MODULE -2

ARRAYS

- An array is a collection of similar data elements of **same data type.**
- The elements of the array are stored in consecutive memory locations and are referenced by an **index** (also known as subscript)
- Subscript indicates the number of elements counted from beginning of the array.
- An array allows us to use a single variable name to store a numbers of values with the same data type.

Declaration of array

✓ An array must be declared before being used, it consists of 3 things

- **Data type** → what kind of values it can store, **ex:** int, float, char, double
- **Name** → The name of the array, to identify the array.
- **Size** → The maximum number of values that the array can hold.

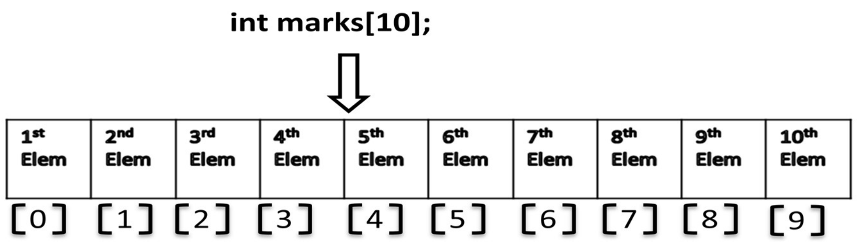
Syntax: datatype arrName[size];

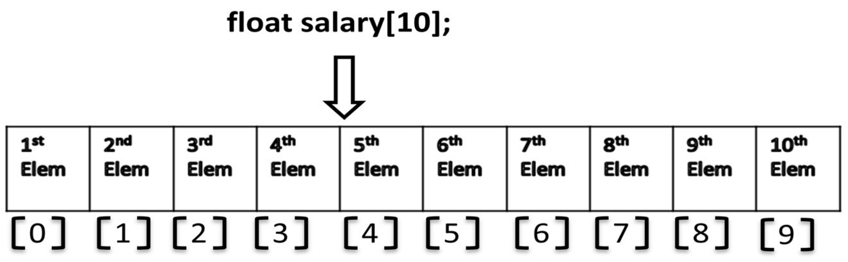
Ex: An array called ***marks*** that can hold integers is declared as follows:

int marks [10];

- The 10 specifies the number of elements in the array (size).
- Array ***marks*** has ten elements, numbered from 0 to 9.
- 0 is the lower bound or smallest subscript.
- 9 is the upper bound or largest subscript.
- Here the subscript which refers to marks should be in the range from 0 to 9.
- The type is the array declaration tells what data type applies to each element, from the lower to the upper bound, here it is ***int.***
- It allocates 10 consecutive memory locations each of 2 bytes, total 20 bytes of memory.
- The first element is stored in marks [0], the second element is marks [1] and so on. The last element will be stored in marks [9].

Note: 0, 1, 2, 3, …. Are written in square brackets are **index/subscripts**.

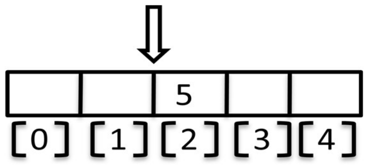




USING ARRAYS

- The elements of the array can be treated like simple variables.
- Each element can be assigned a value, tested, printed or used in an expression.

Ex:1) `int num[2] = 5;`



Here the value 5 is been assigned to the memory location identified by index 2

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

Here it prints the value of an element which is present at the location or index 2 ie it prints 5.

3) `result = 2 * num [0];`

Here the value present in location 0 (zero) is been multiplied by 2 and assigned to variable called result.

4)
```
i = 4;
num [i] = 7;
```

Here the variable I is set to 4 by the first line. The reference to second line the subscripted variable num [i] really means num [4], and that storage location is set to 7.

5) `for ( i = 0; i < 5; i++ )`

```
num[i] = i * 3;
```

it assigns each element of array num a value which is three times the subscript; that is num[0] gets the value 0, num[i] gets 3, num[2] gets 6, num[3] gets 9 and num[4] gets 12.

Initialization of Array

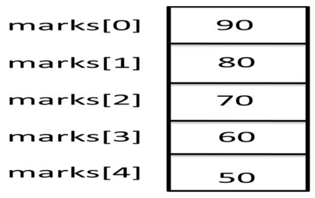
- ✓ Elements of the array can be initialized at the time of declaration
- ✓ When an array is initialized, we need to provide a value for every element in the array.
- ✓ The syntax is:

```
type arr_name [size] = {list of variables};
```

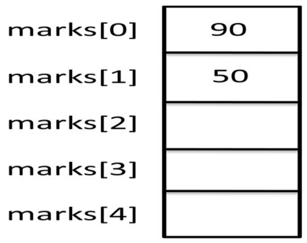
- ✓ The values are written with curly brackets and every value is separated by a comma.

