CS 202: Fundamental Structures of Computer Sciences II

Assignment 2
Due: 23:59, April 3 (Thursday), 2014

In this assignment, you will get familiar with an important programming concept, i.e., algebraic expression tree and different expression notations such as *prefix*, *postfix*, and *infix*. You will be using the concepts of binary tree data structure to convert and evaluate expressions.

Algebraic Expressions

An algebraic expression can be represented using three different notations:

- **Infix:** The most conventional way to write an expression is called infix notation. The arithmetic operators appear in between two operands. *e.g.,* $2 + 5 * 3 + 1$
- **Prefix:** In prefix notation, as the name suggests, operators come before the operands. *e.g.,* $++2*531$
- **Postfix:** In postfix notation, different from infix and prefix notations, operators come after the operands. *e.g.,* $253*+1+$

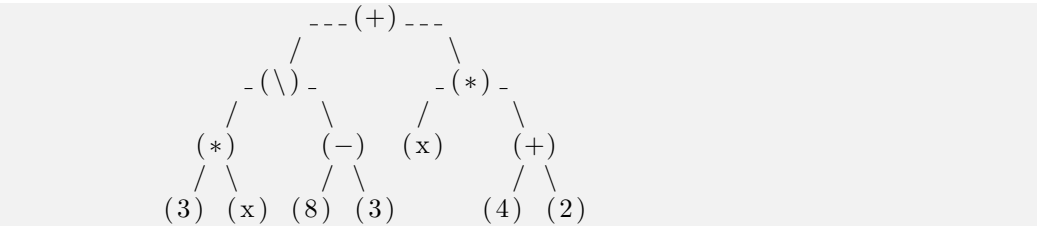
Note that you learned these notations and algebraic expressions in your CS 201 course. If you do not feel comfortable with them, please revise your CS 201 course materials.

Algebraic Expression Trees

One use of the binary trees in computer science is to represent algebraic expressions. In an algebraic expression tree, each leaf node contains an operand and a non-leaf node, including the root node, contains an operator that is applied to the results of its left and right sub-trees. For example, the expression below

$$3 * x/(8 - 3) + x * (4 + 2)$$

can be represented as



Question

In this assignment, **you are supposed to implement an algebraic expression tree using the binary tree data structure**. The data structure will represent a single arithmetic expression, which will be given into the constructor as a character array/pointer (C-style string) parameter. In the constructor, this parameter will be parsed and a binary tree will be constructed. Then, a user will be able to perform tasks on the constructed tree, including expression evaluation, expression conversion, and tree display.

In your implementation, you are supposed to write a separate function for each of these tasks. The details of these functions (tasks) are given below:

1. `AlgExpressionTree(char *expression)`

   This is the constructor that takes a character array/pointer `expression` in the prefix notation and parses this expression in order to construct its corresponding algebraic tree.

   For simplicity, you may assume the following:

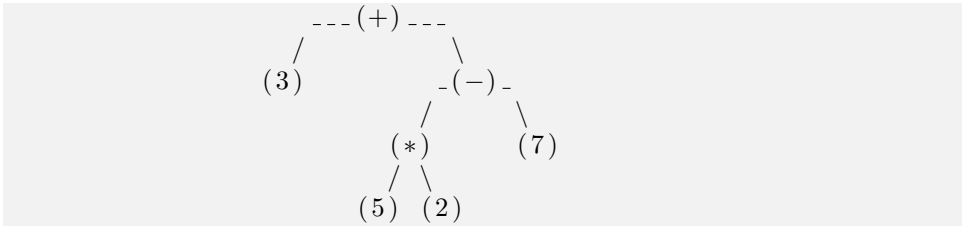
   - An expression contains variables with single-character names (such as x, y, and z) and single-digit integers $(0, ..., 9)$. However, in the expression, there may exist multiple occurrences of the same or different variable names and multiple occurrences of the same or different integers.
   - An expression contains the basic four operators, which are $+, -, *, /$
   - In an expression, no whitespace is allowed in between operators and operands.

   Please note that once you have obtained the algebraic tree for a given expression, the other tasks, which are explained below, become just a

matter of traversing this tree. Thus, please be sure that you implemented this constructor correctly.

2. `void displayTree()`

   This function displays the tree in a graphical format as given in the example output. The function should write every node at the same level into the same column. That is the output of the displayTree function will be the 90 degree rotated version of the original tree display without lines between keys. For example, a tree that is constructed from an expression $3 + (5 * 2 - 7)$ which have a prefix notation $+3 - *527$ will originally have a display given below.

