CS2134 HOMEWORK 7
Due* Tues April 5, 2016 at 11:00 p.m.

Be sure to include your name at the beginning of each file! Assignment 7 includes a programming portion and a written part. The programming portion must *compile* and consist of a single file (hw07.cpp). The written portion should consist of a single file (hw07written.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.

*10% extra credit will be given if you turn this assignment in on Monday April 4, 2016 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.

2 Written Part

1. Using the circular array implementation of a *queue*, with an array of size 4, show the array after each operation in the sequence below. (Show the array, and the values of `theFront`, `theBack`, and `currentSize`. Also show the value of variable `x`.) Assume that the popped item is no longer visible (i.e. we do not explicitly remove the popped items, but we conceptually think of them as having been removed.)

```
int x;
queue<int>q; //[ , , , ],theFront=0,theBack=3, currentSize=0,x=?
q.push(1);
q.push(2);
q.push(3);
x = q.front();
q.pop();
q.push(4);
x = q.front();
q.push(5);
q.pop();
q.push(6);
x = q.front();
q.pop();
q.pop();
q.pop();
x = q.front();
q.pop();
q.push(7);
```

2. 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
```

3. 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  +
```

4. Add % to the PREC_TABLE table[4] 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
{
    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
};
```

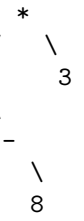
5. Show what is printed by the following code and show the contents of the stack just before the program terminates. (This code has been modified/simplified from the code in the lecture slides.)

```
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;
```

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

```
    while( PREC_TABLE[ lastType ].inputSymbol <= PREC_TABLE[ topOp = opStack.top( ) ].topOfStack )
    {
        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;
}
```

6. 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 `'-'`?