Batch: C1 **Roll No.: 16010122257**

Experiment / assignment / tutorial No. 04

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-array-representation-of-stack-push-pop-display.html*
7. *Our Classroom slides.*

Assigned Stack application:Parenthesis matching using stack.

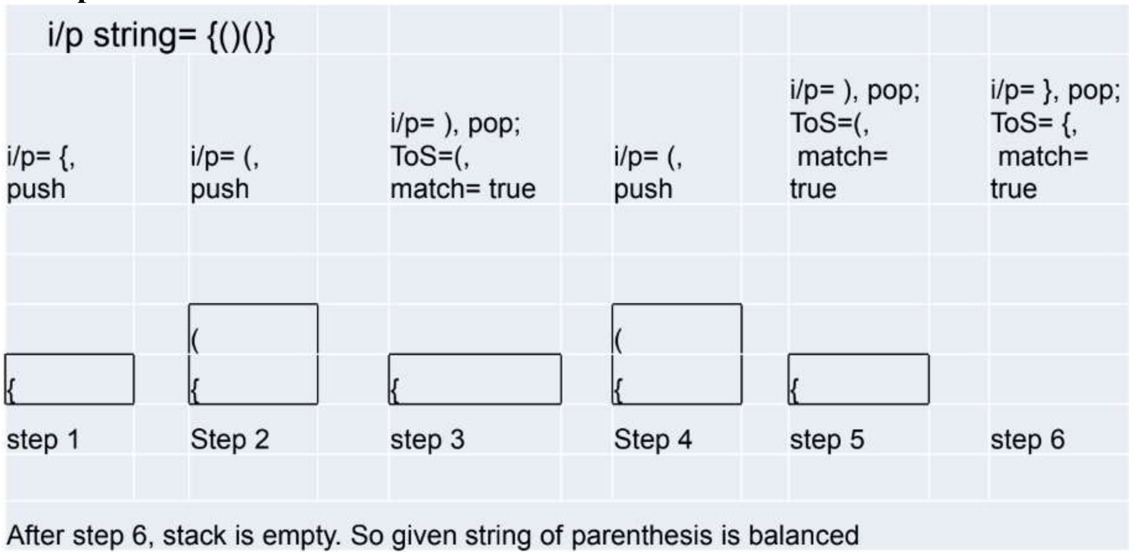
Algorithm:

```
Algorithm Boolean ParenMatch(X,n):

Input: An array X of n tokens, each of which is either a grouping symbol, a

variable, an arithmetic operator, or a number

Output: true if and only if all the grouping symbols in X match

Let S be an empty stack

for i=0 to n-1 do

if X[i] is an opening grouping symbol then

S.push(X[i])

else if X[i] is a closing grouping symbol then

if S.isEmpty() then


return false {nothing to match with}

if S.pop() does not match the type of X[i] then


return false {wrong type}


if S.isEmpty() then

return true {every symbol matched}
```

