Return to Assignments

DSA - Assignment 1
Matching Parentheses
using Stack Data Structure

Introduction

In this subsection, we explore two related applications of stacks, both of which involve testing for pairs of matching delimiters. In our first application, we consider arithmetic expressions that may contain various pairs of grouping symbols, such as

- Parentheses: "(" and ")"
- Braces: "{" and "}"
- Brackets: "[" and "]"

Each opening symbol must match its corresponding closing symbol. For example, a left bracket, "[," must match a corresponding right bracket, "]," as in the following expression

$[(5+x)-(y+z)]$.

The following examples further illustrate this concept:

- Correct: ( )(( )){([( )])}
- Correct: ((( )(( )){([( )])}))
- Incorrect:)(( )){([( )])}
- Incorrect: ({[ ])}
- Incorrect: (

An Algorithm for Matching Delimiters

An important task when processing arithmetic expressions is to make sure their delimiting symbols match up correctly. We can use a stack to perform this task with a single left-to-right scan of the original string. Each time we encounter an opening symbol, we push that symbol onto the stack, and each time we encounter a closing symbol, we pop a symbol from the stack (assuming it is not empty) and check that these two symbols form a valid pair. If we reach the end of the expression and the stack is empty, then the original expression was properly matched. Otherwise, there must be an opening delimiter on the stack without a matching symbol. If the length of the original expression is n, the algorithm will make at most n calls to push and n calls to pop.

The task of the assignment:

Write Java program to accept a string s, which is a expression containing delimiter pairs ( ), { }, and [ ], then check whether the string s is a valid expression or not (suppose that except the delimiters, the string s contains valid numbers and operators).