CS2134 HOMEWORK 8
Due* Wednesday Nov 16, 2016 at 5:00 p.m.

Be sure to include your name at the beginning of each file! Assignment 8 includes a programming portion and a written part. The programming portion must compile and consist of a single file (hw08.cpp). The typed portion should consist of a single file (hw08written.pdf). It *must* be in a .pdf format. Be sure to include your name at the beginning of each file! You must hand in the file via NYU Classes.

1 Programming Part

1. Write the code to check for balanced parenthesis in a C++ program. If the C++ program has a mismatched parenthesis[1], your code prints out the line number where the mismatched[2] occurred.

   To do this you will finish writing the `Balance` class. The signature for the `Balance` class is in a file called `Balance.cpp`.

   The `Balance` class has a single public method called `checkBalance` that:

   - takes no arguments and returns the number of mismatched parenthesis. The method `checkBalance` will print an error[3] message and update the error count if the stack is *empty* and a closing parenthesis is found, or the *end of the file* is reached and the stack is not empty
   - uses the `tokenizer` class to pull out the parenthesis from the C++ program. The `tokenizer` class is in a file called `tokenizer.cpp`
   - calls a method called `checkMatch` that compares two parenthesis to see if they match. This method prints an error message if the parenthesis are mismatched, and updates the error count

   Check that your algorithm works on your own C++ program.

2. In this part, you will store a list of correctly spelled words and a point value associated with each word into a variable of type `map<string, int>`. You will perform the following steps:

   - Enter the point value from a file called `Letter_point_value.txt` for each letter into a variable of type `vector<int>`. The point value associated with 'A' goes into position 0, and 'B' goes into position 1, etc.
   - The point value of a word is determined by the point value of each letter. To compute the point value of a word, add up the point values of each letter in the word. For example, the point value of "cat" is 4 + 1 + 1 = 6, since 'C' has a point value of 4, 'A' has a point value of 1, and 'T' has a point value of 1. Upper and lower case letters have the same point value. (e.g. So "CAT" also has a point value of 6.) Create a function to compute the point value of a word.

*5% extra credit will be given if you turn this assignment in on Tues Nov 15 at 11:00 p.m.

[1] I encourage you to expand your code to find mismatched brackets.

[2] For us, the mismatched occurred where you first realized the parenthesis did not have a match. See the lecture slides for clarification.

[3] The method `checkBalance` only checks some of the possible errors. The method `checkMatch` will check if the parenthesis do not match.

- The words you will store are in a file called `ENABLE.txt`. You will read in each word and store it and its associated point value in a variable of type `map<string,int>`.

On your own[4]. You can write a program to help you play Words With Friends.

Remember the recursive function from the extra credit problem in a previous assignment, where the user enters a string and you find all the combinations of the string.

For each string created from the recursive function, you can test to see if it is in the ENABLE word list. If so, you print out the word and the points associated with the word. Use the `map<string,int>` you created in programming part 2.

[4]Do not turn this in

2 Written Part

1. For each of the following infix expressions, illustrate the operation of the algorithm for converting infix to postfix. Show the contents of the stack, and the output stream, after each token from the input is processed. If the infix expression is not syntactically valid, give an error message:

```
a.)  1 -  2  +  3  ^  2
b.)  (2 ^ 3) ^ 2
c.)  2 ^ 3 ^ 2
d.)  (2  +  6)  /  3 - (32  +  4  *  7) * 2
e.)  3 + 2 - 4 + 5
f.) (3 + 2) ^ 4 ^ (3 * 2 + 4)
```

2. For each of the following postfix expressions, illustrate the operation of the stack-based evaluation algorithm. Show the contents of the stack after each operand or operator symbol from the input is processed. Additionally, indicate the value of the expression or give an error message if the expression is not syntactically valid.

