

## **Lab 7: STRING HANDLING**

### **OBJECTIVES:**

- ✓ To be familiar with the operations on two-dimensional arrays.
- ✓ To know about the way of passing multidimensional arrays using functions.
- ✓ To be familiar with the standard library string handling functions.
- ✓ To be able to perform basic string manipulations with as well as without using the standard string handling library functions.

### **THEORY:**

String manipulation is often a very large and important part of any programming task. C does not have a built-in string data type unlike some other programming languages (Java, PHP, Pascal, C#, BASIC, etc.). Instead, C treats a string as a one-dimensional array of data type `char`. By convention, a string in C is an array of characters terminated by the end-of-string sentinel `'\0'` (called NULL character). The NULL character (`\0`) is nothing but a single byte of value zero. (Note that this isn't same as the character `'0'`). While creating a string variable, the programmer must allocate sufficient memory for the number of characters in the string (including the NULL character). For example, to store the word `'home'`, the string variable should be capable of holding at least 5 characters.

The declaration of a string variable is just like the declaration of an array of characters.

```
char word[5];
```

A string constant, syntactically, is a sequence of characters enclosed in double quotes. (e.g. `"Dipesh"`). The compiler automatically places the NUL character at the end of string constant. So we can say that `"D"` and `'D'` are not same in C, because `"D"` has 2 elements, i.e. `'D'` and `'\0'`.

Although C does not have a built-in string data type, it provides the programmer a rich set of library functions for handling strings. The function prototypes for the string handling functions are given in the standard header file `<string.h>`.

Some of the most commonly used string handling functions are:

- `strlen()` is used to get the length of a string.
- `strcmp()` is used to compare two strings alphabetically.
- `strcpy()` is used to copy the contents of one string to another.
- `Strrev( )` is used to reverse the string
- `strcat()` is used to add the contents of one string at the end of another, and so on...

