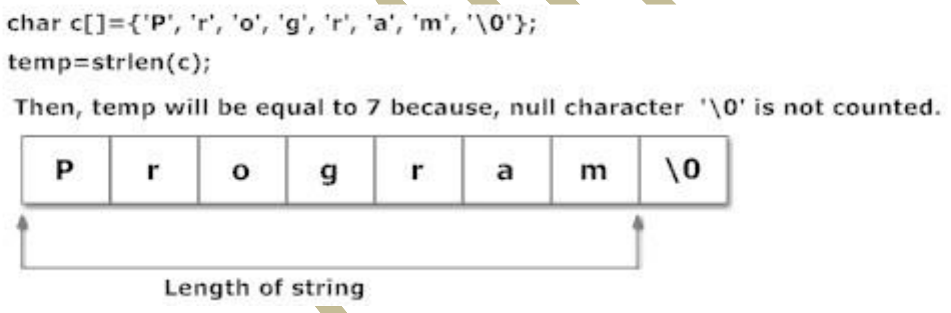
In C, strlen() function calculates the length of string. It takes only one argument, i.e, string name.

Defined in Header File <string.h>

Syntax of strlen()

temp_variable = strlen(string_name);

Function strlen() returns the value of type integer.



```
char c[]={'P', 'r', 'o', 'g', 'r', 'a', 'm', '\0'};
temp=strlen(c);
```

Then, temp will be equal to 7 because, null character '\0' is not counted.

Example of strlen()

```
#include <stdio.h>
#include <string.h>
int main(){
    char a[20]="Program";
    char b[20]={'P','r','o','g','r','a','m','\0'};
    char c[20];
    printf("Enter string: ");
    gets(c);
    printf("Length of string a=%d \n",strlen(a));
    //calculates the length of string before null charcter.
    printf("Length of string b=%d \n",strlen(b));
    printf("Length of string c=%d \n",strlen(c));
    return 0;
}
```

Output

```
Enter string: String
Length of string a=7
Length of string b=7
Length of string c=6
```

strcpy()

- Function strcpy() copies the content of one string to the content of another string. It takes two arguments.
- **Defined in Header File** <string.h>

  Syntax of strcpy()

  strcpy(destination,source);
- Here, source and destination are both the name of the string. This statement, copies the content of string source to the content of string destination.

Example of strcpy()

```
#include <stdio.h>
#include <string.h>
int main(){
    char a[10],b[10];
    printf("Enter string: ");
    gets(a);
    strcpy(b,a);   //Content of string a is copied to string b.
    printf("Copied string: ");
    puts(b);
    return 0;
}
```

Output

```
Enter string: Programming Tutorial
Copied string: Programming Tutorial
```

strcat()

In C programming, strcat() concatenates(joins) two strings. It takes two arguments, i.e, two strings and resultant string is stored in the first string specified in the argument.

Defined in Header File <string.h>

Syntax of strcat()

strcat(first_string,second_string);

Example of strcat()

```
#include <stdio.h>
#include <string.h>
int main(){
    char str1[]="This is ", str2[]="programiz.com";
    strcat(str1,str2);   //concatenates str1 and str2 and
resultant string is stored in str1.
    puts(str1);
    puts(str2);
    return 0;
}
```

Output

```
This is programiz.com
programiz.com
```

strcmp()

In C programming, strcmp() compares two string and returns value 0, if the two strings are equal. Function strcmp() takes two arguments, i.e, name of two string to compare.

Defined in Header File: <string.h>

Syntax of strcmp()

```
temp_varaible=strcmp(string1,string2);
```

Example of strcmp()

```
#include <stdio.h>
#include <string.h>
int main(){
  char str1[30],str2[30];
  printf("Enter first string: ");
  gets(str1);
  printf("Enter second string: ");
  gets(str2);
  if(strcmp(str1,str2)==0)
      printf("Both strings are equal");
```

```
    else
        printf("Strings are unequal");
    return 0;
}
```

Output

```
Enter first string: Apple
Enter second string: Apple
Both strings are equal.
```

If two strings are not equal, strcmp() returns positive value if ASCII value of first mismatching element of first string is greater than that of second string and negative value if ASCII value of first mismatching element of first string is less than that of second string. For example:

```
char str1[]="and",str2[]="cat";
temp=strcmp(str1,str2);
```

Since, ASCII value of 'a' is less than that of 'c', variable temp will be negative.