Ex: 1) `int marks [5] = {90, 80, 70, 60, 50};`

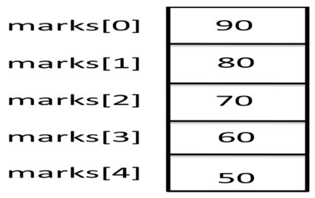
is stored as :



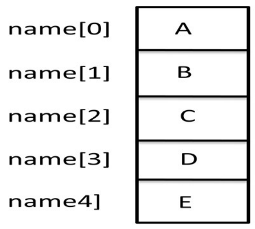
2) `int marks[5] = {90, 50};`



3) `int marks [ ] = {90, 80, 70, 60, 50};`



4) `char name[5] = {‘A’,’B’,’C’,’D’,’E’};`



Note: If we have more initializes than the declared size of the array, then a compile time error will be generated.

For ex: `int marks [3] = {1, 2, 3};`

- ✓ Since only 3 memory locations are allocated and 4 values are been initialized, so there is no 4th location to store the 4th value, so it generates a compile time error.

Using a constant as the Array Size

We can also use a constant as the size of the array to modify the size, we write as,

```
# define NUM 150
int test [NUM];
```

- ✓ The value 150 is substituted for NUM prior to setting up storage for the array, making this a legal declaration.

Inputting (reading) the values of Array

- ✓ An array can be filled by inputting values from the keyboard
- ✓ In this method, a while / do-while or a for loop is executed to input the value for each element of the array.

Ex: /* inputting values of each element of the array */

```
        int i, marks [10];
        for ( i=0;i<10;i++ )
        scanf (" % d", &marks[i]);
```

- ✓ In the code we start with the index **i** at 0 and input the value for the first element of the array. Since the array can have 10 elements, we must input values for elements whose index varies from 0 to 9.
- ✓ So therefore in the for loop, we test for condition (i < 0) which means the number of elements in the array.

Outputting (displaying) the values of Array

- ✓ An array can be displayed on to the output unit
- ✓ In this method, a while / do-while or a for loop is executed to output the value for each element of the array.

Ex: /* outputting values of each element of the array */

```
        for ( i=0;i<10;i++ )
        printf (" % d", marks[i]);
```

- ✓ In the code we start with the index **i** at 0 and output the value for the first element of the array. Since the array can have 10 elements, we must output values for elements whose index varies from 0 to 9.
- ✓ So therefore in the for loop, we test for condition (i < 0) which means the number of elements in the array.

/* program to read and display the contents of an array */

```
    #include<stdio.h>
    void main ( )
    {
    int i, marks [10];
    printf(" Enter the array elements \n");
    for ( i=0;i<10;i++ )
         scanf (" % d", &marks[i]);
    printf(" The entered array elements  are \n");
         for ( i=0;i<10;i++ )
    printf (" % d", marks[i]);
    }
```

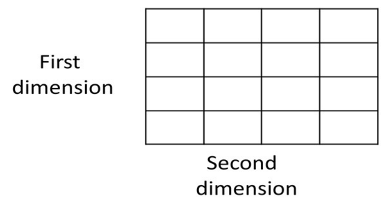
/* program to read N elements and perform sorting using Bubble sorting */

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int i,j,n;
int a[100];
clrscr();
printf("enter size\n");
scanf("%d",&n);
     printf("enter array\n");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
     printf("Given array");
for(i=0;i<n;i++)
 printf("\n%d",a[i]);
          for(i=0;i<n-1;i++)
     {
          for(j=n-1;j>i;j--)
          {
                     if(a[j]<a[j-1])
                    {
          temp=a[j];
          a[j]=a[j-1];
          a[j-1]=temp;
                     }
          }
}
printf("\nThe sorted array is\n");
for(i=0;i<n;i++)
```

