CS 1410 Introduction to Computer Science – CS2
Section 1: MWF 9:30 a.m. – 10:20 a.m.
Instructor: Xiaojun Qi
Programming Assignment #11

Given: Tuesday, April 16, 2013
Due: 11:59 p.m. Monday, April 22, 2013
Total Points: 20 points

Postfix notation is a notation for writing arithmetic expressions in which the operands appear before their operators. There are no precedence rules to learn, and parantheses are never needed. Because of this simplicity, some popular hand-held calculators use postfix notation to avoid the complications of the multiple parentheses required in nontrivial infix expressions. In this assignment, you are to write a computer program that converts any **valid** infix expression to its equivalent postfix expression. For simplicity, the operands can be any reasonable number represented by a single captalized alphabetic letter ("A" to "Z") and the operators are limited to +, -, *, /, (, and).

The input is a series of arithmetic expressions entered interactively from the keyboard in infix notation. The user is prompted to enter an expression made up of any of six required operators and numbers represented by any single captalized alphabetic letter. The end of the expression is marked by the expression terminator character, ']'. Here are several sample inputs and their expected output. Note: <space> indicates the program outputs the space key.

The input is: A*(B-(C+D))]
Its postfix notation is:
A<space>B<space>C<space>D<space>+<space>-<space>*<space>

The input is: (A+B)*C/D]
Its postfix notation is:
A<space>B<space>+<space>C<space>*<space>D<space>/<space>

The input is: (A+B)*(C-D)]
Its postfix notation is:
A<space>B<space>+<space>C<space>D<space>-<space>*<space>

The input is: A+B*(C+(D-E))]
Its postfix notation is:
A<space>B<space>C<space>D<space>E<space>-<space>+<space>*<space> +<space>

The input is: A+B*C-D]
Its postfix notation is:
A<space>B<space>C<space>*<space>+<space>D<space>-<space>

Create a Dynamic Stack class to represent a dynamic stack of a data type of your choice [Refer to Chapter18Example2.zip]. Test your conversion with the above infix expressions and ***several others.***

A few general examples are listed here for your reference:

	Infix notation	Postfix notation
1.	a+(b+c)	abc++
2.	(a+b)+c	ab+c+
3.	a-b*c	abc*-
4.	(a/b)*(c/d)	ab/cd/*
5.	a/(b+c*d-e)	abcd*+e-/
6.	a-b*c+d/e	abc*-de/+

The basic idea for converting a valid infix expression to its equivalent postfix expression is:

Push [onto the stack.

Move from left to right in the infix expression to read each input item until "]" is read

For each input item

If it is a number (i.e., operand), output this number (e.g., capitalized 'A' to 'Z')

Else if it is an operator:

a) If the current input item is '(', push '(' onto the stack.

