MIT-WPU

MIT-WORLD PEACE UNIVERSITY

F. Y. B. Tech

Trimester: I/II/III Subject: Programming and Problem Solving

Name: Krishnaraj Thadesar **Division:** 9

Roll No.: 109054 **Batch:** I3

Experiment No.: 5

Name of the Experiment: Write a Menu driven C program to perform all String operations using User Defined functions.

Performed on: 10rd February 2022

Submitted on: 15th February 2022

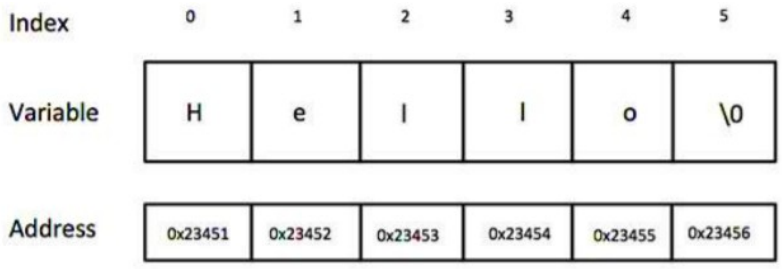
AIM: Write a Menu driven C program to perform all String operations using User Defined functions.

OBJECTIVE:

To learn string Operations in C.

THEORY:

Strings are actually one-dimensional array of characters terminated by a null character '\0'



String Library Functions:

strcpy(s1, s2);

Copies string s2 into string s1.

strcat(s1, s2);

Concatenates string s2 onto the end of string s1.

strlen(s1);

Returns the length of string s1.

strcmp(s1, s2);

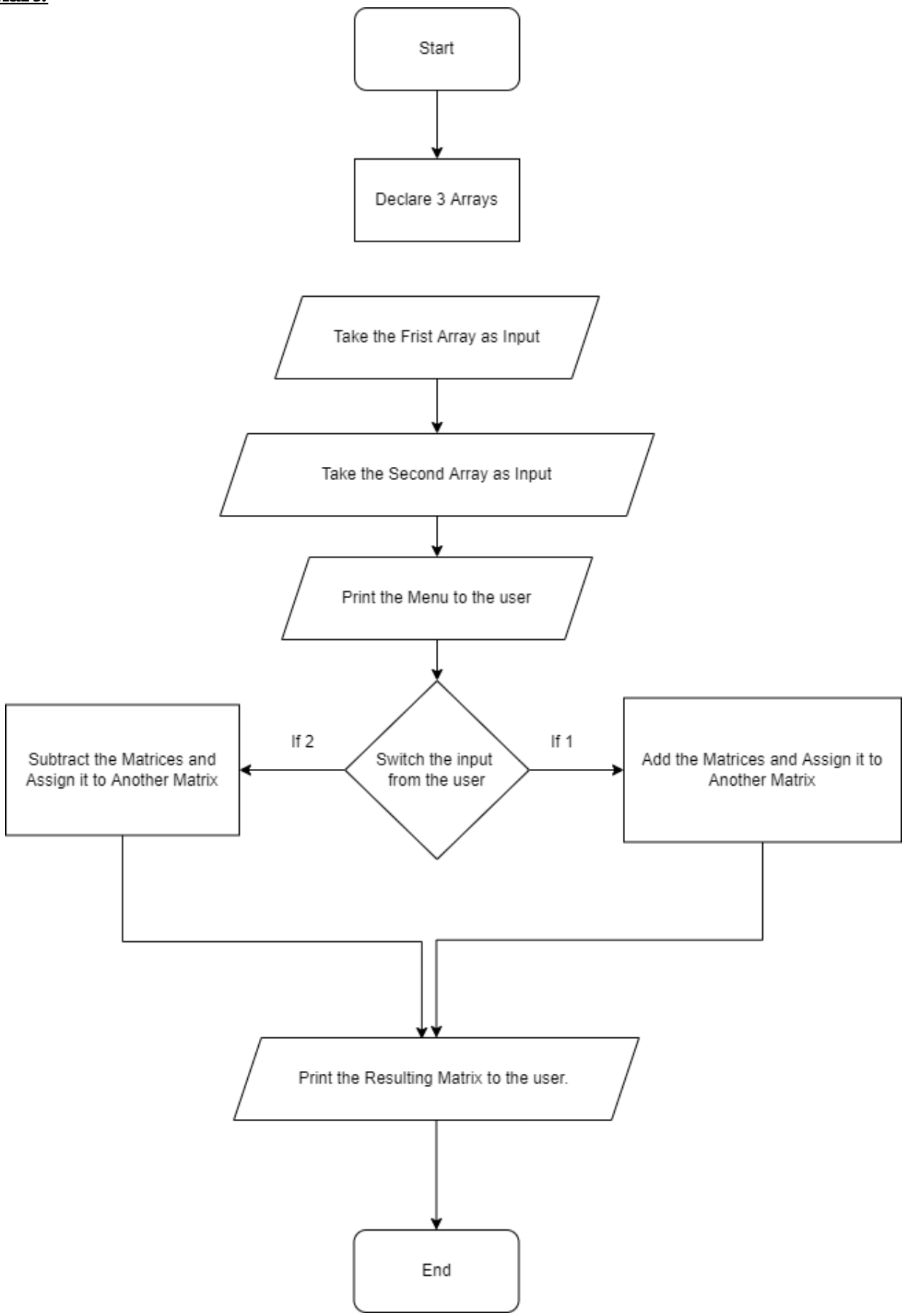
Returns 0 if s1 and s2 are the same; less than 0 if s1s2.