```
    printf("%d\n",a[i]);
 getch();
}
```

TWO- DIMENSIONAL ARRAYS

- ✓ A 2D array is specified using two subscripts where one subscripts denotes row and the other denotes column.
- ✓ 2D is known as array of array.



Declaration of 2D Arrays

- ✓ 2D arrays must be declared before being used.
- ✓ The declaration statement tells the compiler the name of the array, the data type of each element in the array, and the size of each dimension.
- ➢ A 2D array is declared as:

data_type array_name [row_size] [column_size];

- ✓ Therefore a two dimensional m*n array is an array that contains m*n data elements and each element is accessed using two subscripts, i and j where i <=m and j<=n

Ex: int marks [3] [5];

- ✓ A two dimensional array called marks is declared that has m(3) rows and n(5) columns.
- ✓ The first element of the array is denoted by marks [0] [0], the second element as marks [0] [1], and so on.
- ✓ Here, marks [0] [0] stores the marks obtained by the first student in the first subject, marks[1] [0] stores the marks obtained by the second student in the first subject, and so on.

Row/Column	**Column 0**	**Column 1**	**Column 2**	**Column 3**	**Column 4**
Row 0	marks[0][0]	marks[0][1]	marks[0][2]	marks[0][3]	marks[0][4]
Row 1	marks[1][0]	marks[1][1]	marks[1][2]	marks[1[3]	marks[1][4]
Row 2	marks[2][0]	marks[2][1]	marks[2][2]	marks[2][3]	marks[2][4]

Initialization of Two-dimensional Arrays

- ✓ The initialization of 2D array is done row by row. This statement5 is written as:

int marks [2] [3] = { {90,87,78} , {68,62,71}};

- ✓ The given 2D array has two rows and three columns. Here, the elements in the first row are initialized first and then the elements of the second row are initialized.

```
marks [0] [0] = 90  marks [0] [1] = 87  marks [0] [2] = 78
marks [1] [0] = 68  marks [1] [1] = 62  marks [1] [2] = 71
```

- ✓ In two dimensional array we can omit the first dimension. Therefore the following declaration is valid:

int marks [ ] [3] = { {90,87,78} , {68,62,71}};

- ✓ If some values are missing in the initialize then it is automatically set to zero.

int marks [2] [3] = { {90,87,78} };

In order to initialize the entire 2D array to zero, simply specify the first value as zero.

int marks [2] [3] = { 0};

Inputting (reading) the values of 2D Array

- ✓ An 2D array can be filled by inputting values from the keyboard
- ✓ In this method, a while / do-while or a for loop is executed to input the value for each element of the array.

Ex: /* inputting values of each element of the 2D array */

```
int i,j, marks [10];
for ( i=0;i<10;i++ )
for ( j=0;j<10;j++ )
scanf (" %d", &marks[i][j]);
```

Outputting (displayng) the values of 2D Array

- ✓ An 2D array can be displayed on to the output unit
- ✓ In this method, a while / do-while or a for loop is executed to output the value for each element of the array.

Ex: /* outputting values of each element of the 2D array */

```
for ( i=0;i<10;i++ )
for ( j=0;j<10;j++ )
printf (" % d", marks[i][j]);
```

/* program to print elements of 2D array*/

```
#include<stdio.h>
#include<conio.h>
void main( )
{
int i,j;
int marks [2][2] = {12, 34, 56, 32};
     for ( i=0;i<2;i++ )
     {
          for ( j=0;j<2;j++ )
          {
          printf (" % d", marks[i][j]);
          printf("\n");
          }
     }
}
```

Output:

```
12 34
56 32
```

STRINGS

- ✓ A string is a sequence of elements of the char data type.
- ✓ String can contain any characters including special characters like newline, tab, etc. which are typed as two symbols (\n, \t) but stored as a single character in a memory.

Literal string: Is a sequence of characters enclosed by double quotation mark,

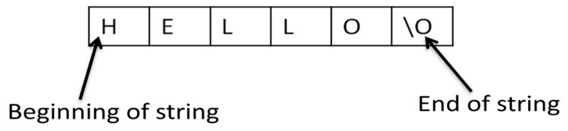
Ex: **`"Welcome to CCP"`**

- ✓ It is a string constant, and character constant can be enclosed by double quotation.

Storage for Literal String

- ✓ A string is array of characters that must end with null character.

Ex: `printf ("HELLO " );`



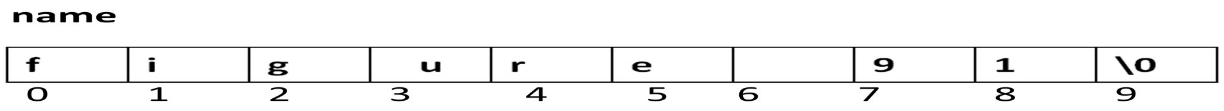
- ✓ Whenever we enter any string then the compiler stores string in consecutive memory location and automatically terminate the string by null character.
- ✓ Declaring string variables
- ✓ A string is declared like an array of characters, we can declare string variable that can hold upto 10 characters.