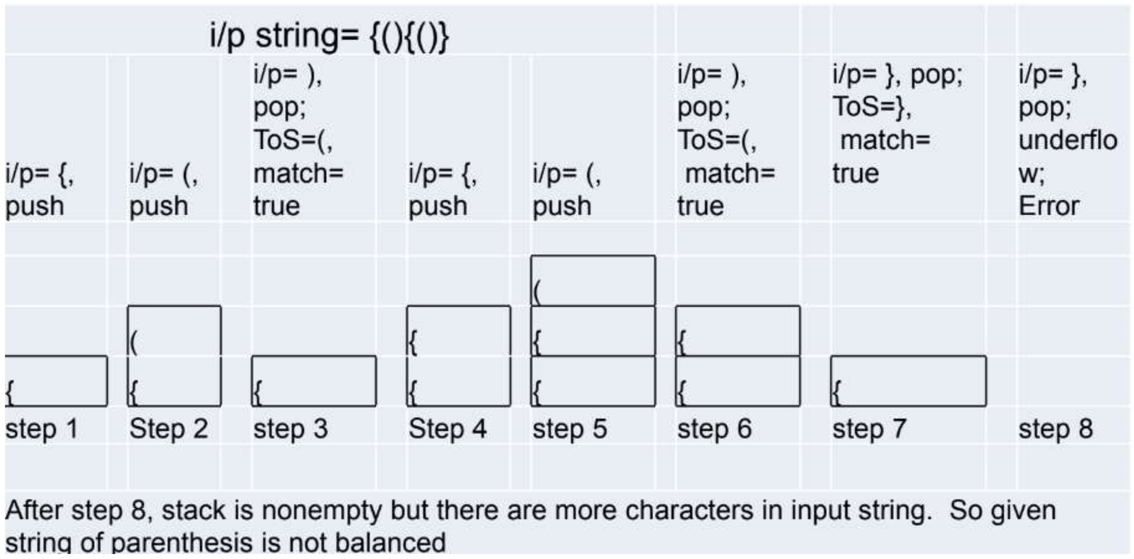
else

return false {some symbols were never matched}

Example:


i/p string= {()()}
i/p= {, push
i/p= (, push
i/p=), pop; ToS=(, match= true
i/p= (, push
i/p=), pop; ToS=(, match= true
i/p= }, pop; ToS= {, match= true
step 1
Step 2
step 3
Step 4
step 5
step 6

After step 6, stack is empty. So given string of parenthesis is balanced


i/p string= {(){()}
i/p= {, push
i/p= (, push
i/p=), pop; ToS=(, match= true
i/p= {, push
i/p= (, push
i/p=), pop; ToS=(, match= true
i/p= }, pop; ToS=}, match= true
i/p= }, pop; underflow; Error
step 1
Step 2
step 3
Step 4
step 5
step 6
step 7
step 8

After step 8, stack is nonempty but there are more characters in input string. So given string of parenthesis is not balanced

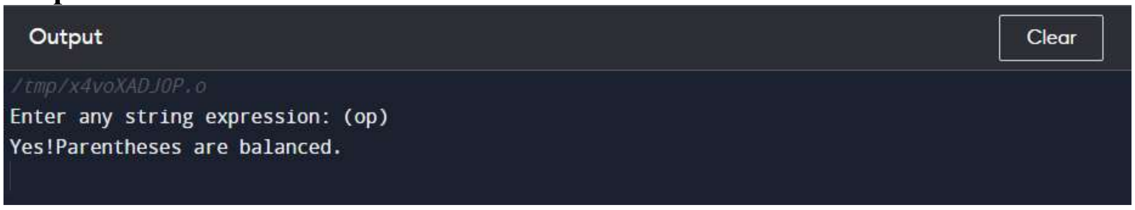
Sourcecode:

```
#include <stdio.h>

#include <string.h>



struct Stack {

    char arr[150];
    int top;
};

void initialize(struct Stack *s) {
    (*s).top = -1;
}

void push(struct Stack *s, char c) {
    if ((*s).top < 99) {
        (*s).top++;
        (*s).arr[(*s).top] = c;
    }
}

char pop(struct Stack *s) {
    char result;
    if ((*s).top >= 0) {
        result = (*s).arr[(*s).top];
        (*s).top--;
    }
    return result;
}

char peek(struct Stack *s) {
    char result;
    if ((*s).top >= 0) {
        result = (*s).arr[(*s).top];
    }
    return result;
}

int isEmpty(struct Stack *s) {
    int empty;
    if ((*s).top == -1) {
        empty = 1;
    } else {
        empty = 0;
    }
    return empty;
}

int parenMatch(char X[]) {
    struct Stack S;
    initialize(&S);

    int m = strlen(X);
    int i = 0;
      while(i < m) {
        if (X[i] == '(') {
            push(&S, X[i]);
```