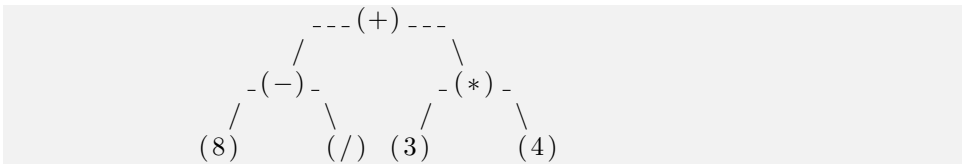


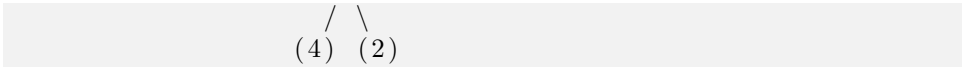
   To make it easier, the output of the 'displayTree' function should be as follows:

```
                7
        -
                        2
                *
                        5
+
        3
```

   Similarly, the original display and the output of a tree constructed from an expression $(8-4/2)+3*4$ which have a prefix notation $+-8/42*34$ is given below as another example.

   Display:





Output:

```
                4
        *
                3
+
                        2
                /
                        4
        -
                8
```

3. `void displayPostfix()`

   This function displays the postfix notation of the expression represented by the algebraic tree.

   IMPORTANT NOTE: In your program, you should make this conversion on the algebraic expression tree. If you do it on the corresponding expression string (as you did in your CS 201 class), you will get NO POINTS for this part.

4. `void setVariable(char varName, int varValue)`

   This function assigns an integer value `varValue` to the variable whose name is `varName`. If the variable with this name does not exist in the expression, this function will not make any assignment.

   Note that this function is NOT permanently assign a value to the given variable. Thus, the value of any variable can change many times during the program, calling this function repeatedly. See the example program and output below for better understanding of how this function works.

5. `double evaluate()`

   This function evaluates the expression represented by its algebraic tree and returns the result. For each particular variable, it uses the integer value assigned by the most recent call of the `setVariable` function for this particular variable. If the `setVariable` function has not been

called for a variable, it uses the default integer value of 0. Again see the example program and output below for better understanding of how this function works.

IMPORTANT NOTE: In your program, you should make these evaluations on the algebraic expression trees. If you evaluate them on the corresponding expression strings (as you did in your CS 201 class), you will get NO POINTS for this part.

Below is the required public part of the `AlgExpressionTree` class that you are going to implement for this assignment. The name of the class MUST be `AlgExpressionTree`, and it MUST include the following public member functions. We will use these functions to test your code. The interface for the class must be written in a file called AlgExpressionTree.h and its implementation must be written in a file called AlgExpressionTree.cpp. In your implementation, you may define additional public and private member functions and data members of this class. You may also define additional classes in your solution.

```
class AlgExpressionTree {

public:
     AlgExpressionTree(char *expression);    // constructor
     ~AlgExpressionTree();                   // destructor

     void displayTree();
     void displayPostfix();
     void setVariable(char varName, int varValue);
     double evaluate();
}
```

Sample Program and Output:

```
#include <iostream>
#include "AlgExpressionTree.h"

int main(){
     AlgExpressionTree T1("+*3x5");
```

```
    AlgExpressionTree T2("++y9z");

    T1.displayTree();
    cout << "Result: " << T1.evaluate() << endl;
    // evaluates the expression for x = 0 (default value)

    T1.setVariable('x', 8);
    T1.displayTree();
    cout << "Result: " << T1.evaluate() << endl;
    // evaluates the expression for x = 8,
    // but it does not change the tree

    T1.setVariable('x', 6);
    cout << "Result: " << T1.evaluate() << endl;
    // evaluates the expression for x = 6,
    // but it does not change the tree

    T1.setVariable('y', 10);
    cout << "Result: " << T1.evaluate() << endl;
    // the setVariable function does not do anyhing
    // since y does not exist in the expression and
    // the evaluate function uses the most recent value of x

    cout << "Postfix form: ";
    T1.displayPostfix();

    T2.setVariable('z', 10);
    cout << "Result: " << T2.evaluate() << endl;
    // evaluates the expression for z = 10
    // but uses the default value for y, which is 0

    T2.setVariable('y', 3);
    cout << "Result: " << T2.evaluate() << endl;
    // evaluates the expression for z = 10 and y = 3

    return 0;
}
```

```
OUTPUT:

+
*5
3x$$

Result: 5

+
*5
3x$$

Result: 29
Result: 23
Result: 23

Postfix form: 3x*5+

Result: 19
Result: 22
```

Bonus Question (20 points)

The implementation of the `displayPostfix` function is quite straightforward since both the prefix and postfix notations do not require parentheses. However, displaying the infix notation of an expression that is represented by an algebraic tree is somewhat more complex since you need to use parentheses for precedence and associativity.