```
char name[10];
```

- ✓ Here the array variable is name; number (11) in brackets is the size or maximum length of the variable. ie size is 11 means it can store up to 10 characters + 1 null character to terminate a string.

✓ In a name string we can store letter, number, blank, punctuation mark and so on

Ex:



1) **NULL character:**

   The NULL character marks the end of the string.

2) **String size and length**

✓ In the declaration of string, an array of characters allocates a specific number of bytes called as size of the string, ie name [10] means name has size 10.

✓ A string also has a length which is different from its declared size

✓ Name string length is 5 excluding NULL character (\0).

✓ Individual positions in a string are numbered, starting from 0, and can be accessed individually,

ie name[0] has value h

name[4] has value o

name[5] has value \0

Initialization during Declaration

✓ Array of characters (string) can be initialized at the time of declaration.

Method 1: Initialize string as an array of characters and assign values to the individual position

```
char first[10] = { ‘t’, ‘a’, ‘b’, ‘l’, ‘e’, ‘\0’ }
```

And assign NULL character at the end of the string.

Method 2: Initialize the string as a unit and NULL character is supplied by the compiler automatically.

```
char second [10] = “table”;
```

Method 3: Initializing a string in the code (in statement section)

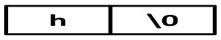
We can initialize explicitly each character value in statement section and insert NULL character at the end.

Ex:
```
char name [6];
    name [0] = ’h’;
    name [1] = ’e’;
    name [2] = ’l’;
    name [3] = ’l’;
    name [4] = ’o’;
    name [5] = ’\0’;
```

Legal chatracters

- ✓ A string literal is enclosed by a double quotation mark and character constant is enclosed ny a single quotation marks so both double quotes and apostrophe (‘) are legal characters in a string.

Ex: “h”;



Here it requires 2 memory locations

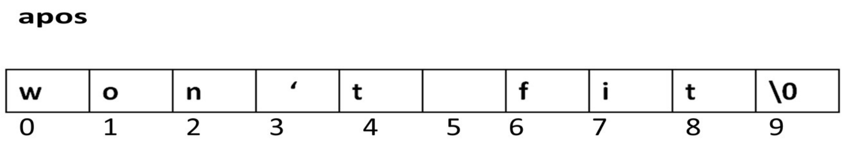
Ex: ‘h ‘;

Here it requires only 1 memory locations

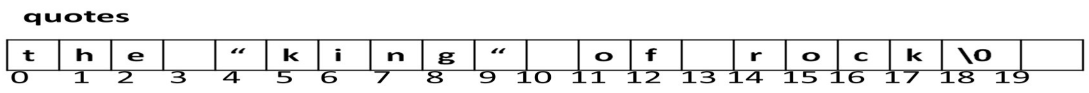
Escape character (\ )

- ✓ If we want to make “is one of the character then we need to use escape character followed by double quotes (\”) it tells to the compiler that it is a character but not delimiter, same thing is done with apostrophe (‘) and backslash.

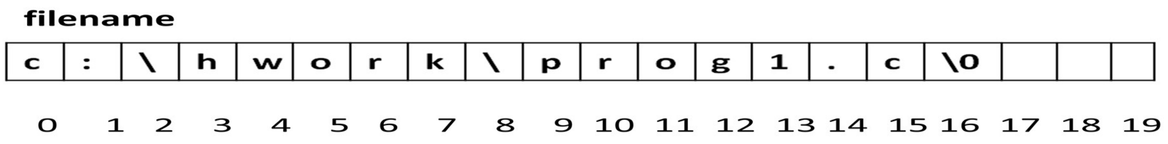
Ex: 1) `char apos[10] = “won’t fit”;`



Ex: 2) `char quotes[20] = “the \ “king\” of rock”;`



Ex: 3) `char filename[20] = “c:\\hwork\\prog1.c”;`



Printing a string

- ✓ Printf statement can be used to print a string with all the characters of string but not NULL character and use %s conversion specification for a string variable.

Ex:
```
char str[5] = " CCP ";
    printf ( "%s is",str);
    printf ( " the Best " );
```

Output: CCP is the Best.

- ✓ First printf statement prints literal string is followed by the value of **str.**
- ✓ 2nd printf statement prints another literal string the Best but it does not contain variable name or conversion specification (%s).

Reading string using scanf

- ✓ We are using scanf statement to read a string and **%s** conversion specification, in case of strings.
- ✓ scanf does **not** need **&** symbol, it is optional.

Ex:
```
char str[5];
    printf ( " Enter string: " );
    scanf ( "%s ",str);
```

Problem with using scanf

- ✓ There are a few problems with reading string using scanf:

1) The scanf function stops reading at the first whitespace character(spaces, tabs and newline)

   Ex: `"Girish Kumar"`

- ✓ The scanf reads only **Girish** but **Kumar** is not stored
- ▪ One solution to this problem is to read in each part of the string as a separate unit.

