Data Structures and Algorithms Lab

Instructions

Work on this lab individually. *Write **main** function first and keep on testing the functionality of each function once created.*

Program the following tasks in your **C++** compiler and then compile and execute them.

Email your solution (**.cpp**) file only to the following respective recipient till **Friday, April 16, 2021**.

DO NOT compress/zip your solution.

The email must be sent from your **official PUCIT email id**, otherwise it will **NOT BE ACCEPTED** and will be marked **ZERO**.

The subject of the email should be the exact name of the lab i.e. **Lab 07**. **2 MARKS** will be **DEDUCTED**, otherwise.

Degree	Recipient Email	Subject of Email
BSIT Morning	dsaubt01@gmail.com	Lab 07
BSIT Afternoon	dsaubt02@gmail.com	

What you have to do

Generally, the **stack** is very useful in situations in which data have to be stored and then retrieved in reverse order because of its **LIFO**—Last In, First Out—behavior. One application of the stack is in matching delimiters in a program. This is an important example because delimiter matching is a part of any compiler: No program is considered correct if the delimiters are mismatched.

In C++ programs, we have the following delimiters:

Delimiters	Symbol
Parentheses	()
Square brackets	[]
Curly brackets	{}
Comments	/* */

Consider the following C++ expression.

```
a = (f[b] – (c+d)) / 2;
a = /* initializing a */ (f[b] – (c+d)) / 2;
```

The compiler has to be able to determine which pairs of opening and closing parentheses, square braces, etc. go together and whether the whole expression is correctly delimited. A number of possible errors can occur because of unpaired delimiters or because of improperly placed delimiters. For instance, the expression below lacks a closing parenthesis.

```
if(a < 5) { p = 7;
```

The following expression is also invalid. There are the correct numbers of parenthesis and braces, but they are not correctly balanced. The first closing parenthesis does not match the most recent opening delimiter, a brace.

```
while( m < n[8) + o]
```

Implement the following function

```
bool delimitersMatching( const string &expression )
```

Returns ***true*** if all the parentheses, braces etc. in the string are legally paired. Otherwise, returns ***false***.

Your task is to write a program that inputs from a text file **expressions.txt** and displays **valid** or **invalid** for each correct and incorrect expression respectively.

The input file **(expressions.txt)** will contain input and is in exactly the following format: First line of the input file will contain the total number of expressions exists in the file and then each new expression exists on a new line.

sample.txt

```
3
a = /* initializing a */ (f[b] — (c+d)) / 2;
while( m < n[8) + o]
3 * (a+b)
```

Output

```
Valid
Invalid
Valid
```

☺ ☺ ☺ **BEST OF LUCK** ☺ ☺ ☺