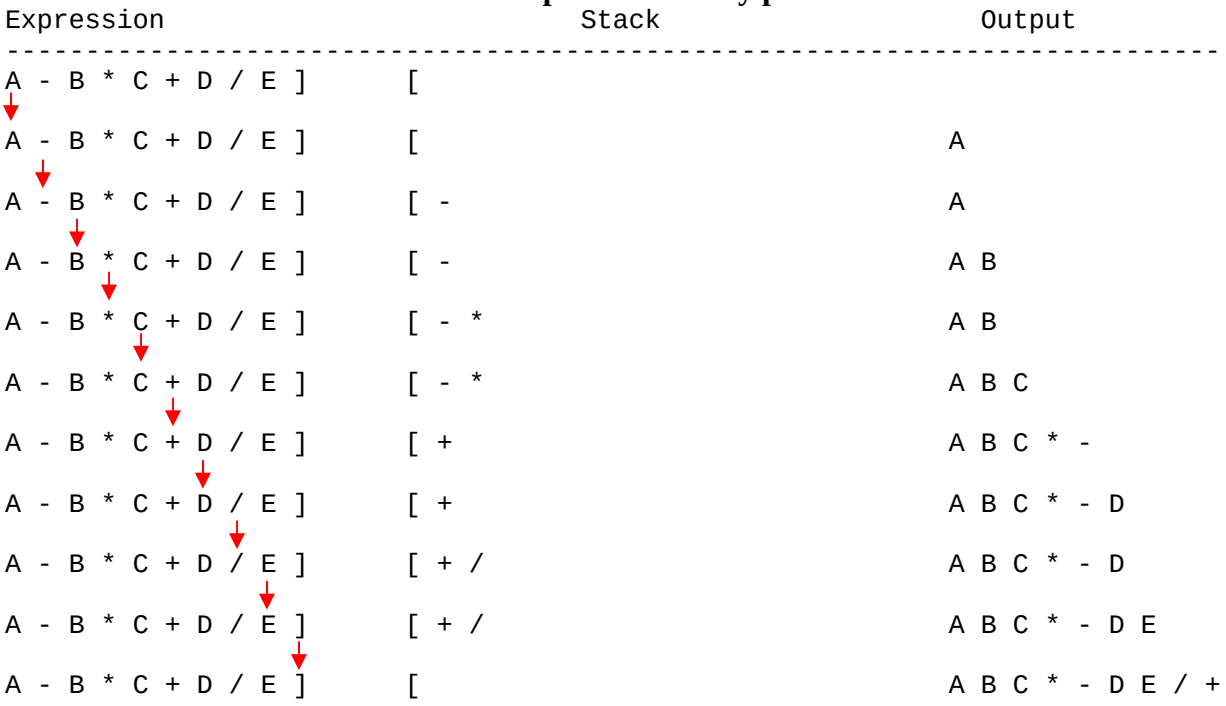
b) If the current input item is ')', pop elements off the stack and output them until reaching a corresponding '(' symbol. This operation ensures that the elements after '(' is printed in the Last-In-First-Out manner and the '(' symbol is popped off the stack.

c) If the top operator on the stack has higher precedence than the current input item, c.1) pop any operators on the stack which are of higher precedence and output the popped operators; c.2) push the input operator.