## **PROBLEMS:**

**1. WAP to read two 2-dimensional arrays, pass them to a function to multiply and display the result.**

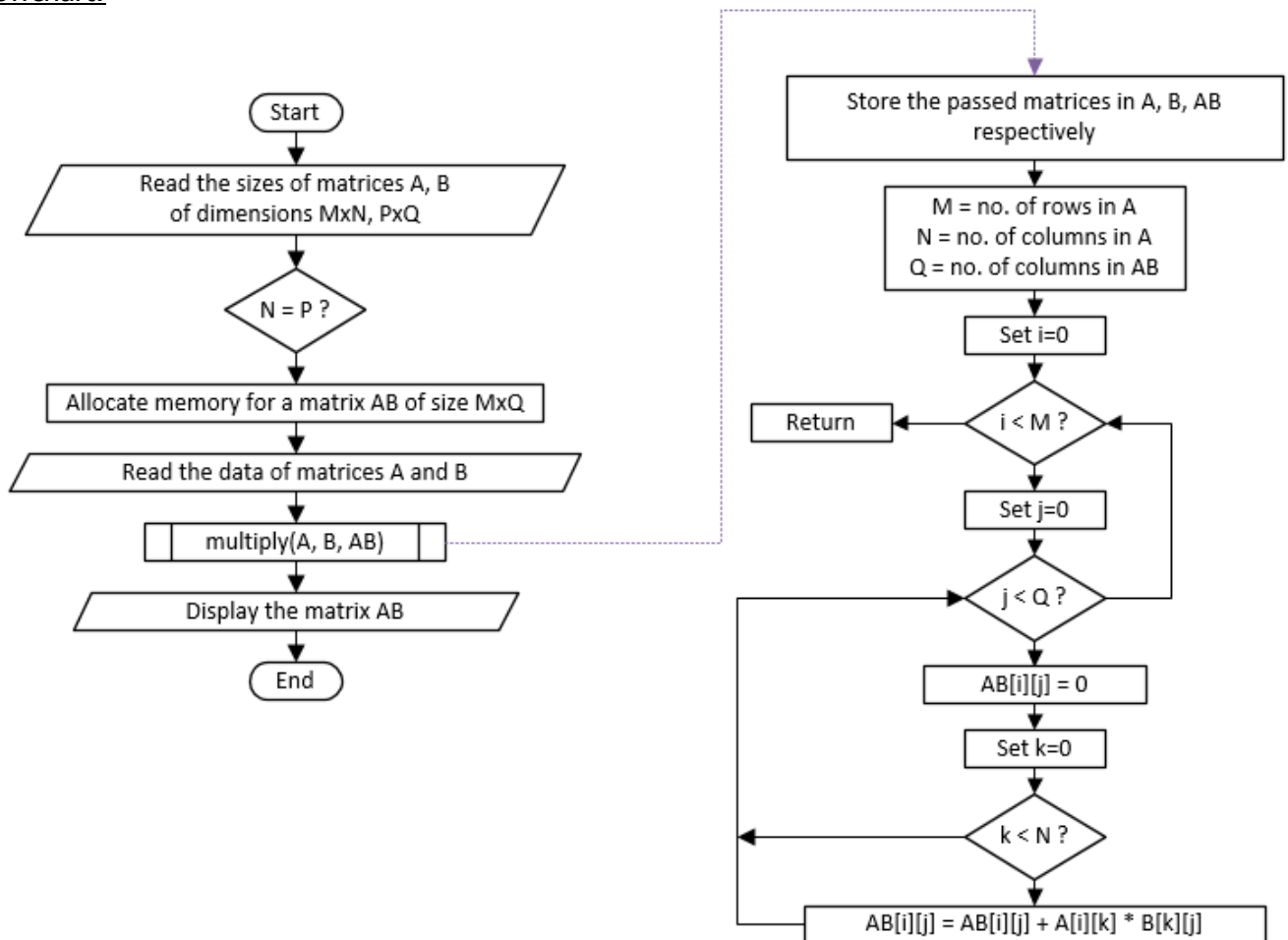
### Algorithm:

1. Start
2. Read the sizes of two matrices A and B of dimensions MxN and PxQ.
3. If  $n \neq p$ , then exit.
4. Allocate memory for a matrix AB of size MxQ.
5. Read the data of matrices A and B.
5. Call multiply(A, B, AB)
6. Display the matrix AB.
7. Stop.

### Algorithm of multiply function:

1. Get matrix A and B from the arguments.
2. Let M = number of rows in A
3. Let N = number of columns in A
4. Let Q = number of columns in B
5. For i = 1 to M
  - 5.1: For j = 1 to Q
    - 5.1.1:  $AB[i][j] = 0$ ;
    - 5.1.2: For k = 1 to N
      - 5.1.2.1:  $AB[i][j] = AB[i][j] + A[i][k] * B[k][j]$

