Date 21-NOV-2020

Assignment 09 | 21118 | E-1 (SE-1)

Problem Statement:

In any language program mostly syntax errors occurs due to unbalancing delimiter such as (), {}, []. Write C++ program using stack to check wheather given expression is well parenthesized or not.

Objectives:

a) To understand the concept of OOP paradigm.
b) To understand usage of ~~linked list~~ stack data-structure to support FIFO operations.

Outcome:

a) To implement stack using linked list / arrays.
b) Use stack to various real world software engineering applications.

Hardware Requirement:

Manufacturer & Model: Acer 8011ff-3.
Processor: Intel core i-5 8th gen (8265U @ 1.8GHz)
Installed Memory: 8GB DDR4 RAM, 512 GB SSD.
Architecture: 64-bit.

Software Requirement:

Operating System: Ubuntu 20.04 LTS on Oracle Virtual Machine (Base memory allocated is 4096 MB & 3 processors)
Cpp Version used: C++14
Compiler for C++: g++ (version 10.1.0)
Code editor: Sublime Text (Build 2011)

Theory:

Stacks:

1) Stack is a linear data structure which follows a particular

order in which the operations are performed.

2) The order is LIFO (Last in first out) or FIFO (first in first out)

3) The stack can be implemented using array or linked list.

4) There are many real world applications of stack.
ex. balanced parantheses, web pages surfing.

Balanced Parantheses Expression:

If we consider a expression containing brackets, it is said to be balanced if

1) it contains no unmatched brackets.
2) The subset of brackets enclosed within the confines of matched pair of brackets is also a matched pair of brackets.

Pseudo Code:

```
Algorithm checkBalancedParenthes(string exp){
    n ← exp.length
    Stack stk
    for i←0 to n {
        if (exp[i] is opening) {
            stk.push(exp[i])
        }
        else if (exp[i] is closing) {
            if (stk.empty)
                return false
            if (stk.top() & exp[i] are not matching)
                return false
            stk.pop()
        }
    }
```

```
    if (stk.empty):
        return true.
    else.
        return false;
}
```

ADT of classes:

1> class stack {
```
private:
    char arr[];
    int top;
public:
    stack() {
        top ← -1
    }
    void Push(char x) {  // Push method of stack.
    { // code
    void Pop() {  // Pop method of stack.
    } // code
    bool isEmpty() {  // check if stack is empty or not
    { // code
    }
    char Top() {  // Top method of stack.
    { // code
    }
}
```

2> class Expression {
```
private:
    string exp;
    bool isOpening(char ch) {
    // code.
    }
    bool isClosing(char ch) {
    { // code
```

```
bool areMatching(char open, char close){
    //code
}
public:
    Expression(){          //constructor
    }
    void getInput(){
    // method to get i/p from user.
    }
    bool isValid(){        //checks for valid parentheses.
        //code
    }
}
```

Analysis of Algorithms:

For array implementation (used in program):

1. Push operation: This operation takes const time.
2. Pop operation: This operation also takes const. time
3. Top operation: This operation also takes const time.

→ If the stack is constructed using dynamic arrays then while doing push operation, if stack is already full it will take $\theta(n)$ time. Still amortized time complexity will be $O(1)$.

→ In case of linked-lists all operations will be of $O(1)$.

⇒ The space is needed to store the expression itself & also for the storing elements in stack. In worst case it will be $O(n)$.

Testcases:

Sr No.	Testcase Description	Expected Output	Actual Output
1)	opening bracket in excess eg [(())4}	Expression is not balanced.	Not balanced.

3>	closing brackets are in excess. eg ()]	Expression is not balanced.	Not balanced.
2>	Balanced expression. eg [](){}[{}]	Expression is balanced.	Balanced.
4>	All pair are present but expression is not balanced. eg [{]}	Expression is not balanced.	Not balanced.

Applications

The real world applications of stacks are as follows:

1> expression evaluation (infix, postfix, prefix)
2> To check matching paranthesis in text editors.
3> expression conversion (eg infix to postfix)
4> Memory management
5> Recursive programs use inbuilt stack in computer memory while calling the function again & again.

Conclusion:

Stack is a very useful data structure is case of real world scenarios. This assignment includes array implementation of stack due to which one can know the details of data structure in much more depth.