Data Structures and Algorithms Lab

Lab 06 **Marks 10**

Instructions

- Work in this lab individually. Follow the best coding practices and include comments to explain the logic where necessary.
- You can use your books, notes, handouts, etc. but you are not allowed to borrow anything from your peer student.
- Do not use any **AI** tool for help; doing so will be considered cheating and may result in lab cancellation and possible disciplinary action.
- Test your program thoroughly with various inputs to ensure proper functionality and error handling.
- Show your work to the instructor before leaving the lab to get some or full credit.

Delimiter Validation in Expressions Using Stacks

One of the tasks that **compilers** and **interpreters** must frequently perform is deciding whether pairs of **expression delimiters** are properly matched, even if they are nested multiple pairs deep. Consider the following C++ expression:

$$a = \big(f[b] - (c + d)\big)\ /\ 2;$$

The compiler must determine which pairs of **opening** and **closing delimiters** (parentheses, square braces, etc.) correspond and whether the entire expression is **correctly delimited**. Several types of errors can occur, such as **unpaired delimiters** or **improperly placed delimiters**. For instance, the expression below is missing a closing parenthesis:

$$a = (f[b] - (c + d)\ /\ 2;$$

Another example of an invalid expression has the correct number of delimiters but **incorrectly balanced** pairs. The first closing parenthesis does not match the most recent opening delimiter (a brace):

$$a = (f[b) - (c + d])\ /\ 2;$$

A **stack** is extremely useful for solving this problem due to its **LIFO (Last-in, First-out) behavior**. A closing delimiter must correctly match the most recently encountered opening delimiter. This can be managed by **pushing** opening delimiters onto a stack as they appear. When a closing delimiter is encountered, it should be possible to **pop** the matching opening delimiter off the stack. If every closing delimiter has a matching opening delimiter, the expression is **valid**.

```
bool delimitersOk( const string &expression );
```

This function should **return true** if all the **parentheses and braces** in the string are correctly paired, and **false** otherwise.

Your program should:

1. **Read input from a file** named input.txt.
2. **Display** "valid" if the expression is correct and "invalid" if the expression is incorrect.

Input Format

The **input.txt** file will contain:

- The first line with the **total number of expressions** in the file.
- Each subsequent line will contain **one expression** with **no spaces or blank characters**.

Sample (input.txt):

6
$a = \big(f[b] - (c + d)\big)\ /\ 2;$
$a = (f[b] - (c + d)\ /\ 2;$
$a = (f[b) - (c + d])\ /\ 2;$
$3 * (a + b)$
$f[3 * (a + b)]$
$a = f[b + 3$

Output

```
Valid
Invalid
Invalid
Valid
Valid
Invalid
```