APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

KTU SPECIAL

We can together dream the B.tech

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

• KTU STUDY MATERIALS	• SYLLABUS
• KTU LIVE NOTIFICATION	• SOLVED QUESTION PAPERS

JOIN WITH US

WWW.KTUSPECIAL.IN KTUSPECIAL t.me/ktuspecial1

Arrays

2.1 Introduction

An array is a collection of items stored at contiguous memory locations. The idea is to store multiple items of the same type together. Thus, array is a group of similar (same type/ homogenous) items referenced under a common name. In other words, it is a named list of a finite number of similar data elements.

When we need to store a number of values of same type, arrays come in handy than declaring different variables to hold the values. That is, instead of declaring 50 variables to store 50 numbers, we can declare a single array of size 50 and each element can be accessed using its index.

Arrays are of two types: One-dimensional arrays and multi-dimensional arrays. A ***one-dimensional*** array has one subscript. One-dimensional array is an array which is represented either in one row or in one column. An array having more than one dimension is called ***multidimensional array*** in C language. A two-dimensional array is the simplest form of a multidimensional array.

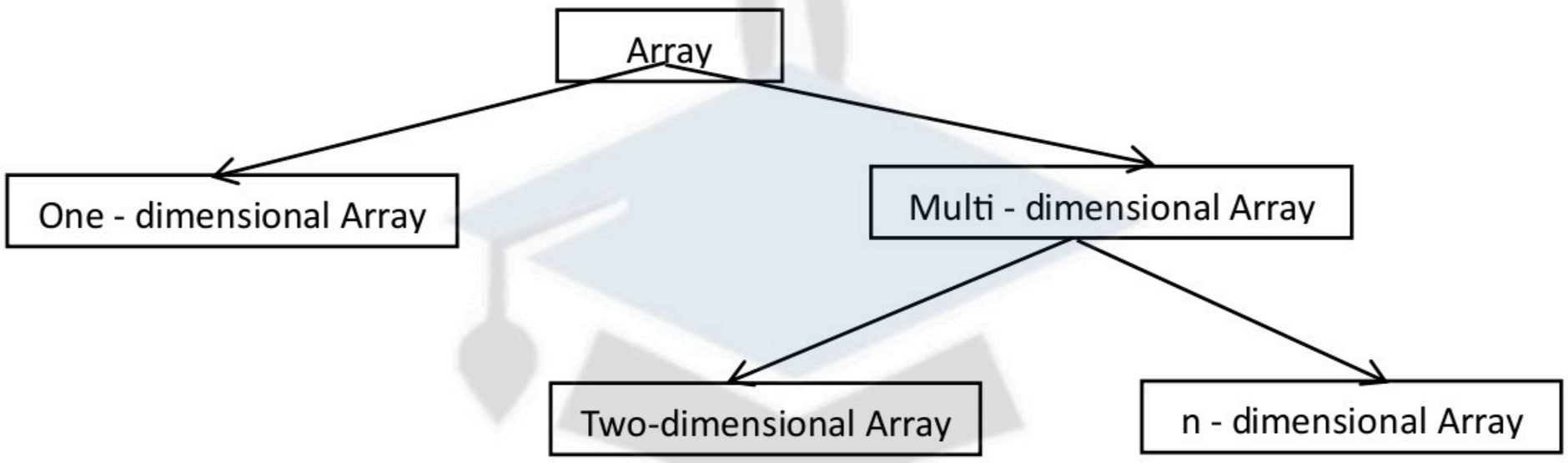

Fig. 2.1 Classification of arrays

2.2 ONE-DIMENSIONAL ARRAYS

One-dimensional array (ID array) can be thought of as a list of variables/values of similar data type. That is, it comprises of finite number of homogeneous elements. They are also called *single dimensional arrays.*