   EXAMPLE: input 1 2 3 * +

```
        3
     2  2  6
  1  1  1  1  7  stack (growing up)
```

   If you prefer, you can write the input vertically and the stack growing from left to right, as follows (easier to type!):

```
input   stack
1       1
2       1 2
3       1 2 3
*       1 6
+       7
```

```
a.)  4 2 + 3 3 ^ -
b.)  3 2  ^  3  2 * -
c.)  4 2 3 * - 3 2 ^ - 6 +
d.)  4 3 +  2 * 1   -
e.)  3  5  *  1 +  4  /  6  +
```

3. Add `%` to the `PREC_TABLE` table[5] and to the `TokenType` so that `%` precedence can now be determined using `PREC_TABLE`.

```
enum TokenType { EOL, VALUE, OPAREN, CPAREN, EXP,
                 MULT, DIV, PLUS, MINUS };
// PREC_TABLE matches order of Token enumeration
struct Precedence
{
```

[5] % has the same precedence as * and /.

```
    int inputSymbol;
    int topOfStack;
};
vector<Precedence>  PREC_TABLE  =
{
    { 0, -1 }, { 0, 0 },       // EOL, VALUE
    { 100, 0 }, { 0, 99 },     // OPAREN, CPAREN
    { 6, 5 },                  // EXP
    { 3, 4 }, { 3, 4 },        // MULT, DIV
    { 1, 2 }, { 1, 2 }         // PLUS, MINUS
};
```

4. Show what is printed by the following code and show the contents of the stack just before the program terminates. (This code has been modified/simplified from the code in the lecture slides.)

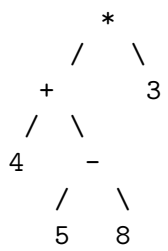
```
enum TokenType { EOL, VALUE, OPAREN, CPAREN, EXP, MULT, DIV, PLUS, MINUS };

// PREC_TABLE matches order of Token enumeration
struct Precedence
{
int inputSymbol;
int topOfStack;
};
// and
vector<Precedence>  PREC_TABLE =
{
{ 0, -1 }, { 0, 0 },       // EOL, VALUE
{ 100, 0 }, { 0, 99 },     // OPAREN, CPAREN
{ 6, 5 },                  // EXP
{ 3, 4 }, { 3, 4 },        // MULT, DIV
{ 1, 2 }, { 1, 2 }         // PLUS, MINUS
};
int main ( ) {

    stack<TokenType> opStack;
    opStack.push(EOL); // EOL == end of line
    opStack.push(PLUS);
    opStack.push(MULT);
    opStack.push(EXP);
    opStack.push(EXP);
    TokenType topOp;
    TokenType lastType = DIV;
    while( PREC_TABLE[ lastType ].inputSymbol <= PREC_TABLE[ topOp = opStack.top( ) ].topOfStac
    {
        opStack.pop();
        cout  << topOp << endl;
    }
    if( lastType != EOL )
        opStack.push( lastType );
// show what are the contents of opStack at this point in the code.
```

```
return 0;
}
```

5. For the following tree:



   (a) what is the value of the root node?
   (b) which node is the sibling of 4?
   (c) which nodes are leaf nodes?
   (d) which nodes are internal nodes?
   (e) what is the height of the node containing '-'?
   (f) what is the depth of the node containing '-'?
   (g) what is the size of the tree?
   (h) which nodes are the children of the node containing '+'?
   (i) which node is the parent of the node containing '-'?
   (j) what is the output of an inorder traversal of the tree?
   (k) what is the output of an preorder traversal of the tree?
   (l) what is the output of an postorder traversal of the tree?

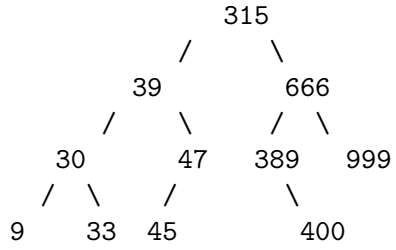
6. What does the height of a binary search tree mean in relation to its searching efficiency?

7. How many different binary trees can be made from three nodes that contain the values 1, 2, 3?

8. Given the implementation of the binary search tree presented in class, what is the best order to insert the following numbers $\{0, 1, 2, 3, 4, 5, 6, 7\}$ so that the tree has:

   - minimal height. Show the tree that would be created if they were added in that order.
   - maximal height. Show the tree that would be created if they were added in that order.

9. Consider the following binary search tree:



   (a) Show what happens when 315 is removed.
   (b) Show what happens when 39 is removed (to original tree, not the tree after 315 was removed).
   (c) Show what happens when 398 is added (to original tree, not the tree after 315 or 39 was removed).