```
char first[20];
char last[20];
printf (" Enter a first name up to 20 characters");
scanf("%s",first);
printf (" Enter a last name up to 20 characters");
scanf("%s",last);
```

- Here if the user types in the name, **“Girish”** is stored in **first** and **“Kumar”** is stored in **last.**

2) The most serious problem with reading in a string using scanf is that there is no way to set the limit on the size of value read in. The scanf function reads until it finds a whitespace character; if it reads more character than the size, then those characters are stored in adjacent memory locations, even if they overwrite existing values of the other variables.

Note: when typing a string data, never enter a value which is longer than the number of characters in the variable’s declaration minus 1. If you do so, there will be no place to store the NULL character within the variable, and the result is unpredictable.

- ✓ The alteration for scanf is **gets( )** and **puts( ).**

STRING MANIPULATION FUNCTIONS FROM THE STANDARD LIBRARY

Functions from the standard library

The standard library ***string.h*** contains many functions for string manipulation. Among them are the following:

- ✓ **strcpy:** which allows us to give a string an initial value as a unit, rather than character by character.
- ✓ **strlen:** which allows us to determine the length of a string.
- ✓ **strcmp:** which allow us to compare one string with another.
- ✓ **strcat:** which allows us to combine two strings.
- ✓ **strncpy:** which allow us to copy a block of n characters from one string to another.
- ✓ **strncmp:** which allow us to compare a block of n characters from one string to another.

Note: To use these library functions, we must include the ***string.h*** header file in our program.

Assigning a Value to a String Variable: strcpy

- ✓ Strcpy function is used to assign a value to a string. The strcpy function lets us to copy a string from one location to another.

- ✓ The string to be copied can be literal string or a string already stored in a array of char. The strcpy function will copy the entire string including the terminating NULL character to the new location.

The general form of a call to strcpy is :

```
strcpy ( dest, source );
```

- ✓ The strcpy function has two parameters; both are pointers to strings. The first is the destination where the value will be stored and the second is the source the string literal or variable which is going to be copied to the destination.

Ex:
```
char first[20];
     char last[20];
     strcpy ( first, "Girish Kumar");
     strcpy (last,first);
```

- ✓ After the two calls to strcpy, first and last each will have the value **"Girish Kumar\0".**
- ✓ Determining the length and size of a string: strlen and sizeof
- ✓ The strlen function can be used to find the length of the string in bytes.
- ✓ The sizeof operator can be used to determine the declared size of a string. It accepts one operand, a variable or a type, and returns the number of bytes allocated to it.

The general form of a call to strlen is :

```
Length = strlen(str );
```

- ✓ The parameter to ***strlen, str***, is a string; the return value, length is an integer representing the current length in bytes, excluding the NULL character.
- ✓ **For ex:** if **str** has the value "hello", **strlen** returns 5.If str has the value **"\0",** strlen returns 0.

<u>Note:</u> The length does not depend upon the number of characters specified in the declaration for str.

The general form of a call to sizeof is