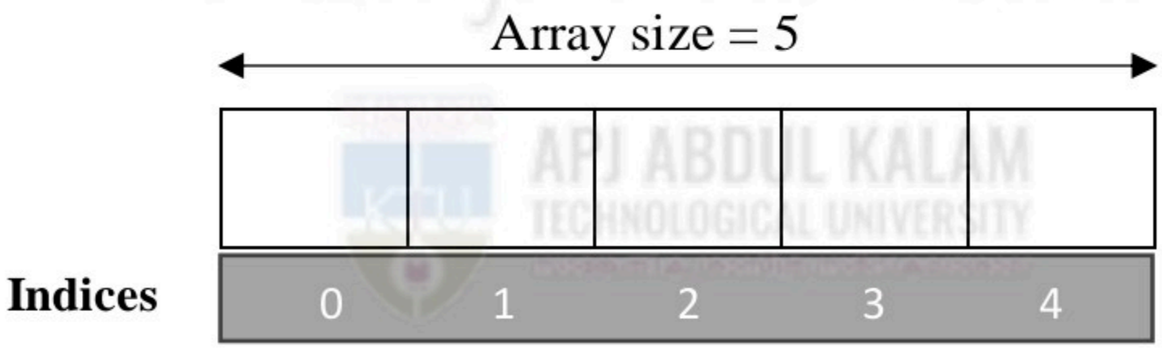

Fig.2.2 1-dimensional array representation

Each item in an array is called an element, and each element is accessed by its numerical index. For an array of size ‘n’, index ranges from 0 to n-1.

Character arrays refer to a collection of characters stored in the form of arrays. **Strings** are a special type of character arrays terminated by a null character (**\0**).

Index	**0**	**1**	**2**	**3**	**4**	**5**
Variable	H	E	L	L	O	\0
Address	0x23451	0x23452	0x23453	0x23454	0x23455	0x23456

Fig. 2.3 Character array representation

Array Declaration:

The syntax for declaring a single dimensional array in C is as follows:

datatype array name[size];

where datatype' denotes the type of the value to be stored (For e.g int, char etc) and 'array_name' is the name of the array declared. The 'array name' must be a valid identifier, i.e., it should satisfy the rules of declaring an identifier. The 'size' denotes the number of elements the array can hold. It should be an integer value. The size of an array is established when the array is created/declared. After declaration, its size/length is fixed.

e.g

```
int marks[10];
char a[20];
float f[10];
```

Note: If you do not know the exact size of the array (for programs in which user enters the number of elements/size), give a possible approximate value which is optimum (not too small or too large) to the program.

Array Initialization:

Array initialization can be done in one of the three ways.

1. Along with declaration:

Here, the array will be initialized along with the declaration, ie., the value will be given in the declaration statement itself.

syntax:

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

eg:

```
int A[5] = {10, 12, 14, 16, 18};
char name[10] = "Computer";
char name[10] = {'C', 'o', 'm', 'p', 'u', 't', 'e', 'r', '\0'};
```

2. Direct assignment:

Each value will be assigned to each position using the index.

syntax:

```
datatype array_name[size];
array_name[index]=value;
```

e.g.:

```
int a[5];
a[0] = 10
a[1] = 12
a[2] = 14
a[3] = 16
a[4] = 18
```

```
char name[10];
name[0]= 'H';
name[1]= 'E';
name[2]= 'L';
name[3]= 'L';
name[4]= 'O';
name[5]= '\0';
```

3. Unsized array initialization:

C also allows unsized array initialization.

syntax:

```
datatype array_name[ ] = {valuel,value2,...};
```

eg:

```
int val[ ] = {2,3,4,5};
char S[]="FIRST";
```

Accessing elements:

The elements are accessed using its index. For example, a[2] stands for the third element in the array, 'a'. To read/display the elements of an array in the program we will make use of a for loop.

- To read an array,

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

- To print the elements,

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

2.2.1 SEQUENTIAL SEARCH / LINEAR SEARCH

A linear search or sequential search is a method for finding an element within a list. It sequentially checks each element of the list until a match is found or the whole list has been searched.

Eg-: Searching element is 33.