PLATFORM: *64-Bit ArchLinux x86 with gdb/g++ compiler.*

ALGORITHM:

Step 1: Start
Step 2: Declare 3 Matrices of Size [3][3] each and assign them to zero.
Step 3: Input the First Matrix
Step 4: Input the Second Matrix
Step 5: Write another nested For loop
Step 6: Switch the Choice variable inside the for loop
Step 7: If choice is 1, add the values, if choice is 2, subtract the values and assign them to the third matrix C.
Step 6: Output the Third Matrix
Step 7: End

Flowchart:


Start
Declare 3 Arrays
Take the Frist Array as Input
Take the Second Array as Input
Print the Menu to the user
Switch the input from the user
If 2
If 1
Subtract the Matrices and Assign it to Another Matrix
Add the Matrices and Assign it to Another Matrix
Print the Resulting Matrix to the user.
End

CODE:

```
//     strlen() computes string's length strcpy() copies a string to another
//     strcat() concatenates(joins) two strings
//     strcmp() compares two strings
//     strlwr() converts string to lowercase
//     strupr() converts string to uppercase

// Write a menu driven program to perform all string operations (user defined
functions)

#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

// User Defined Functions
int str_length(char *ptr)
{
    /*
        Function: Returns the number of characters in the given character.
        Input: char * pointing to the character array.
        Returns: Integer.
     */
    int count = 0;
    for (int i = 0; ptr[i] != '\0'; i++)
    {
        count++;
    }
    return count;
}
char *str_concat(char *user_string_1, char *user_string_2)
{
    /*
        Function: Returns a character pointer pointing to an array of characters that
is made by concatenating 2 strings.
        Input: char * pointing to the 2 strings.
        Returns: char *.
     */

    // allocating on the heap coz otherwise it would be a local variable
```

```
    // that you cant pass outside the scope of this function as a pointer, as memory
would be invalid.
    char *concat_string = malloc(1000);
    strcpy(concat_string, user_string_1);
    for (int i = 0; i ⩽ str_length(user_string_2); i++)
    {
        concat_string[str_length(user_string_1) + i] = user_string_2[i];
    }
    return concat_string;
}
int str_compare(char *user_string_1, char *user_string_2)
{
    /*
        Compares the C string str1 to the C string str2.
        This function starts comparing the first character of each string.
        If they are equal to each other, it continues with the following pairs until
the characters
        differ or until a terminating null-character is reached.
        Returns:
        <0  the first character that does not match has a lower value in ptr1 than in
ptr2
        0   the contents of both strings are equal
        >0  the first character that does not match has a greater value in ptr1 than
in ptr2
    */
    int result = 0;
    for (int i = 0; user_string_1[i] ≠ '\0' || user_string_2[i] ≠ '\0'; i++)
    {
        if (user_string_1[i] == user_string_2[i])
        {
            if (user_string_1[i + 1] == '\0' && user_string_2[i + 1] ≠ '\0')
            {
                result = 0;
                continue;
            }
            else if (user_string_1[i + 1] == '\0' && user_string_2[i + 1] ≠ '\0')
            {
                result = -1;
            }
            else if (user_string_1[i + 1] == '\0' && user_string_2[i + 1] == '\0')
```

```
            {
                result = 0;
            }
            if (user_string_1[i + 1] != '\0' && user_string_2[i + 1] == '\0')
            {
                result = 1;
            }
        }
        else if (user_string_1[i] < user_string_2[i] || user_string_1[i] >
user_string_2[i])
        {
            result = (user_string_1[i] - user_string_2[i]) / abs(user_string_1[i] -
user_string_2[i]);
            break;
        }
    }
}
char *str_lower(char *user_string)
{
    /*
    Returns a new char * to an array that contains the converted lowercase of the
user_string
    */
    char *lower_string = malloc(1000);
    strcpy(lower_string, user_string);
    for (int i = 0; i < str_length(lower_string); i++)
    {
        if (lower_string[i] >= 'A' && lower_string[i] <= 'Z')
        {
            int AASCI_val = lower_string[i] + 32;
            lower_string[i] = AASCI_val;
        }
    }
    return lower_string;
}
char *str_upper(char *user_string)
{
    /*
    Returns a new char * to an array that contains the converted uppercase of the
user_string
```

```
    */
    char *upper_string = malloc(1000);
    strcpy(upper_string, user_string);
    for (int i = 0; i < str_length(upper_string); i++)
    {
        if (upper_string[i] ⩾ 'A' && upper_string[i] ⩽ 'Z')
        {
            int AASCI_val = upper_string[i] - 32;
            upper_string[i] = AASCI_val;
        }
    }
    return upper_string;
}
char *str_reverse(char *user_string)
{
    /*
    Returns a new char * to an array that contains the reversed user_string
    */

    // allocating on the heap coz otherwise it would be a local variable
    // that you cant pass outside the scope of this function as a pointer, as memory
would be invalid.
    char *rev_string = malloc(1000);
    strcpy(rev_string, user_string);
    for (int i = 0; i < str_length(user_string); i++)
    {
        rev_string[i] = user_string[str_length(user_string) - i - 1];
    }
    rev_string[str_length(user_string)] = '\0';
    return rev_string;
}

int main()
{
    int choice = 0;
    char user_string[500];
    char user_string_1[500], user_string_2[500];

    printf("Enter What operation you want to perform [1, 2, 3, 4, 5]: \n\
        1. Find the length of the String\n\
```

```
        2. Concatenate 2 Strings\n\
        3. Compare 2 Strings\n\
        4. Convert a String to lowercase\n\
        5. Convert a String to Uppercase\n\
        6. Reverse a string\n\
        ");
    scanf("%d", &choice);
    switch (choice)
    {
    case 1:
        printf("Enter the String that you want to find the length of: ");
        scanf("%s", &user_string);
        printf("The Length is: %d", str_length(user_string));
        break;
    case 2:
        printf("Enter the First String: ");
        scanf("%s", &user_string_1);
        printf("Enter the First String: ");
        scanf("%s", &user_string_2);
        printf("The Concatenated is: %s", str_concat(user_string_1, user_string_2));
        break;
    case 3:
        printf("Enter the First String: ");
        scanf("%s", &user_string_1);
        printf("Enter the First String: ");
        scanf("%s", &user_string_2);
        printf("The Comparison of the Strings is: %d", str_compare(user_string_1,
user_string_2));
        break;
    case 4:
        printf("Enter the String that you want to convert to lowercase to: ");
        scanf("%s", &user_string);
        printf("The converted String is: %s", str_lower(user_string));
        break;
    case 5:
        printf("Enter the String that you want to convert to Uppercase to: ");
        scanf("%s", &user_string);
        printf("The converted String is: %s", str_upper(user_string));
        break;
    case 6:
```

```
        printf("Enter the String that you want to reverse: ");
        scanf("%s", &user_string);
        printf("The converted String is: %s", str_reverse(user_string));
        break;
    default:
        printf("Incorrect Choice, Please try again.");
    }
    return 0;
}
```