```
size = sizeof (typename );
```

- ✓ The sizeof operator takes a single operand, which can be a type or a variable.
- ✓ The result is assigned to size, is an integer representing the size of the operand in bytes.

- ✓ If itme is the name of a variable, including an array, sizeof returns its size in bytes;
- ✓ **Ex:** if item has been declared char item[30], size of returns 30. If the operand is the name of a type, sizeof returns the number of bytes allocated to that type in the system.

Ex:
```
int length,size;
    char dest[20];
    char source[20];
    scanf("%s",source);
    length = strlen(source);
    size = sizeof(dest);
         if ( length < size)
               strcpy(dest,source);
        else
               printf("won't fit");
```

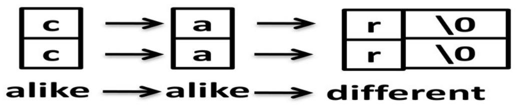
- ✓ Here using strlen allows checking the length of source before copying it to **dest.** If the **source** is too long, the copy operator is not performed. Another function, **strcpy** can be used to copy only part of a string.

<u>Comparing two string: strcmp</u>

- ✓ The compiler compares pair of characters one from each string, until it finds a pair that are not the same. If the first characters are alike, it looks at the next character in each string, until it gets to the end of one of the strings or finds a pair of character in each alike.
- ✓ Thus when comparing **"car"** with **"cat",** it finds the first two characters are alike; then it determines that "car" is less than "cat", because **'r'** is less than**'t'.**
- ✓ Normally the length of the string is not a factor because the character by character comparison starts at the left end.

<u>Characters and their Numeric Equivalent</u>

Character	**'\0'**	**' '**	**'A'**	**'B'**	**'Z'**	**'a'**	**'b'**	**'c'**	**'z'**
ASCII value	**0**	**32**	**65**	**66**	**90**	**97**	**98**	**99**	**122**



’r’ <‘t’, so str1 < str2

- ✓ Notice that **“cat”** and **“Cat”** are different strings since uppercase and lowercase letters have different numeric values. **“Cat”** is less than **“cat”** because uppercase letters have smaller numeric values.

The general form of a call to strcmp is

```
result = strcmp (first, second );
```

- ✓ The function **strcmp** returns an integer determined by the relationship between the strings pointed by two parameters. While the result is an integer, we usually don’t care about its exact value, just whether it is **positive, negative or 0.**

```
result  > 0 if  first  >  second
result  = 0 if  first  = =  second
result  < 0 if  first  <  second
ex: int result;
      result = strcmp (“cat”, ”car”);   /*  result  >  0  */
      result = strcmp (“big”, ”little”); /*  result  <  0  */
      result = strcmp (“ABC”, ”abc”);  /*  result  <  0  */
      result = strcmp (“ab”, ”abc”);  /*  result  <  0  */
      result = strcmp (“pre”, ”prefix”);  /*  result  <  0  */
      result = strcmp (“potato”, ”pot”);  /*  result  >  0  */
      result = strcmp (“cat”, ”cat”);    /*  result  = =  0  */
```

<u>**Concatenation : Joining two strings**</u>

- ✓ The **strcat** function is used to combine two strings and the resulting string has only one NULL character at the end.

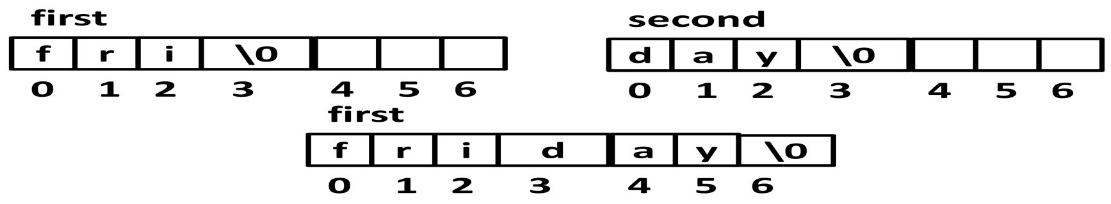
The general form of a call to strcat is

```
strcat ( first, second );
```

- ✓ After the call, first contains all the characters from first, followed by the ones from second up to and including the first NULL character in second.

- ✓ The strcat function stops copying when it finds a NULL character. If the second string does not contain NULL character, strcat continues copying bytes from memory until it finds a NULL character, no matter how far away.
- ✓ Strcat function does not check to see whether there is room for the resulting string at the specified location. Therefore the programmer must check to make sure that the resulting string fits in the variable.

Ex:
```
char first[7] = " fri ";
   char second[7] = " day ";
   strcat ( first, second);
```



SUBSTRING FUNCTION: strncmp and strncpy

The strncmp Function

The general form of a call to strncmp is

```
result = strncmp (first, second, numchars );
```

- ✓ The strncmp function compares up to numchars characters from the two strings. Like strcmp, this function returns an integer representing the relationship between the compared string.
- ✓ Like strcmp, the strncmp function stopos at a NULL character. It compares character by character until it finds a NULL character that are different or until it has compared numchars characters.

Ex:
```
char first[30] = "string";
    char second[20] = "stopper";
    if ( strncmp ( first, second, 4) = = 0 )
              printf ("first four characters are alike\n");
   else if ( strncmp ( first, second, 4) < 0 )
```