### Flowchart:



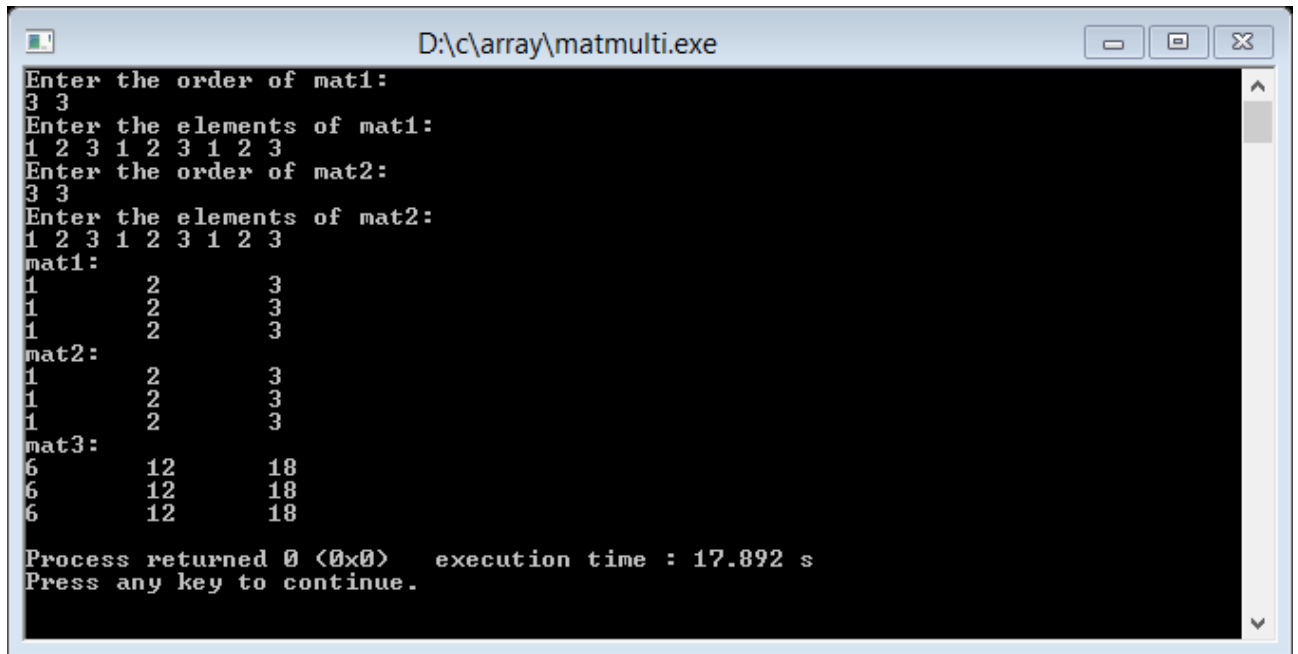
## Source Code:

```
#include<stdio.h>
int main()
{
    int i,j,m,n,p,q,k,l;
    int mat1[10][10], mat2[10][10], multi[10][10];
    int sum=0;
    printf("Enter the order of mat1:\n");
    scanf("%d%d",&m,&n);
    printf("Enter the elements of mat1:\n");
    for(i=0;i<m;i++)
        {for(j=0;j<n;j++)
            {
                scanf("%d",&mat1[i][j]);
            }
        }
    printf("Enter the order of mat2:\n");
    scanf("%d%d",&p,&q);
    if(n!=p)
        printf("The matrices cannot be multiplied\n");
    else
    {
        printf("Enter the elements of mat2:\n");
        for(i=0;i<p;i++)
            {for(j=0;j<q;j++)
                scanf("%d",&mat2[i][j]);}

        for(i=0;i<m;i++)
        {
            for(j=0;j<q;j++)
            {
                for(k=0;k<p;k++)
                {
                    sum+=mat1[i][k]*mat2[k][j];
                }
                multi[i][j]=sum;
                sum=0;
            }
        }
        printf("mat1:\n");
        for(i=0;i<m;i++)
        {
            for(j=0;j<n;j++)
                printf("%d\t",mat1[i][j]);
            printf("\n");
        }

        printf("mat2:\n");
        for(i=0;i<m;i++)
        {
            for(j=0;j<n;j++)
                printf("%d\t",mat2[i][j]);
            printf("\n");
        }
        system("cls");
        printf("mat3:\n");
        for(i=0;i<m;i++)
        {
            for(j=0;j<n;j++)
                printf("%d\t",multi[i][j]);
            printf("\n");
        }
    }
    return 0;}
```

Output:



```
D:\c\array\matmulti.exe
Enter the order of mat1:
3 3
Enter the elements of mat1:
1 2 3 1 2 3 1 2 3
Enter the order of mat2:
3 3
Enter the elements of mat2:
1 2 3 1 2 3 1 2 3
mat1:
1      2      3
1      2      3
1      2      3
mat2:
1      2      3
1      2      3
1      2      3
mat3:
6      12     18
6      12     18
6      12     18
Process returned 0 (0x0)   execution time : 17.892 s
Press any key to continue.
```

