VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE ENGINEERING

AND INFORMATION SYSTEMS

FALL SEMESTER 2024-2025

PMCA501P – DATA STRUCTURES AND ALGORITHM LAB

DIGITAL ASSIGNMENT – 1

SUBMITTED ON: 03 – SEPT - 2024

SUBMITTED BY-

AKASH KUMAR BANIK

PROGRAM: MCA

REGISTER No.: 24MCA0242

PROBLEM STATEMENT – 1:

USING ARRAYS SOLVE THE PROBLEM GIVEN

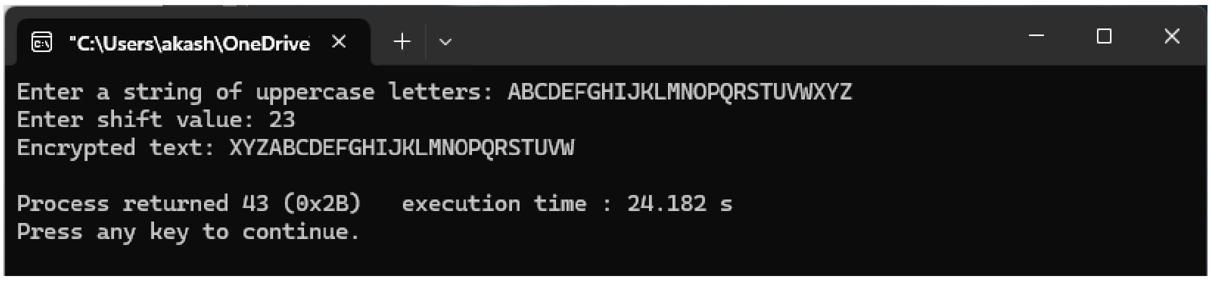
Encryption technique is simply a type of substitution cipher that is each letter of a given text is replaced by a letter with a fixed number of positions down the alphabet. One of such encryption type is called Caesar Cipher technique. For example with a shift of 2, A would be replaced by C, B would become D, and so on. The technique is apparently named after Julius Caesar, who apparently used it to communicate with his officials.

Cipher text is given as (x+n)%26

CODE:

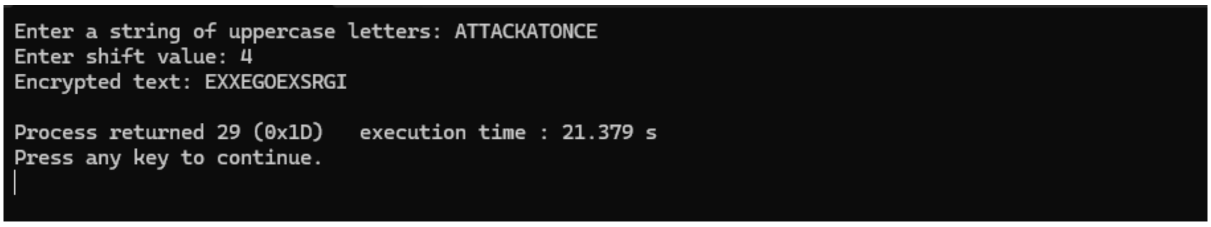
```
#include <stdio.h>

#include <string.h>

void caesarCipher(char text[], int shift) {

    int i;

    int len = strlen(text);

    char cipherText[len + 1];

    for(i = 0; i < len; i++) {

        char ch = text[i];

        if(ch >= 'A' && ch <= 'Z') {

            cipherText[i] = ((ch - 'A' + shift) % 26) + 'A';

        } else {

            cipherText[i] = ch;

        }

    }

    cipherText[len] = '\0';

    printf("Encrypted text: %s\n", cipherText);
}

void main() {

    char text[100];

    int shift;

    printf("Enter a string of uppercase letters: ");

    scanf("%s", &text);

    printf("Enter shift value: ");

    scanf("%d", &shift);

    caesarCipher(text, shift);

}
```

OUTPUT:

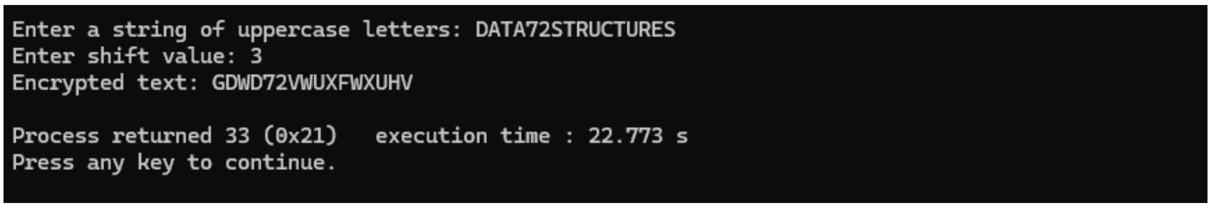


"C:\Users\akash\OneDrive

```
Enter a string of uppercase letters: ABCDEFGHIJKLMNOPQRSTUVWXYZ
Enter shift value: 23
Encrypted text: XYZABCDEFGHIJKLMNOPQRSTUVW

Process returned 43 (0x2B)   execution time : 24.182 s
Press any key to continue.
```



```
Enter a string of uppercase letters: ATTACKATONCE
Enter shift value: 4
Encrypted text: EXXEGOEXSRGI

Process returned 29 (0x1D)   execution time : 21.379 s
Press any key to continue.
```



```
Enter a string of uppercase letters: DATA72STRUCTURES
Enter shift value: 3
Encrypted text: GDWD72VWUXFWXUHV

Process returned 33 (0x21)   execution time : 22.773 s
Press any key to continue.
```

PROBLEM STATEMENT – 2:

VALIDATE STACK SEQUENCES

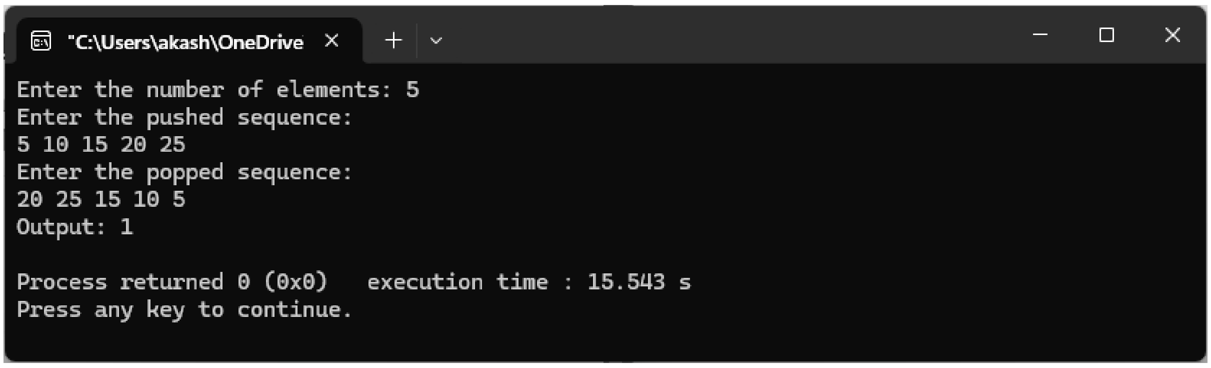
Given two sequences pushed and popped with distinct values, write a program to return true if and only if this could have been the result of a sequence of push and pop operations on an initially empty stack.

CODE:

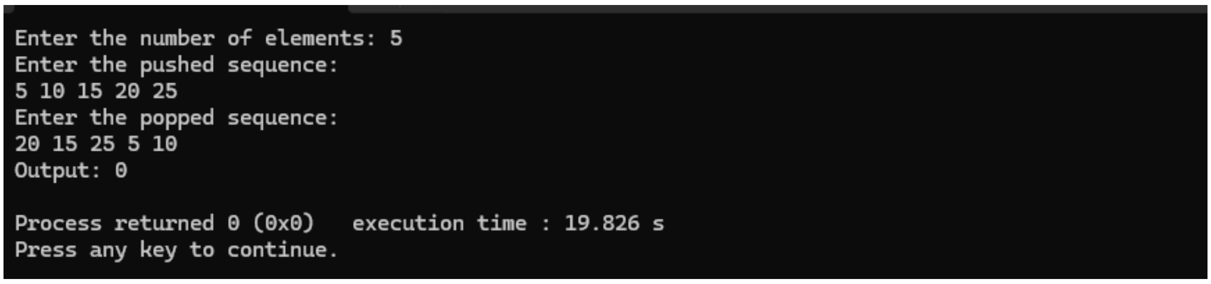
```
#include <stdio.h>

#include <stdbool.h>

#define MAX 100

bool validateStackSequences(int pushed[], int popped[], int n) {

   int stack[MAX], top = -1;

   int j = 0;

   for (int i = 0; i < n; i++) {

      stack[++top] = pushed[i];

      while (top >= 0 && stack[top] == popped[j]) {

         top--;

         j++;

      }

   }

   return top == -1;

}


int main() {

   int n;
```

```
    printf("Enter the number of elements: ");

    scanf("%d", &n);

    int pushed[n], popped[n];

    printf("Enter the pushed sequence:\n");

    for (int i = 0; i < n; i++) {

        scanf("%d", &pushed[i]);

    }

    printf("Enter the popped sequence:\n");

    for (int i = 0; i < n; i++) {

        scanf("%d", &popped[i]);

    }

    if (validateStackSequences(pushed, popped, n)) {

        printf("Output: 1\n");

    } else {

        printf("Output: 0\n");

    }

    return 0;

}
```