List →

10	14	19	26	27	31	33	35	42	44

10	14	19	26	27	31	33	35	42	44
= 33									

10	14	19	26	27	31	33	35	42	44
	= 33								

10	14	19	26	27	31	33	35	42	44
		= 33							

10	14	19	26	27	31	33	35	42	44
			= 33						

10	14	19	26	27	31	33	35	42	44
				= 33					

10	14	19	26	27	31	33	35	42	44
					= 33				

10	14	19	26	27	31	33	35	42	44
						= 33			

The element is found

```
#include<stdio.h>
int main(){
    int arr[10], i, num, n, pos=-1;
    printf("Enter the array size: ");
    scanf("%d", &n);
    printf("Enter Array Elements: ");
for(i=0; i<n; i++)
{
    scanf("%d", &arr[i]);
}
printf("Enter the number to be searched: ");
scanf("%d", &num);
for(i=0; i<n; i++)
{
    if(arr[i]==num)
    {
        pos = i+1;
        printf("Element %d found at position %d", num, pos);
        break;
    }
}
if (pos == -1)
{
    printf("Number not found..!!\n");
}
return 0;
}
```

Output:

```
Enter the array size: 5
Enter Array Elements: 7 1 2 6 9
Enter the number to be searched: 2
Element 2 found at position 3
```

```
Enter the array size: 5
Enter Array Elements: 7 1 2 6 9
Enter the number to be searched: 8
Number not found..!!
```

2.2.2 BUBBLE SORT

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

We take an unsorted array for our example. Sorted array is the output.

14	33	27	35	10

Bubble sort starts with very first two elements, comparing them to check which one is greater.

14	33	27	35	10

In this case, value 33 is greater than 14, so it is already in sorted locations. Next, we compare 33 with 27.

14	33	27	35	10

We find that 27 is smaller than 33 and these two values must be swapped.

14	33	27	35	10

14	27	33	35	10

Next, we compare 33 and 35. We find that both are in already sorted positions

14	27	33	35	10

Then, we move to the next two values, 35 and 10. We know then that 10 is smaller 35. Hence, they are not sorted.

14	27	33	35	10

We swap these values and find that we have reached the end of the array. After one iteration, the array should look like this

14	27	33	10	35

After the second iteration,

14	27	10	33	35

After the third iteration,

14	10	27	33	35

After last iteration,

10	14	27	33	35

Array is sorted

```
#include<stdio.h>
int main(){
    int n, i, arr[50], j, temp;
    printf("Enter total number of elements: ");
    scanf("%d", &n);
    printf("Enter the elements: ");
    for(i=0; i<n; i++)
    scanf("%d", &arr[i]);
    for(i=0; i<(n-1); i++)
    {
        for(j=0; j<(n-i-1); j++)
        {
            if(arr[j]>arr[j+1])
            {
                temp=arr[j];
                arr[j]=arr[j+1];
                arr[j+1]=temp;
            }
        }
    }
printf("Elements sorted successfully..!!\n");
    printf("Sorted list in ascending order :\n");
    for(i=0; i<n; i++)
        printf("%d\t", arr[i]);
    return 0;
}
```

Output:

Enter total number of elements: 5

Enter the elements: 5 1 8 9 3

Elements sorted successfully..!!

Sorted list in ascending order :

1 3 5 8 9

2.3 TWO-DIMENSIONAL ARRAYS

Two-dimensional (2D) arrays are the simplest form of multi-dimensional arrays and takes a rectangular format with rows and columns. The 2D array in C programming is also known as *matrix*. A matrix can be represented as a table of rows and columns.

	Column 0	Column 1	Column 2
Row 0	**x[0][0]**	**x[0][1]**	**x[0][2]**
Row 1	**x[1][0]**	**x[1][1]**	**x[1][2]**
Row 2	**x[2][0]**	**x[2][1]**	**x[2][2]**

Fig.3.4 2D array representation

Declaring a 2D array:

A two - dimensional array can be seen as a table with 'x' rows and 'y' columns where the row number ranges from 0 to (x-1) and column number ranges from 0 to (y-1).

The syntax of declaring a two-dimensional array is:

datatype array_name[rows] [columns];

e.g.:

```
int arr[5][5];
char str[10][50];
```

For an Array[m][n],

Number of elements = m X n

i.e.,

```
int A[20][10]; //20 rows 10 columns
```

No of elements will be 20 x 10 = 200

Elements in two-dimensional arrays are commonly referred to by lil where 'i' is the row number and 'j' is the column number.