Discussion:

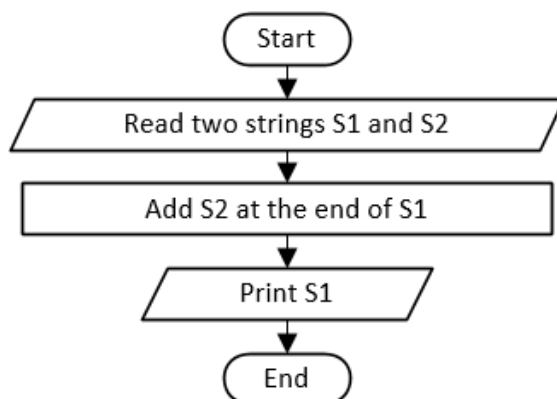
In this program, we asked user for sizes of matrices and then dynamically allocated memory for corresponding two-dimensional matrices and read the data of each matrices. Then we pass these two-dimensional arrays into a function that performs their matrix multiplication and stores the product matrix in another matrix of required size.

## 2. WAP that concatenates two strings with and without using string handling functions.

Algorithm:

- 1: Read two strings s1 and s2
- 2: Add the contents of s2 at the end of s1
- 3: Print s1

Flowchart:



### Source Code:

(Without using string handling functions)

```
#include <stdio.h>
int main(void) {
    char s1[51], s2[51], s3[101];
    int i, j;

    printf("Enter first string: \n"); gets(s1);
    printf("Enter second string: \n"); gets(s2);

    for (i=0; s1[i] != '\0'; i++)
        s3[i] = s1[i];
    for (j=0; s2[j] != '\0'; j++)
        s3[i+j] = s2[j];

    printf("\nThe concatenated string is: \n%s\n\n", s3);
    return 0;
}
```

(Using string handling functions)

```
#include <stdio.h>
#include <string.h>
int main(void) {
    char S1[51], S2[51];
    printf("Enter first string: \n"); gets(S1);
    printf("Enter second string: \n"); gets(S2);

    strcat(S1, S2);

    printf("\nThe concatenated string is: \n%s\n\n", S1);
    return 0;
}
```

### Output:

### Discussion:

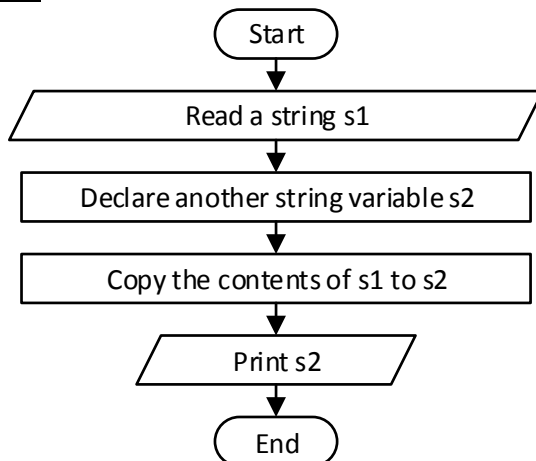
This program concatenates (merges) two strings into one first without using any string handling functions and then by using string handling function. The standard function `strcat()` is used to merge two strings. It is defined in `<string.h>` header file. It takes two string parameters and adds the contents of the 2<sup>nd</sup> string at the end of the 1<sup>st</sup> one.