```
                printf ("first four characters of first string
are less\n");
        else
                printf ("first four characters of first string
are greater\n");
```

- ✓ In this example strncmp compares up to four characters from the string first and second. If the first four match, strncmp returns 0, the condition is true and the message **"first four characters are alike"** is printed.
- ✓ If the first four don't match, strncmp returns a negative or positive value, depending on this value, the appropriate message is printed.

The strncpy Function

The general form of a call to strncpy is

result = strncpy (dest, source, numchars);

- ✓ The strncpy function allows copying just a second of a string. It allows us to extract a substring from one string and copy it to another.
- ✓ The strncpy function takes three parameters: **dest** is pointer to the destination; **source** is a pointer to the source and numchars is an integer that specifies how many characters to copy. This call to strncpy copies up to numchars from **source** to **dest.**

Note: unlike strcpy, strncpy does not automatically append a NULL character. If the extracted string does not end with a NULL character, the programmer has to supply one.

Ex:
```
char source[20] = "computers are fun ";
     char dest[10];
     strncpy ( dest, source, 9 );
    dest[3] = '\0';
    printf (" %s\nn", dest );
```

output: computers

strncpy copies three characters **"computers"** from **source to dest**. Then the NULL character is copied after 's'.

STRING INPUT/OUTPUT FUNCTIONS

Printing a string: puts

- ✓ Although we can print a string with the printf function, it is better o use the puts function, which is dedicated to printing strings. Here is the format of puts function.

```
puts ( str );
```

- ✓ The puts function takes a pointer to a string str as a parameter and sends str to standard output stream. After printing the string, puts prints the newline character, which has the effect of printing entire line.
- ✓ Although puts returns a value (**on success, a non negative value, on error, EOF**), it is usually called as a void function.

Ex:
```
char str[20] = " This is a string to display ";
      puts(str);
```

- ✓ The call to p[uts prints the string below and then goes to a new line:

Output: This is a string to display.

Reading a value into a Character Array: gets

- ✓ Although we can read in a string value using the scanf function, normally scanf stops when it reads a whitespace character.
- ✓ This makes scanf inappropriate for performing a task like reading in an entire line of input (possibly containing whitespace characters)
- ✓ An alternative is the string input function gets.

Here is the format of puts function

```
result = gets ( instring );       or         gets (instring);
```

- ✓ The gets function allows reading in an entire line of input, including whitespace characters. It reads everything up to a newline character (created by pressing ENTER key).
- ✓ The parameter instring is a pointer to a character array, gets returns a value of type character.

Ex:
```
char str[50];
      gets(str);
      puts(str);
```

- ✓ The function gets is called to read a value into **str.** If the call to gets is successful, **str** has a value. Suppose the user types in **"Oh boy!"** followed by pressing ENTER key. In that case the **str** has value: **"Oh boy!",** and puts will print the value which is present in **str** and goes to new line.

problems with gets

- ✓ While gets function is designed to be used with strings, it has a great potential for disaster. The function doesn't check to see whether there is room for the string which is been read in.
- ✓ Unfortunately, this means that even though call to gets function may be successful, the value read in any overwrite and destroy other variables in your program.

Note: when reading in data using gets, make sure that the values entered are shorter than the variable into which they are being read.

WRITING SOME USEFUL STRING FUNCTIONS OF OUR OWN

1) Sending a string as a parameter: length function

```
int length(char str[20]);
```

- ✓ The main program calls length function as follows:

```
int len;
        char str[20];
        gets(str);
        len = length(str);
```

- ✓ Now let's write the length function. In the body, we must look character by character until we find the end of the string (the NULL character). Since I, the index variable of the loop, starts at 0 and increments as we go through the string, it serves as both the index and the counter.

/* length function returns an integer representing length of parameter */

```
int length(char str[20])
    {
     int i = 0;
     while (str[i] != ‘\0’)
     i++;
     return i;
    }
```

- ✓ If **str** has the value “winter”, length returns 6. If **str** is the NULL string “\0”, length returns 0.

String copy operation

```
void strcopy(char str1[100],char str2[100])
{
int i;
for(i=0;str1[i]!='\0';i++)
str2[i]=str1[i];
str2[i]='\0';
return(str2);
}
```

```
// user defined function for comparing 2 strings

void compare(char str1[],char str2[])
{
  int i = 0;
   while (str1[i] == str2[i] && str1[i] != '\0')
      i++;
   if (str1[i] > str2[i])
      printf("str1 > str2");
   else if (str1[i] < str2[i])
      printf("str1 < str2");
   else
      printf("str1 = str2");

