K. J. Somaiya College of Engineering, Mumbai

(A Constituent College of Somaiya Vidyavihar University)

Department of Computer Engineering

Batch: **B4** **Roll No.:** **16010122828**

Experiment / assignment / tutorial No. 4

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: Implementation of Stack Applications

Objective: To implement applications of stack

Expected Outcome of Experiment:

CO	Outcome
1	Explain the different data structures used in problem solving

Books/ Journals/ Websites referred:

1. *Fundamentals Of Data Structures In C* – Ellis Horowitz, Satraj Sahni, Susan Anderson-Fred
2. *An Introduction to data structures with applications* – Jean Paul Tremblay, Paul G. Sorenson
3. *Data Structures A Pseudo Approach with C* – Richard F. Gilberg & Behrouz A. Forouzan
4. https://www.cprogramming.com/tutorial/computersciencetheory/stack.html
5. https://www.geeksforgeeks.org/stack-data-structure-introduction-program/
6. https://www.thecrazyprogrammer.com/2013/12/c-program-for-arrayrepresentation-of-stack-push-pop-display.html

Assigned Stack application: check if the set of parenthesis are balanced.

Algorithm:

1. Initialize an empty stack.
2. Iterate through each character in the string.
3. For each open parenthesis (i.e. '(', '{', or '['), push it onto the stack.
4. For each closed parenthesis (i.e. ')', '}', or ']'), check if the stack is empty. If it is, return false (i.e. unbalanced).
5. If the stack is not empty, pop the top element and check if it matches the closed parenthesis (i.e. '(' matches ')', '{' matches '}', '[' matches ']'). If it does not match, return false (i.e. unbalanced).
6. If the loop completes without returning false, check if the stack is empty. If it is, return true (i.e. balanced). If it is not, return false (i.e. unbalanced).

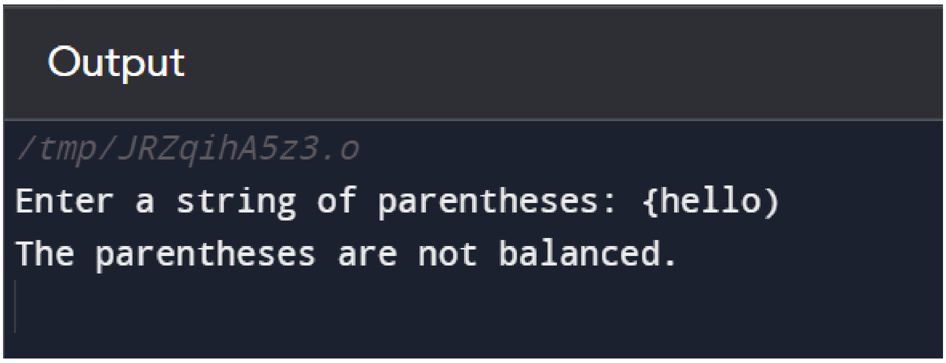
The algorithm makes use of the stack property that last-in-first-out (LIFO) which means that when a closed parenthesis is encountered, it should match the last open parenthesis that was pushed on the stack. If they match, it is balanced otherwise it's unbalanced.

Sourcecode:

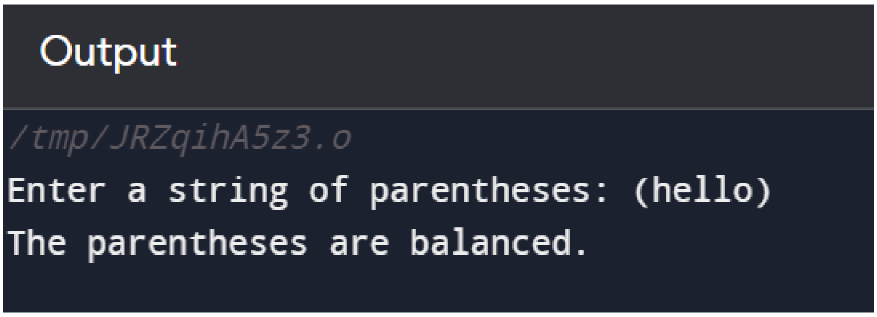
```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX_SIZE 100
typedef struct {
   char data[MAX_SIZE];
   int top;
} Stack;
void push(Stack* stack, char item) {
   if (stack->top == MAX_SIZE - 1) {
      printf("Error: Stack overflow\n");
      return;
   }
   stack->data[++stack->top] = item;
}
char pop(Stack* stack) {
   if (stack->top == -1) {
      printf("Error: Stack underflow\n");
      exit(1);
```

```
    }
    return stack->data[stack->top--];
}
bool isBalanced(char* str) {
    Stack stack;
    stack.top = -1;
    for (int i = 0; str[i] != '\0'; i++) {
        if (str[i] == '(' || str[i] == '{' || str[i] == '[') {
            push(&stack, str[i]);
        } else if (str[i] == ')' || str[i] == '}' || str[i] == ']') {
            if (stack.top == -1) {
                return false;
            }
            char item = pop(&stack);
            if (str[i] == ')' && item != '(') {
                return false;
            } else if (str[i] == '}' && item != '{') {
                return false;
            } else if (str[i] == ']' && item != '[') {
                return false;
            }
        }
    }
    return stack.top == -1;
}
int main() {
    char str[MAX_SIZE];
    printf("Enter a string of parentheses: ");
    scanf("%s", str);
    if (isBalanced(str)) {
        printf("The parentheses are balanced.\n");
    } else {
        printf("The parentheses are not balanced.\n");
    }
    return 0;
}
```

Output Screenshots:



```
/tmp/JRZqihA5z3.o
Enter a string of parentheses: {hello)
The parentheses are not balanced.
```



```
/tmp/JRZqihA5z3.o
Enter a string of parentheses: (hello)
The parentheses are balanced.
```

Conclusion:

The aim of the experiment is verified.