### 3. WAP to copy one string to another string with and without using string handling functions.

#### Algorithm:

- 1: Read a string s1.
- 2: Declare another string s2.
- 3: Copy all the contents of s1 to s2.
- 4: Print s2.
- 5: End

#### Flowchart:



#### Source Code:

(without using string handling functions)

```
#include <stdio.h>
int main(void) {
    char string[100], copied[100]; int i;
    printf("Enter a string : \n"); gets(string);

    for (i=0; string[i] != '\0'; i++)
        copied[i] = string[i];

    printf("\nYou entered: \n%s\n", copied);
    return 0;
}
```

(by using string handling functions)

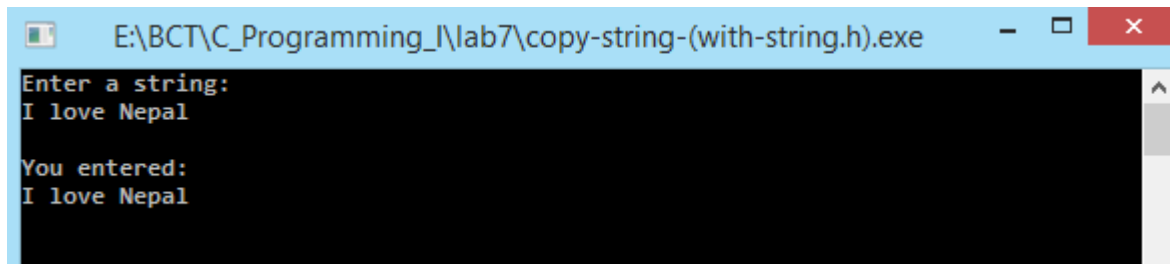
```
#include <stdio.h>
#include <string.h>
int main(void) {
    char string[100]; int i;

    printf("Enter a string: \n");
    gets(string);
    int len = strlen(string);

    char* copied = malloc((len+1) * sizeof(*copied));
    strcpy(copied, string);

    printf("\nYou entered: \n%s\n", copied);
    free(copied); return 0;
}
```

Output:



```
E:\BCT\C_Programming_I\lab7\copy-string-(with-string.h).exe
Enter a string:
I love Nepal

You entered:
I love Nepal
```

Discussion:

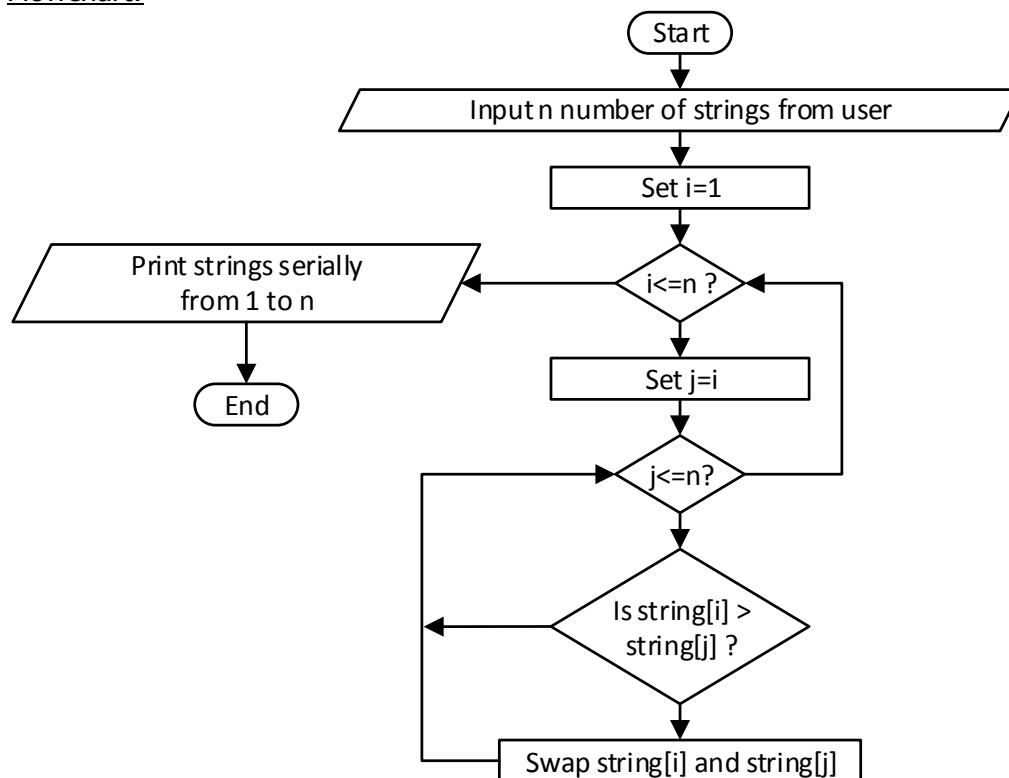
In this program, we read a string from user and copied that string into another string. Without using any string handling functions, it can be done by copying every character of first string into another using loop. But with string handling functions it can be done much more easily. The <string.h> header file provides a function strcpy() that copies the contents of one string to another.

