Lab 13: Stacks

One of the tasks that compilers must frequently perform is checking whether some pair of expression delimiters are properly paired, even if they are embedded multiple pairs deep.

Consider the following C++ statement:

a = (func (b) – (c + d)) / 2;

The compiler has to be able to determine which pairs of opening and closing parentheses go together and whether the whole expression is correctly parenthesized. A number of possible errors can occur, because of incomplete pairs of parentheses, or because improperly placed parentheses. For example, the statement below is missing a closing parenthesis.

a = (func (b) – (c + d) / 2;

A **stack** is extremely helpful in implementing a solution to this type of problem because of its **LIFO** behavior. A closing parenthesis needs to be matched with the most recently encountered opening parenthesis; this is handled by pushing opening parentheses onto a stack as they are encountered. When a closing parenthesis is encountered, it should be possible to pop the matching opening parenthesis off the stack. If it is determined that every closing parenthesis had a matching opening parenthesis, then the expression is valid.

The start project contains a **main** function that asks the user to enter an **expression** (a C++ statement) to check its **validity**. You need to implement the function **checkDelimiters** by passing the expression as a string and using a stack to check its validity.

Possible output:

```
This program checks for properly matched delimiters.

Enter delimited expression:
a = (func (b) - (c + d)) / 2;
Valid

Check another one? (y/n) y

Enter delimited expression:
a = (func (b) - (c + d) / 2;
Invalid

Check another one? (y/n) n

Press any key to continue . . .
```