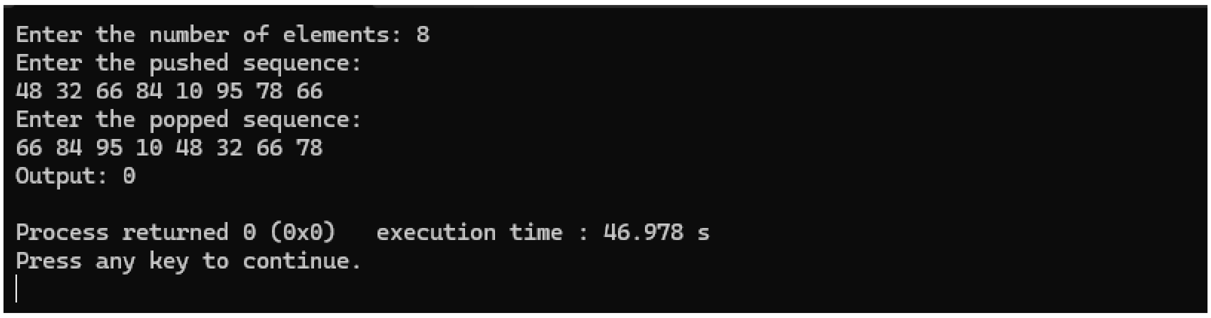
OUTPUT:



```
Enter the number of elements: 5
Enter the pushed sequence:
5 10 15 20 25
Enter the popped sequence:
20 25 15 10 5
Output: 1

Process returned 0 (0x0)   execution time : 15.543 s
Press any key to continue.
```



```
Enter the number of elements: 5
Enter the pushed sequence:
5 10 15 20 25
Enter the popped sequence:
20 15 25 5 10
Output: 0

Process returned 0 (0x0)   execution time : 19.826 s
Press any key to continue.
```



```
Enter the number of elements: 8
Enter the pushed sequence:
48 32 66 84 10 95 78 66
Enter the popped sequence:
66 84 95 10 48 32 66 78
Output: 0

Process returned 0 (0x0)   execution time : 46.978 s
Press any key to continue.
```

PROBLEM STATEMENT – 3:

CHECKING PARENTHESIS IN SCRIPTING CODE

```
<script>

document.getElementById("demo").innerHTML = "Hello JavaScript!";

</script>

document.getElementById("demo").style.fontSize = "25px";

document.getElementById("demo").style.color = "red";

document.getElementById("demo").style.backgroundColor = "yellow";

document.getElementById("image").src = "picture.gif";

<style>

body {background-color: powderblue;}

h1 {color: red;}

p {color: blue;}

</style>
```

Write a C program to read the above java script or part of the above code as a stream of input text and check the parenthesis like curly braces, regular parenthesis and angle brackets are rightly placed. Also count the total number of such pairs, if each category is less than five then print it as insufficient and between 5 and 10 then print it as moderate, and above 10 means then print it as sufficient.

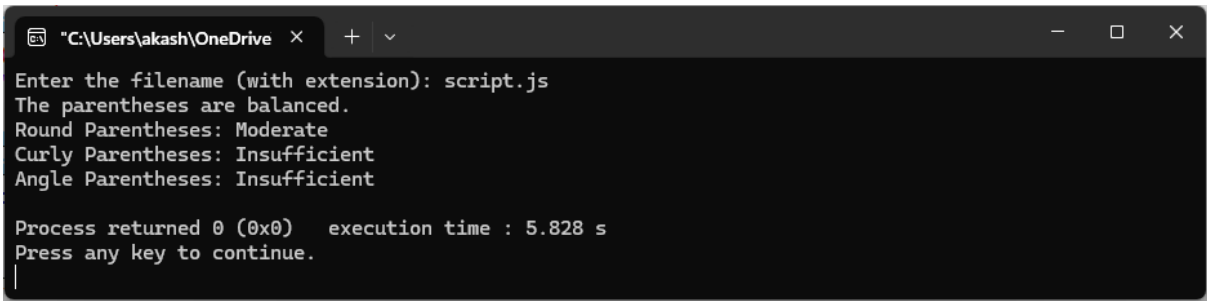
CODE:

```
#include <stdio.h>

#include <stdlib.h>

#define MAX 1000

int isOpening(char ch) {

   return (ch == '(' || ch == '{' || ch == '<');

}

int isClosing(char ch) {

   return (ch == ')' || ch == '}' || ch == '>');

}

int isMatchingPair(char opening, char closing) {

   return ((opening == '(' && closing == ')') ||

           (opening == '{' && closing == '}') ||

           (opening == '<' && closing == '>'));

}

int checkBalancedParentheses(char code[], int *roundCount, int *curlyCount, int *angleCount)
{

   char stack[MAX];

   int top = -1;

   int i;
```

