Programming Assignment #6
Stack – Part I

Due Date: _____

Purposes:
Work with a stack data structure to implement a conversion of infix-notation to postfix-notation arithmetic expressions. Work with file I/O operations.

Description:
Write a program that evaluates arithmetic expressions in infix notation. Your program should read arithmetic expressions from an input file (`expr.dat`). Your program converts an expression of each line to postfix notation and evaluates the postfix expression. Your program should output the original infix expression and the equivalent postfix expression onto the screen.
Optional: Write the postfix expression to an output file (`postexpr.dat`) as well as to the screen.

Note that you need to implement your own stack class where the items of the stack are stored in the array – either static or dynamic. Your stack class needs to support the basic operations that were discussed in the class such as *pop, push, top, isEmpty*, etc.

You will need a stack object to store operators in arithmetic expression.

Note that the expression may not be fully parenthesized, in such a case, the usual C++ precedence rules are used to determine the order of evaluation.

Create `expr.dat` file with the following sample expressions as the first 4 expressions:
4 * 20 – 100 + 40 / 5
(10 – 1) % 2 * 10 + (5 + 9) * 9 / 3
25 * (15 – 110 / 10) / 3 + 7 % 5
(3 + (10 – 5) * (4 – 2) – 5 / 2) / (4 + 2 * (6 – 4))

Process a line (one expression) at a time. That is, you need to check for EOF(end-of-file) or a newline character. Valid characters in the expressions include digits, operators(+,-,*,/,%), and parentheses. Any blank spaces should be ignored (not treated as invalid character).
You may read numbers as integer. (Optional enhancement is to read them as double.)

Add more sample expressions to the file to show how your program handles invalid expressions such as unbalanced parentheses, invalid characters in an expression.

Your program should make sure that the file is open properly before attempting to read data from the file. Make your program robust by adding graceful error-handling – for instance, miss matching parentheses.

Other useful hints:

Use the following member functions of input stream class from the standard library.

peek() returns the next character of the input stream without actually reading it.
ignore() reads and discards the next character from the input stream.

Use the following member functions of the character functions from the standard library.

isdigit(ch) returns true if the character given (ch) is digit (0 – 9)
isspace(ch) returns true if the character given (ch) is white space character

Make your program robust by adding graceful error-handling – for instance, miss matching parentheses.

Your main program should be organized using your own functions. Utilize reference parameters for your functions.
All the codes including your header file must be well commented.