Initializing a 2D array:

There are two ways in which a Two-Dimensional array can be initialized.

- First method:

```
int cube[5][2]={1,1,2,8,3,27,4,64,5,125};
```

- Second method:

```
int cube[5][2]={{1,1}, {2,8},{3,27},{4,64},{5,125}};
```

OR

```
int cube[5][2]={{1,1},
                {2,8},
                {3,27},
                {4,64},
                {5,125}};
```

Accessing the elements in a 2D array:

An element in a Two-dimensional array is accessed using the row index and column index. For example, x[3][1] represents the element in the fourth row and second column of the 2D array named 'x' (Remember, the index starts from 0).

To input/output all the elements of a Two-Dimensional array we can use nested for loops. We will require two for loops: one to traverse the rows and another to traverse columns.

For example,

- To read elements of a 2D array,

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

- To print the elements of a 2D array,

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

Qn) C program to perform the addition of two matrices.

```
#include<stdio.h>
int main()
{
   int m, n, i, j, a[10][10], b [10][10], s[10][10];
   printf("Enter the number of rows & columns : ");
   scanf("%d%d", &m, &n);
   printf("Enter the elements of first matrix\n");
   for ( i = 0; i < m ;i++)
   {
      for ( j = 0; j < n ;j++)
      {
         scanf("%d", &a[i][j]);
      }
   }
   printf("Enter the elements of second matrix\n");
   for ( i = 0; i < m ;i++)
   {
      for ( j = 0; j < n ;j++)
      {
         scanf("%d", &b[i][j]);
      }
   }
   printf("Elements of first matrix:\n");
   for ( i = 0; i < m ;i++)
   {
      for ( j = 0; j < n ;j++)
      {
         printf("%d\t", a[i][j]);
      }
      printf("\n");
   }
   printf("Elements of second matrix:\n");
      for ( i = 0; i < m ;i++)
      {
         for ( j = 0; j < n ;j++)
         {
            printf("%d\t", b[i][j]);
         }
         printf("\n");
      }
      for ( i = 0; i < m ;i++)
      {
         for ( j = 0; j < n ;j++)
         {
            s[i][j] = a[i][j] + b[i][j];
         }
      }
      printf("Resultant matrix:\n");
      for ( i = 0; i < m ;i++)
      {
         for ( j = 0; j < n ;j++)
         {
            printf("%d\t", s[i][j]);
         }
         printf("\n");
      }
   return 0;
}
```

```
Enter the number of rows & columns : 3 3

Enter the elements of first matrix

1 2 3 4 5 6 7 8 9

Enter the elements of second matrix

10 11 12 13 14 15 16 17 18

Elements of first matrix:          Elements of second matrix:

1       2       3                  10      11      12

4       5       6                  13      14      15

7       8       9                  16      17      18

Resultant matrix:

11      13      15

17      19      21

23      25      27
```

Qn) C program to perform the multiplication of two matrices.

```
#include <stdio.h>
int main()
{
   int a[10][10], b[10][10], mult[10][10], r1, c1, r2, c2, i, j, k;
   printf("Enter rows and columns for first matrix: ");
   scanf("%d%d", &r1,&c1);
   printf("Enter rows and columns for second matrix: ");
   scanf("%d%d", &r2, &c2);
   while (c1!=r2)
   {
      printf("\nError! column of first matrix not equal to row of second.");
      printf("Enter rows & columns for first matrix: ");
      scanf("%d%d", &r1, &c1);
      printf("Enter rows & columns for second matrix: ");
      scanf("%d%d", &r2, &c2);
   }
```

```
printf("\nEnter elements of matrix 1: \n");
for(i=0; i<r1; i++)
{
   for(j=0; j<c1; j++)
   {
    printf("Enter element a%d%d: ",i+1,j+1);
     scanf("%d", &a[i][j]);
   }
}
printf("\nEnter elements of matrix 2: \n");
for(i=0; i<r2; i++)
{
   for(j=0; j<c2; j++)
   {
    printf("Enter element b%d%d: ",i+1,j+1);
    scanf("%d",&b[i][j]);
   }
}
printf("Elements of matrix1\n");
for (i=0; i<r1; i++)
{
   for (j=0; j<c1; j++)
   {
      printf("%d\t", a[i][j]);
   }
   printf("\n");
}
printf("Elements of matrix2\n");
for (i=0; i<r2; i++)
{
   for (j=0; j<c2; j++)
   {
```

```
      printf("%d\t", b[i][j]);
   }
   printf("\n");
  }
//Initializing elements of matrix mult to 0.
 for(i=0; i<r1; i++)
 {
   for(j=0; j<c2; j++)
   {
     mult[i][j]=0;
     for(k=0; k<c1; k++)
     {
       mult[i][j] = mult[i][j] + a[i][k] *b[k][j];
     }
   }
 }
 printf("Resultant matrix:\n");
 for (i=0; i<r1; i++)
 {
    for (j=0; j<c2; j++)
    {
      printf("%d\t", mult[i][j]);
    }
    printf("\n");
 }
 return 0;
}
```