#### 4. WAP that sorts strings in ascending order.

Algorithm:

- 1: Input n number of strings from user.
- 2: For i = 1 to n  
    For j = i to n  
        If ith string is alphabetically greater than jth string, then swap them
- 3: Print strings serially from 1 to n.
- 4: Stop.

Flowchart:



### Source Code:

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

void swapstr(char* str1, char* str2);

int main(void) {
    int n, i, j;
    printf("How many strings? "); scanf("%d", &n); fflush(stdin);
    char** str = malloc(n * sizeof(*str));
    for (i=0; i<n; i++) str[i] = malloc(51 * sizeof(*str[i]));

    printf("\n");
    for (i=0; i<n; i++) {
        printf("Enter string %d : ", i+1); gets(str[i]);
    }

    for (i=0; i<n; i++)
        for (j=i; j<n; j++)
            if (strcmpi(str[j], str[i]) < 0) {
                swapstr(str[i], str[j]);
            }

    printf("\nASCENDING ORDER : ");
    for (i=0; i<n; i++)
        printf("\n %2d. %s ", i+1, str[i]);

    for (i=0; i<n; i++) free(str[i]);
    free(str);
    return 0;
}

void swapstr(char* str1, char* str2) {
    char temp[51];
    strcpy(temp, str1);
    strcpy(str1, str2);
    strcpy(str2, temp);
}
```

### Output:

### Discussion:

In this program, we input an array of strings and sorted them by comparing each string with all the other strings after it by using the `strcmpi()` function defined in the `<string.h>` header file. It compares the alphabetical value of two strings and returns their difference. It makes it really easy to sort strings.

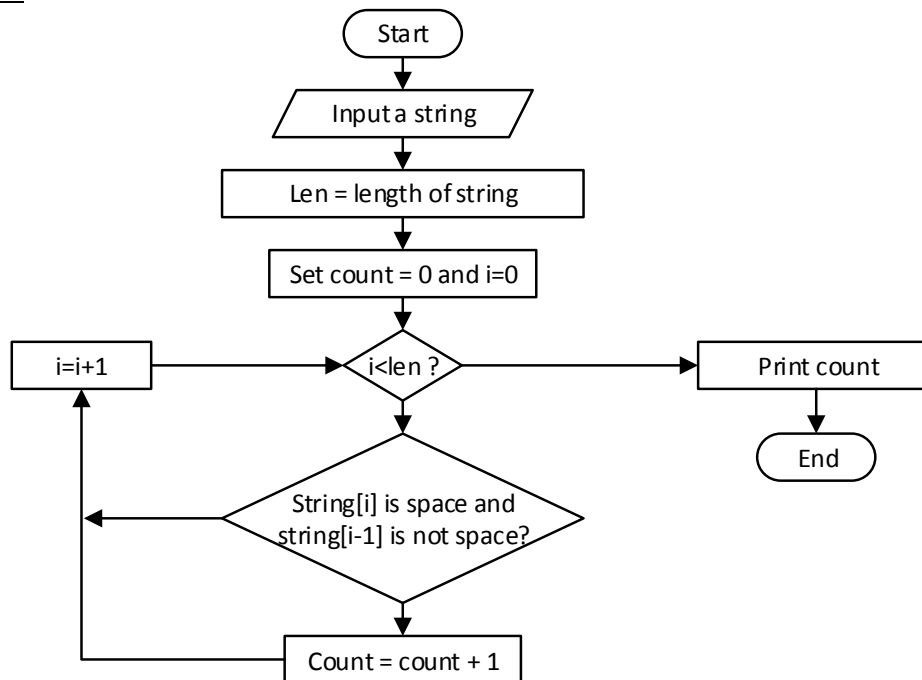


## 5. WAP to find the number of words in a string.

### Algorithm:

```
1: Input a string.
2: Let len = length of string.
3: Set count = i = 0
4: if (i < len) then goto step 5, else goto step
5: if string[i] is space and string[i-1] is not space then count = count + 1
6: i = i + 1
7: goto step 4
8: Print count
9: End
```

### Flowchart:



### Source Code:

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

int main(void) {
    char str[161]; int len, i, count;

    printf("Enter a sentence. (Max 160 characters) \n");
    gets(str);

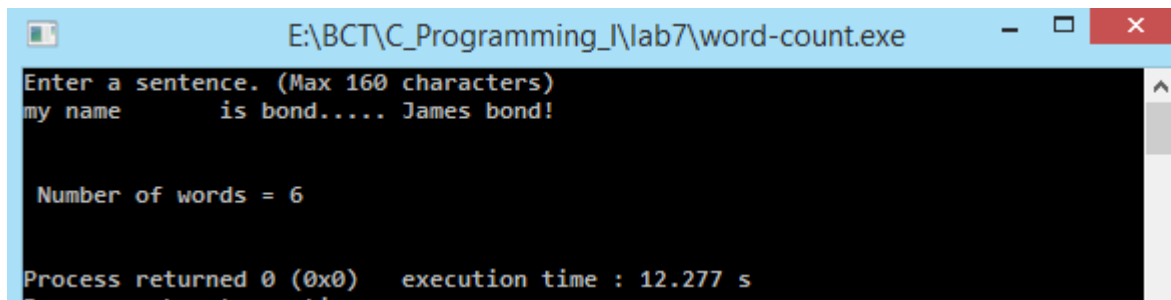
    len = strlen(str);

    count = 0;
    for (i=1; i<=len; i++)
        if((isspace(str[i]) || str[i]=='\0') && !(isspace(str[i-1]))))
            count++;

    printf("\n\n Number of words = %d", count);

    return 0;
}
```

Output:



```
E:\BCT\C_Programming_I\lab7\word-count.exe
Enter a sentence. (Max 160 characters)
my name      is bond..... James bond!

Number of words = 6

Process returned 0 (0x0)   execution time : 12.277 s
```

Discussion:

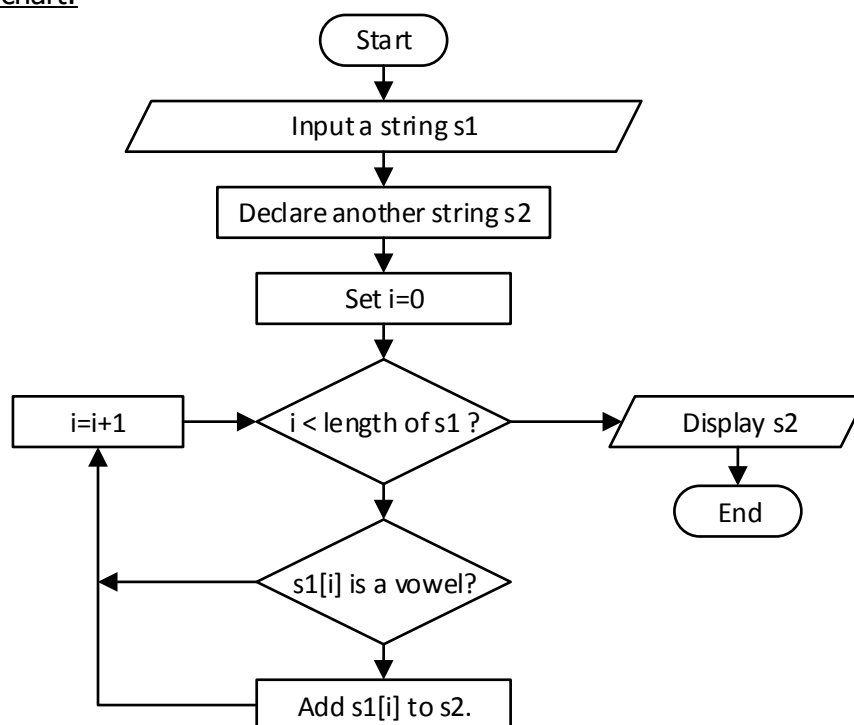
In this program, we used the `strlen()` function to get the length of the inputted string and looped through every character of the string to count the number white spaces. If the character just before the white space is not a white space then we incremented the word count by 1. In this program we used the character function `isspace()` defined in `<ctype.h>` to check whether the character is a white space or not.

## 6. WAP to enter a string and delete all the vowels from the string.

Algorithm:

- 1: Enter a string s1.
- 2: Declare a string s2.
- 3: For every character of s1  
    If it is not vowel, add it to s2
- 4: Print s2.

Flowchart:



### Source Code:

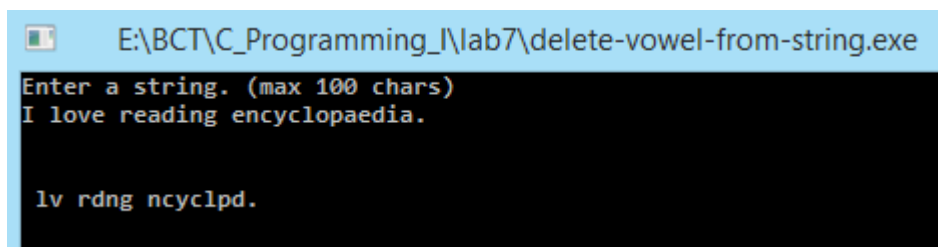
```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

int main(void) {
    char text[101], temp[101], current;
    int i, c;

    printf("Enter a string. (max 100 chars) \n");
    fflush(stdin); gets(text);

    c=0;
    for (i=0; i<=strlen(text); i++)
        switch(toupper(text[i])) {
            case 'A':
            case 'E':
            case 'I':
            case 'O':
            case 'U':
                break;
            default:
                temp[c++] = text[i]; break;
        }
    printf("\n\n%s\n\n", temp);
    return 0;
}
```

### Output:

A screenshot of a Windows command prompt window. The title bar shows the file path "E:\BCT\C\_Programming\_\lab7\delete-vowel-from-string.exe". The prompt displays the program's output: "Enter a string. (max 100 chars)" followed by the input "I love reading encyclopaedia." and the resulting output "lv rdng ncyclpd." where all vowels have been removed from the original string.

```
E:\BCT\C_Programming_\lab7\delete-vowel-from-string.exe
Enter a string. (max 100 chars)
I love reading encyclopaedia.

lv rdng ncyclpd.
```

### Discussion:

In this program, we looped through each character of a string and copied all the non-vowel characters into another string. To make things easier, we used the `strlen()` function to get the length of the inputted string and used the `toupper()` function to convert a character into uppercase to make it easy to check if it's a vowel or not.

### **CONCLUSION:**

From this lab session, we concluded that we can either directly use the defined string operating functions or make our own string handling functions. We also learned to pass the multidimensional arrays using functions.