In this bonus part, you are asked to implement the `displayInfix` function that finds the infix notation of an expression represented by the algebraic tree and displays this notation on the screen. Please revise your CS 201 course materials, if necessary.

IMPORTANT NOTES:

Do not start your homework before reading these notes!!!

1. This assignment is due by 23:59 on Thursday, April 3rd, 2014. You should send your homework to Gökçen Çimen, by email before the deadline. No hardcopy submission is needed.

2. The standard rules about late homework submissions apply. Please see the course syllabus for further discussion of the late homework policy as well as academic integrity.

3. You ARE NOT ALLOWED to modify the given parts (given member functions) of the `AlgExpressionTree` class. However, if necessary, you may define additional data members and member functions. Moreover, you ARE NOT ALLOWED to use any global variables.

4. Your code must not have any memory leaks. You will lose points if you have memory leaks in your program even though the outputs of the operations are correct. Thus, do not forget to implement the destructor.

5. For this assignment, you must use your own implementation of binary trees. In other words, you cannot use any existing binary tree code from other sources such as the tree class in the C++ standard template library (STL). However, you are allowed to use the codes given in our textbook and/or our lecture slides. However, you ARE NOT ALLOWED to use any codes from somewhere else (e.g., from the internet, other text books, other slides ...).

6. **You MUST use a pointer-based implementation in your solution.** You will get no points if you implement array-based solutions (using fixed-sized arrays, dynamically allocated arrays, data structures such as vector from the standard library, etc.).

7. In this assignment, you must have separate interface and implementation files (i.e., separate .h and .cpp files) for your class. We will test your implementation by writing our own driver .cpp file which will include your header file. For this reason, your class' name MUST BE `"AlgExpressionTree"` and your files' name MUST BE "AlgExpressionTree.h" and "AlgExpressionTree.cpp". You should send these two

files and any additional files if you wrote additional classes in your solution as a single archive file (e.g., zip, tar, rar).

8. We also recommend you to write your own driver file to test each of your functions. However, you MUST NOT submit this test code (we will use our own test code). In other words, do not submit a file that contains a function called `"main"`.

9. You are free to write your programs in any environment (you may use either Linux or Windows). On the other hand, we will test your programs on `"dijkstra.ug.bcc.bilkent.edu.tr"` and we will expect your programs to compile and run on the `"dijkstra"` machine. If we could not get your program properly work on the `"dijkstra"` machine, you would lose a considerable amount of points. Therefore, we recommend you to make sure that your program compiles and properly works on `"dijkstra.ug.bcc.bilkent.edu.tr"` before submitting your assignment.

10. The submissions that do not obey these rules will not be graded.

11. To increase the efficiency of the grading process as well as the readability of your code, you have to follow the following instructions about the format and general layout of your program.

    - Do not forget to write down your id, name, section, assignment number or any other information relevant to your program in the beginning of your main file. Example:

```
//-------------------------------------------------------
// Title: Algebraic Expression Trees
// Author: Gokcen Cimen
// ID: 2100000000
// Section: 0
// Assignment: 2
// Description: This program is to construct the
// algebraic expression tree from a given expression
// and to perform a set of related tasks
//-------------------------------------------------------
```

    - Since your codes will be checked without your observation, you should report everything about your implementation. Well comment your

classes, functions, declarations etc. Make sure that you explain each function in the beginning of your function structure. Example:

```
void void setVariable(char varName, int varValue)
//-------------------------------------------------------
// Summary: Assigns a value to the variable whose
// name is given.
// Precondition: varName is a char and varValue is an
// integer
// Postcondition: The value of the variable is set.
//-------------------------------------------------------
{
        // body of the function
}
```

- Indentation, indentation, indentation...
- Pay attention to these instructions, otherwise you may lose some points even though your code has no error.

12. This homework will be graded by your TA, Gökçen Çimen, (gokcen.cimen at bilkent edu tr). Thus, you may ask your homework related questions directly to her.