CS 340
Assignment 1

1 Programming Part [70pts+10pts(Bonus)]

Given an arithmetic expression in infix or postfix notation, the goal of this assignment is to :

1. Write a program that converts the expression to another notation using expression trees.
2. Evaluate the expression.

The main idea consists of the following steps :

1. Convert the given (input) expression into a postfix notation (if it is not the case).
2. Construct the expression tree from the postfix representation.
3. Evaluate the expression tree or print it into another notation.

The expression contains only the arithmetic operators +, *, - and /, integers and (,) if the expression is in infix form. * and / have more priority than + and -. You should leave at least 1 space between operators, operands and symbols (the minus sign("-") for negative numbers should however be attached to the number, -23 for instance).

1.1 Infix to Postfix Conversion

Implement the algorithm that converts an expression from infix to postfix. Below are some examples for conversion from infix to postfix.

```
-12 + 13 --> -12 13 +

13 + 24 * 35 / 46 --> 13 24 35 * 46 / +

( 4 + 8 ) * ( 6 - 5 ) / ( 3 - 2 ) * ( 2 + 2 ) --> 4 8 + 6 5 - * 3 2 - /
2 2 + *

( ( ( ( 1 * ( 2 + 3 ) ) - 3 ) + 4 ) * 5 ) --> 1 2 3 + * 3 - 4 + 5 *
```

1.2 Constructing an expression tree from a postfix notation

Implement the algorithm that converts a postfix expression into an expression tree.

1.3 Printing an arithmetic expression from the expression tree

Using the following tree traversal algorithms, write a program that prints an arithmetic expression in a given notation (prefix, infix or postfix) from an expression tree.

1.3.1 Inorder traversal

An overly parenthesized infix expression can be produced by recursively producing parenthesized left expression, then printing out the operator at the root, and finally recursively producing parenthesized right expression. This general strategy (left,node,right) is known as an **inorder** traversal.

1.3.2 Postorder traversal

The algorithm consists in recursively print out the left subtree, the right subtree, and then the operator.

1.3.3 Preorder traversal

This method consists in printing out the operator first and then recursively print out the left and the right subtrees.

1.4 Evaluating an arithmetic expression from the expression tree

Write a program that evaluates an arithmetic expression represented by an expression tree.

1.5 Marking scheme of the programming part: total = 70pts + 10pts (Bonus)

1. Readability (program style) : 10pts
   - Program easy to read,
   - well commented,
   - good structured (layout, indentation, whitespace, ...) and designed(following the top-down approach).
2. Compiling and execution process : 10pts
   - program compiles w/o errors and warnings
   - robustness : execution w/o run time errors
3. Correctness : 50pts
   - code produces correct results (output) :
     (a) infix → postfix notation 8pts
     (b) infix → prefix notation 8pts
     (c) postfix → infix notation 8pts
     (d) postfix → prefix notation 8pts
     (e) evaluate the expression tree 6pts
     (f) Handling numbers with more than 1 digit 6pts
     (g) Detecting the type of expression in input(prefix, infix or postfix) 6pts
4. Bonus : 10pts
   - Handling expressions in prefix form in input 5pts
   - Detecting the type of errors (missing operator or operand, unknown symbol ... etc) 3pts
   - Other features that increase functionality and/or presentation 2pts

2 Hand in

Using UR Courses, submit all source files in one single zip file named: **assign1username.zip**. Your source files should include the following:

1. README file listing your name and ID #, and the compiling and execution commands of your program on Titan. Any requirement regarding the input format should also be listed.
2. A screenshot showing your command line for execution and the execution results for 5 different examples.
3. headers (.h)
4. implementations (.cpp)
5. the Makefile :

- should be named "**makefile**". In the makefile, the generated executable should be named : "**assign1username**"

You can give any name to your source files. The marker will run "**make**" to compile your program and "**assign1username**" to execute it.