Output:

Enter rows and columns for first matrix: 3 2

Enter rows and columns for second matrix: 2 3

Enter elements of matrix 1:

Enter element a11: 1

Enter element a12: 2

Enter element a21: 3

Enter element a22: 4

Enter element a31: 5

Enter element a32: 6

Enter elements of matrix 2:

Enter element b11: 7

Enter element b12: 8

Enter element b13: 9

Enter element b21: 1

Enter element b22: 2

Enter element b23: 3

Elements of matrix1

1 2

3 4

5 6

Elements of matrix2

7 8 9

1 2 3

Resultant matrix:

9 12 15

25 32 39

41 52 63

Qn) C program to perform the transpose of a matrix.

```
#include <stdio.h>
int main(){
    int a[10][10], transpose[10][10], row, column, i, j;
    printf("Enter rows and columns of matrix: ");
    scanf("%d%d", &row, &column);
    printf("\nEnter elements of matrix: \n");
for (i=0; i<row; i++)
{
    for (j=0; j<column; j++)
    {
        printf("Enter element a%d%d: ", i+1, j+1);
        scanf("%d", &a[i][j]);
    }
}
printf("Elements of matrix\n");
for (i=0; i<row; i++)
{
    for (j=0; j<column; j++)
    {
        printf("%d\t", a[i][j]);
    }
    printf("\n");
}
for (int i=0; i<row; i++)
{
    for (int j=0; j<column; j++)
    {
        transpose[j][i] = a[i][j];
    }
}
printf("\nTranspose of Matrix: \n");
```

Output:

```
Enter rows and columns of matrix: 3 3
Enter elements of matrix:
Enter element a11: 1
Enter element a12: 2
Enter element a13: 3
Enter element a21: 4
Enter element a22: 5
Enter element a23: 6
Enter element a31: 7
Enter element a32: 8
Enter element a33: 9
Elements of matrix
1       2       3
4       5       6
7       8       9
Transpose of Matrix:
1       4       7
2       5       8
3       6       9
```

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

2.4 STRINGS

In C programming, strings are actually one-dimensional array of characters terminated by a **null character '\0'**. Strings are enclosed in double quotes.

e.g.:

```
char str[30]={'H', 'E', 'L', 'L', 'O', '\0'};
```

OR

```
char str[30]= "Hello";
```

Note-: Here, the C compiler automatically places the '\0' at the end of the string when it initializes the array.

Note: When declaring a string, always give a size slightly more than the required size (atleast 1 byte more, as the last position will be obtained by null character.)

Accessing the elements in string

To read a string	To display a string

#include <stdio.h>
int main() {
  char name[30];
  printf("Enter your name: ");
  scanf("%s", name); // Read a string
  printf("Hello, %s", name);
  return 0;
}
	printf("%s", name);

2.4.1 String Handling Functions

C supports a wide range of functions that manipulate null-terminated strings, **<string.h>** is the header file required for string functions. Some of the string handling functions are:

Function	Purpose
strcpy(dest,src)	Copies src into desc
strlen(str)	Returns the length of the string
strcat(dest,src)	Appends src to dest
strcmp(str1,str2)	Compare two strings (returns 0 if equal)
strrev(str)	Reverse the string
strchr(str,ch)	Finds first occurrence of a character
strstr(str1,str2)	Finds substring in a string
strlwr(str)	Converts the string into lowercase
strupr(str)	Converts the string into uppercase

i. strcpy()

strcpy() is a standard library function in C and is used to copy one string to another.

syntax**:**

strcpy(dest, src);

```
#include <stdio.h>
#include <string.h>
int main() {
    char source[50], destination[50];
    printf("Enter a string: ");
    scanf("%s", source); // Read a single word
    strcpy(destination, source); // Copy string
    printf("Copied string: %s\n”, destination);
    return 0;
}
```

Output:

```
Enter a string: programming
Copied string: programming
```

ii. strlen()

It returns the length of a string. However, one can also write a program manually to find out the length of any string, but the use of this direct function can save our time.

syntax:

variable= strlen(string_name);

```
#include <stdio.h>
#include <string.h>
int main() {
    char str[100];
    int len;
    printf("Enter a string: ");
    scanf("%s",str);
    len=strlen(str);
    printf("Length of the string: %d",len );
    return 0;
}
```

Output:

```
Enter a string: Hello
Length of the string: 5
```

iii strcat()

The strcat() function will append a copy of the source string to the end of destination string. It will append copy of the source string in the destination string. The terminating character at the end of destination is replaced by the first character of source. The strcat() function returns destination the pointer to the destination string.

syntax:

strcat(dest, src);

```
#include <stdio.h>
#include <string.h>
int main() {
    char str1[50] = "Hello ";
    char str2[50] = "World";
    strcat(str1, str2);  // Appends str2 to str1
    printf("Concatenated String: %s”,str1);
    return 0;
}
```

Output:

```
Concatenated String: Hello World!
```

iv. strcmp()

This function takes two strings as arguments and compares these two strings alphabetically. strcmp() compares the two strings that means it starts comparison character by character starting from the first character until the characters in both strings are equal or a NULL character is encountered. It returns an integer value based on the comparison of strings (Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.).

syntax:

variable = strcmp(string1,string2);

```
#include <stdio.h>
#include <string.h>
int main() {
    char str1[50], str2[50];
    printf("Enter first string: ");
    scanf("%s", str1);
    printf("Enter second string: ");
    scanf("%s", str2);
    if (strcmp(str1, str2) == 0)
        printf("Strings are equal.\n");
    else
        printf("Strings are not equal.\n");
    return 0;
}
```

Output:

```
Enter first string: World
Enter second string: World
Strings are equal.
```

v. strrev()

The strrev(string) function returns reverse of the given string.

syntax:

strrev(string_name);

```
#include <stdio.h>
#include <string.h>
int main() {
    char str[] = "Hello, World!";
```

Output:

```
Original String: Hello, World!
Reversed String: !dlroW ,olleH
```

```
    printf("Original String: %s\n", str);
    printf("Reversed String: %s\n", strrev(str));
    return 0;
}
```

vi. strchr()

Returns a pointer to the first occurrence of specified character in a string.

syntax:

strchr(string_name, character);

vii. strstr()

Returns a pointer to the first occurrence of a string in another string.

syntax:

strstr(string1, string2);

viii. strlwr()

Converts the given string to lowercase.

syntax:

strlwr(string_name);

xi. strupr()

Converts the given string to uppercase.

syntax:

strupr(string_name);

x. gets()

The gets() function enables the user to enter some characters followed by the enter key. All the characters entered by the user get stored in a character array. The null character is added to the array to make it a string. The gets() allows the user to enter the space-separated strings.

syntax:

gets(string_name);

e.g.:

gets(s1);

xi. puts()

The puts() function is very much similar to printf() function. The puts() function is used to print the string on the console which is previously read by using gets() or scanf() function. (It prints an additional newline character with the string, which moves the cursor to the new line on the console.)

syntax:

puts(string_name);

e.g.:

puts(s1);

```
#include<stdio.h>
#include<string.h>
int main() {
   char str1[20]="Welcome";
   char str2[20],str3[20];
   int len;
   strcpy(str3,str1);
   puts(str3);
   printf("Enter second string: ");
   gets(str2);
   strcat(str1,str2);
   puts(str1);
   len = strlen(str1);
   printf("Length = %d",len);
   return 0;
}
```

Output:

```
Welcome
Enter second string: Sachin
Welcome Sachin
Length = 12
```

2.5 ENUMERATED DATA TYPE

An enum (enumeration) is a user-defined data type in C. It is used for assigning names to group of constants. **'enum'** is the keyword used to define enumerations. An enum helps improve code readability and maintainability by assigning meaningful names to constants instead of using numeric values directly. In other words, the enum is used when the variable has only a set of values.