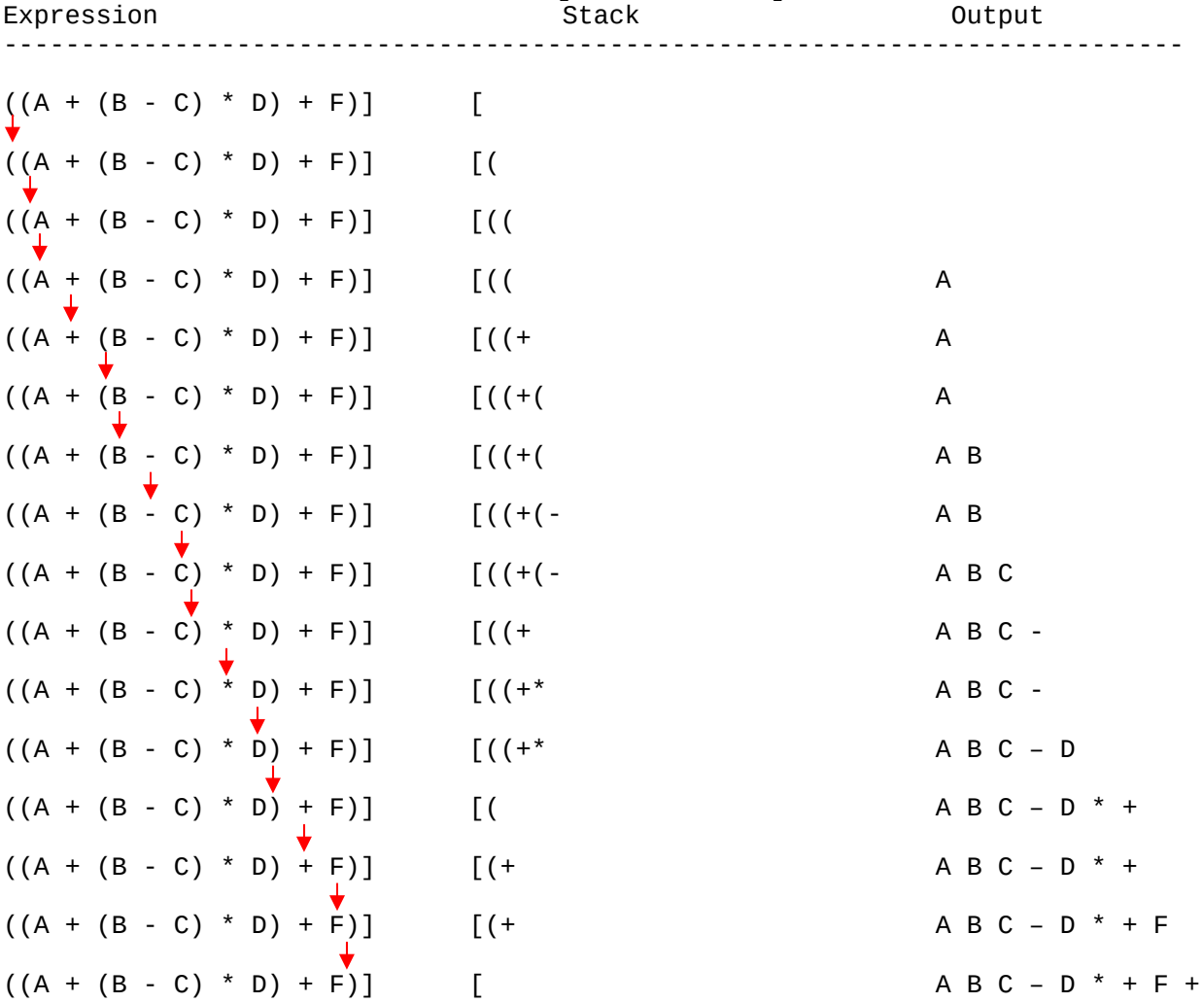
d) If the top operator on the stack has lower precedence than the current input item, push the input operator.

e) If the current input item is ']', pop any operators on the stack until the stack is empty and output the popped operators.

Here is the illustration of the first example without any parathesis:



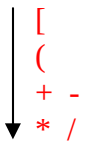
Here is the illustration of the second example with a lot of paratheses:



Expression	Stack	Output
((A + (B - C) * D) + F)]	[	
((A + (B - C) * D) + F)]	[(	
((A + (B - C) * D) + F)]	[((	
((A + (B - C) * D) + F)]	[((	A
((A + (B - C) * D) + F)]	[((+	A
((A + (B - C) * D) + F)]	[((+(	A
((A + (B - C) * D) + F)]	[((+(	A B
((A + (B - C) * D) + F)]	[((+(-	A B
((A + (B - C) * D) + F)]	[((+(-	A B C
((A + (B - C) * D) + F)]	[((+	A B C -
((A + (B - C) * D) + F)]	[((+*	A B C -
((A + (B - C) * D) + F)]	[((+*	A B C – D
((A + (B - C) * D) + F)]	[(	A B C – D * +
((A + (B - C) * D) + F)]	[(+	A B C – D * +
((A + (B - C) * D) + F)]	[(+	A B C – D * + F
((A + (B - C) * D) + F)]	[	A B C – D * + F +

You must write a driver program to test several infix to postfix conversions and demonstrate you have correctly implemented the above algorithm. The grader will create his own driver code to test all the functionalities.

Note:
For your convenience, I list the operators from the lowest precedence to the highest precedence.



‘)’ is not used for the precedence comparison. See condition b) in the algorithm.
‘]’ is not used for the precedence comparison. See condition e) in the algorithm.
‘[’ is a special symbol used to be matched with the expression terminator character, ']'.