   }
```

```
// user defined function for concatinating 2 strings

void concat(char str1[], char str2[])
 {
   int i, j;

   i = length(str1);

   for (j = 0; str2[j] != '\0'; i++, j++)
    {
      str1[i] = str2[j];
   }

   str1[i] = '\0';
}
```

<u>The getchar function</u>

- ✓ The library function specifically designed for input of character values is getchar. It reads a single character (including whitespace character) from the keyboard and returns it.
- ✓ Typically the value returned is assigned to a variable.

Here is the format of getchar function.

C = getchar ( );

- ✓ The function getchar doesn't receive parameters. It returns a single value, which is the character just read in from stdin or else EOF.

Ex:
```
int answer;
       do
         {
           printf("Do u want to continue ? y/n");
           answer = getchar();
         } while (answer = = 'y');
```

The putchar function

- ✓ The library function specifically designed for output of character values is putchar.

Here is the format of getchar function

```
putchar (ch );
```

- ✓ The putchar function takes one parameter- the char value to print (ch) sends it to stdout. It returns a value of type int which is usually ignored.

```
int c, count = 0;
c = getchar( );
while (c != EOF)
{
     putchar (c);
     count++;
     c = getchar( );
}
printf(" there are %d characters in the input", count);
```

ARRAY OF STRINGS

- ✓ An array of strings is an array of arrays, or a two-dimensional array.
- ✓ To declare an array of strings, we must specify **two subscripts:** the number of strings in the array and the number of characters in each string.
- ✓ The first subscript corresponds to the number of **row**s in a two-dimensional array and the second subscript to the number of **columns.**
- ✓ Even though an array of strings is technically a two-dimensional array, it is possible to use elements of the array by specifying only one subscript, as with one-dimensional array.
- ✓ This is because an array of char is treated as a unit, a string.

Reading and Printing an Array of Strings

printf and **scanf** are used for input/output on array of strings.

Ex: The following code initializes and then prints the elements of the **months** array.

```
char months[12][10] = {"January","Febrauary","March","April",

"May","June","July","August",

"September","October","November", "December"};
int i;
for (i = 0; i <= 11; i++ )
printf("%s\n",months[i]);
```

Reading values into an Array of strings

```
char str[10][20];
int i = 0;
while ( scanf("%s", str[i]) != EOF)
{
     printf ("%s",str[i]);
     i++;
}
```

- ✓ The loop reads data into the **str** array, which can hold up to ten strings, each up to 20 characters in length,
- ✓ Each call to **scanf** reads a string into an element of the **str** array.
- ✓ Then the call to **printf** prints the string just stored.
- ✓ Since the call to **scanf** is in the header of the loop, we don't read another.
- ✓ Whenever **scanf** determines that there are no more data values, the loop terminates.

/* Ex: Array of String Program to perform **Binary Search** on N names */

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

```
{
     char  arr[25][20], key[25];
     int i, n, mid, cond;
     clrscr();
     printf("Enter the number of strings you want to enter\n");
     scanf("%d", &n);
     printf("Enter %d strings\n", n);
     for (i=0; i<n; i++)
     {
   scanf("%s", arr[i]);
     }
     for(i=0; i<n; i++)
     {
   printf("%s ", arr[i]);
     }
     printf("\n");
     printf("Enter string to be searched\n");
     scanf("%s", key);
     i = 0;
     n = n-1;
     mid = (i+n)/2;
     while(i <= n)
     {
   mid = (i + n) / 2;
   if((cond = strcmp(arr[mid], key)) == 0)
   {
   printf("Key found at %d position", mid+1);
   getch();
         exit(0);
         }
         else if(cond < 0)
```

```
            i = mid + 1;
        else
            n = mid - 1;
    }
    printf("String is not found\n");
    getch();
    return 0;
}
```

Assignment Questions

1. Explain how to declare and initialize 1-D array
2. Explain how to declare and initialize 2-D array
3. Write a 'C' program that reads '***N***' integer numbers and arrange them in ascending order using ***Bubble Sort*** technique
4. Write a C program to implement ***Binary search*** on 'n' numbers using 1D Array.
5. What is a string? Explain how strings are declared and initialized.
6. Explain any five string manipulation library functions in 'C' with their syntax
7. Write a 'C' program to implement string manipulation function(length(), strcpy(), strcmp(), strcat()) without using built in functions

****************************END OF MODULE – 2** ***************************