Syntax:

enum name

{

memberName1 = <integral constant>,

memberName2,

memberName3,

. . .

memberNameN

};

e.g.:

enum week {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};

Why And When to Use Enums?

- Enums are used to give names to constants, which makes the code easier to read and maintain.
- Use enums when you have values that you know aren't going to change, like month days, days, colors, deck of cards, etc.

If we do not explicitly assign values to enum week, the compiler by default assigns values starting from 0. For example, sunday gets a value of 0, monday gets 1, and so on. We can assign values to specific names in any order. All unassigned names acquire the value of the previous name plus one. For example,

enum week {sunday = 1, monday, tuesday = 21, wednesday, thursday 11, friday, Saturday};

Then, moday will take the value 2, wednesday will take the value 22, friday will get 12 and saturday 13. So,

```
#include <stdio.h>
int main()
{
enum week {sunday = 1, monday, tuesday = 21, wednesday, thursday = 11, friday,
```

```
saturday};
printf("%d %d %d %d %d %d %d", sunday, monday, tuesday, wednesday, thursday,
friday, saturday);
return 0;
}
```

Output:

1 2 21 22 11 12 13

Enums are treated as integers, so you can assign and compare them with integer values. The size of an enum in C is typically the size of an int (usually 2 bytes), but it depends on the compiler. Enums are commonly used in switch statements to improve readability:

Eg-:

```
#include <stdio.h>
enum week{sunday=1, monday, tuesday, wednesday, thursday, friday, saturday};
int main()
{
   enum week d;
   d=monday;
   switch(d)
   {
      case sunday:
         printf("Today is sunday");
         break;
      case monday:
         printf("Today is monday");
         break;
   case tuesday:
         printf("Today is tuesday");
         break;
   case wednesday:
         printf("Today is wednesday");
```

```
        break;
    case thursday:
        printf("Today is thursday");
        break;
    case friday:
        printf("Today is friday");
        break;
        case saturday:
            printf("Today is saturday");
            break;
    }
return 0;
}
```

Output:

Today is monday

2.5.1 C typdef

The typedef is a keyword that is used to provide existing data types with a new name. The C typedef keyword is used to redefine the name of already existing data types. When names of datatypes become difficult to use in programs, typedef is used with user-defined datatypes, which behave similarly to defining an alias for commands.

Syntax of typedef-:

typedef existing_type new_type;

where,

- **existing_type:** The type that we want to alias (e.g., int, float, struct, etc.).
- **new_type:** The new alias or name for the existing type.

Eg-:

```
#include <stdio.h>
typedef int Integer;
int main() {
    // n is of type int, but we are using
      // alias Integer
    Integer n = 10;
```

Output:

10

```
    printf("%d", n);
    return 0;
}
```

2.5.2 Applications of enum

- In many applications, you might need to represent different states of a system, such as the states of a traffic light, a game, or a process in a workflow. Enums make it easier to handle such states.
- Enums are commonly used to represent various error codes, return statuses, or result flags.
- When working with commands or different actions in a system, enums are a great way to represent them. This is especially useful in applications like menu-driven programs, controllers, or network communication.
- Enums are commonly used to represent days of the week, months of the year, and other cyclic or ordinal data.
- Enums are commonly used for bitwise flags or permissions, such as in file system access control, configuration settings, or enabling/disabling features.
- Enums can be used to define different modes of operation in networking, such as different protocols (TCP, UDP) or communication types.
- In game development, enums are often used to represent various character states, such as movement states, actions (e.g., jumping, running), or game modes (e.g., menu, playing,paused).