OUTPUT

```
What operation you want to perform [1, 2, 3, 4, 5]:
        1. Find the length of the String
        2. Concatenate 2 Strings
        3. Compare 2 Strings
        4. Convert a String to lowercase
        5. Convert a String to Uppercase
        6. Reverse a string
        1
Enter the String that you want to find the length of: example
The Length is: 7

2
Enter the First String: example
Enter the First String: String
The Concatenated is: exampleString

3
Enter the First String: First
Enter the First String: Second
The Comparison of the Strings is: -1

3
Enter the First String: First
Enter the First String: First
The Comparison of the Strings is: 0

4
```

```
Enter the String that you want to convert to lowercase to: LOWERcase
The converted String is: lowercase

5
Enter the String that you want to convert to Uppercase to: upperCASE
The converted String is: UPPERCASE

6
Enter the String that you want to reverse: reverse
The converted String is: esrever
```

CONCLUSION:

The working, concept and implementation of single and multi-dimensional arrays was understood in detail and implemented using switch case in a menu driven program.

FAQs:

Q1. What are the different types of arrays and how do we define them?

Ans. There are 2 Types of arrays:

1. *One dimensional arrays*
2. *Two dimensional Arrays*

One Dimensional Arrays:

They are arrays that have a single subscript.

Multi-dimensional Arrays:

An array having more than one dimensions is known as a multi-dimensional array.

```
Syntax: data_type array-name[row-size][col-size]
```

Q2. How are arrays initialized and processed?

Ans. Initialization (static):

```
int matrix[2][2] = {
                {1, 2}
                {2. 3}
                }
```

Initialization (dynamic):

```
int matrix[2];
matrix[0] = 1;
matrix[1] = 2;
```

Q3. How are elements accessed in a 2D array?

Elements in a 2-dimensional array are accessed using row and column indices.

For eg.

```
M = {{1, 2}
     {2, 3}}
M[0][1] = 2;
```

This shows that the element in the 1nd row and 2nd column is 2.