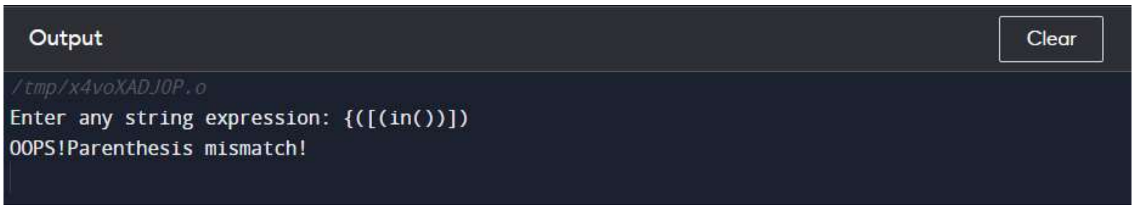
```
} else if (X[i] == '[') {
    push(&S, X[i]);
} else if (X[i] == '{') {
    push(&S, X[i]);
} else if (X[i] == ')') {
    if (isEmpty(&S)) {
        return 0;
    }

    char popped = pop(&S);

    if (popped != '(') {
        return 0;
    }
} else if (X[i] == ']') {
    if (isEmpty(&S)) {
        return 0;
    }

    char popped = pop(&S);

    if (popped != '[') {
        return 0;
    }
} else if (X[i] == '}') {
```

```
            if (isEmpty(&S)) {
                return 0;
            }

            char popped = pop(&S);

            if (popped != '{') {
                return 0;
            }
        }
        i++;
    }

    if (isEmpty(&S)) {
        return 1;
    } else {
        return 0;
    }
}

int main() {
    char expression[150];
    printf("Enter any string expression: ");
    scanf("%s", expression);
    if (parenMatch(expression)) {

        printf("Yes!Parentheses are balanced.\n");

    } else {

        printf("OOPS!Parenthesis mismatch!\n");

    }


    return 0;

}
```

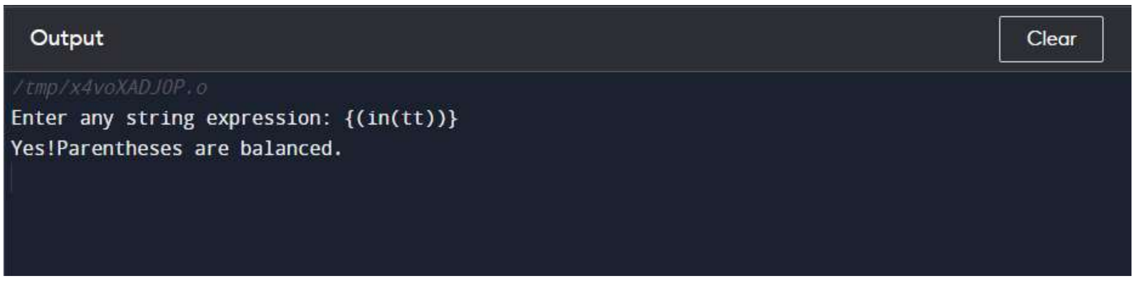
Output Screenshots:



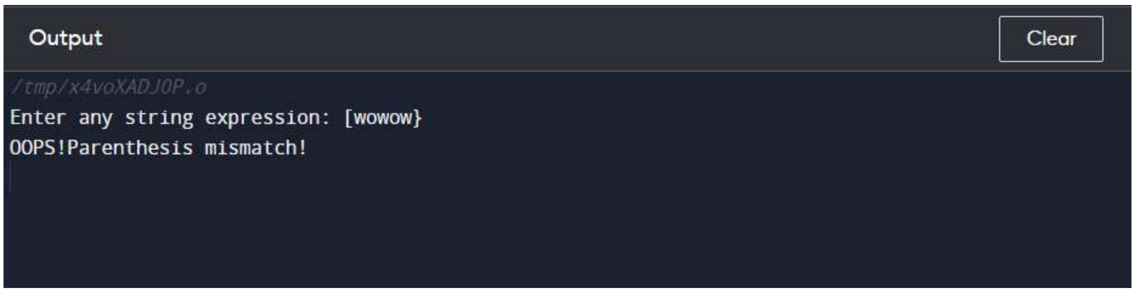
```
/tmp/x4voXADJ0P.o
Enter any string expression: (op)
Yes!Parentheses are balanced.
```



```
/tmp/x4voXADJ0P.o
Enter any string expression: {([(in())])
OOPS!Parenthesis mismatch!
```



```
/tmp/x4voXADJ0P.o
Enter any string expression: {(in(tt))}
Yes!Parentheses are balanced.
```



```
/tmp/x4voXADJ0P.o
Enter any string expression: [wowow}
OOPS!Parenthesis mismatch!
```

Conclusion: Thus applied stack in a given problem statement/application.It is a common problem in computer science and programming.By applying data structures in real life and in computers we’ve transformed them into practical use.