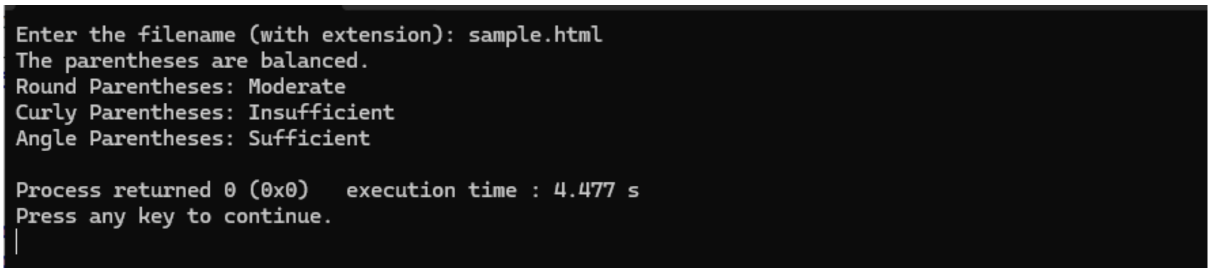
```
    for (i = 0; code[i] != '\0'; i++) {
        char ch = code[i];
        if (isOpening(ch)) {
            stack[++top] = ch;
        } else if (isClosing(ch)) {
            if (top == -1 || !isMatchingPair(stack[top--], ch)) {
                return 0;
            }
            if (ch == ')') (*roundCount)++;
            else if (ch == '}') (*curlyCount)++;
            else if (ch == '>') (*angleCount)++;
        }
    }
    return (top == -1);
}
void categorizeCount(int count, const char* type) {
    if (count < 5) {
        printf("%s Parentheses: Insufficient\n", type);
    } else if (count <= 10) {
        printf("%s Parentheses: Moderate\n", type);
    } else {
        printf("%s Parentheses: Sufficient\n", type);
    }
}
```

```
int main() {
    char code[MAX];
    int roundCount = 0, curlyCount = 0, angleCount = 0;
    FILE *file;
    char filename[100];
    printf("Enter the filename (with extension): ");
    scanf("%s", filename);
    file = fopen(filename, "r");
    if (file == NULL) {
        printf("Could not open file %s\n", filename);
        return 1;
    }
    char ch;
    int i = 0;
    while ((ch = fgetc(file)) != EOF && i < MAX - 1) {
        code[i++] = ch;
    }
    code[i] = '\0';
    fclose(file);
    if (checkBalancedParentheses(code, &roundCount, &curlyCount, &angleCount)) {
        printf("The parentheses are balanced.\n");
    } else {
        printf("The parentheses are not balanced.\n");
    }
    categorizeCount(roundCount, "Round");

    categorizeCount(curlyCount, "Curly");

    categorizeCount(angleCount, "Angle");

    return 0;

}
```

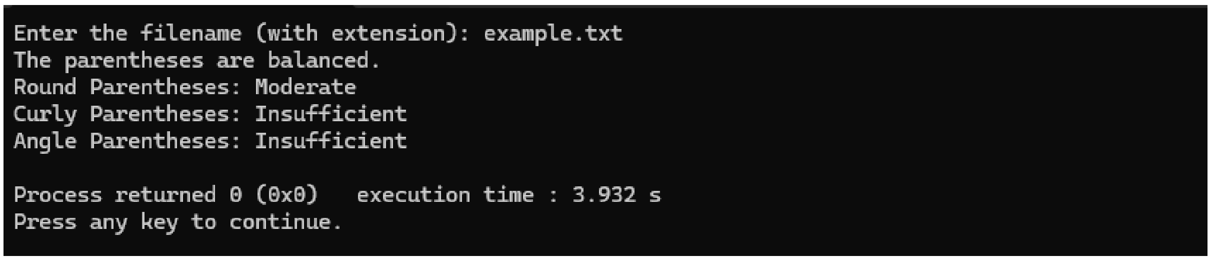
OUTPUT:



```
Enter the filename (with extension): script.js
The parentheses are balanced.
Round Parentheses: Moderate
Curly Parentheses: Insufficient
Angle Parentheses: Insufficient

Process returned 0 (0x0)   execution time : 5.828 s
Press any key to continue.
```



```
Enter the filename (with extension): sample.html
The parentheses are balanced.
Round Parentheses: Moderate
Curly Parentheses: Insufficient
Angle Parentheses: Sufficient

Process returned 0 (0x0)   execution time : 4.477 s
Press any key to continue.
```



```
Enter the filename (with extension): example.txt
The parentheses are balanced.
Round Parentheses: Moderate
Curly Parentheses: Insufficient
Angle Parentheses: Insufficient

Process returned 0 (0x0)   execution time : 3.932 s
Press any key to continue.
```