CS2134 HOMEWORK 5
Spring 2015
Due 11:59 pm on Tues. April 7, 2015

Your third assignment includes both a programming portion and a written portion. The programming portion should consist of a file called hw035.cpp. The written portion should consist of files called hw05written saved in a standard document format (.txt, .doc, .htm., or .pdf). Be sure to include your name at the beginning of each file! *You must hand in both files via NYU Classes.*

PROGRAMMING PART:

1. Create methods for a list implementation that is closer to the STL `forward_list` interface than the code presented in class. You will find the code in a file `simple-list.txt`.

   You will implement the following four methods for the `LList` class:

   - The method `push_front( const Object & x )` adds `x` to the front of the list. It performs as stated in `http://www.cplusplus.com/reference/forward_list/forward_list/push_front/`. This method must take $O(1)$ time.
   - The method `pop_front()` removes the first item in the list. It performs as stated in `http://www.cplusplus.com/reference/forward_list/forward_list/pop_front/`. This method must take $O(1)$ time.
   - The method `size()` that returns the size of the list. You will add another `private` member variable to the class so that this method takes `O(1)` time.[1]
   - The method `erase_after` that has a signature:
     `LListItr<Object> erase_after ( LListItr<Object> position )` and performs as stated in `http://www.cplusplus.com/reference/forward_list/forward_list/erase_after/`. Your method must run in `O(1)` time.
   - The method `remove_if` that performs as stated in `http://www.cplusplus.com/reference/forward_list/forward_list/remove_if/`. This method must run in `O(n)` time. Your method *should* call your method `erase_after`.
   - Change the method `remove_last(const Object & x)` to remove the last occurrences of `x` in the list.

   You will also implement the five methods in the `LListItr` class in the style of the STL iterators. The methods are for advancing an iterator, dereferencing an iterator, checking if two iterators refer to the same node, and checking if two iterators do not refer to the same node.[2]

   We will test your code by including it in a *driver* program that uses the methods you implemented. You must test your code by writing and executing your own driver. Remember to hand in your driver code for any programming assignment.

[1]Update all the methods as needed so the member variable you added holds the size of the class.
[2]The interface for this class has been provided.

WRITTEN PART:

1. Draw an empty linked list with a header implementation.
2. Draw pictures showing how the links change for programming question 1 in this homework assignment, and provide the pseudo code for the following methods:
   (a) For the method `push_front`, show the before and after picture.

   Write the pseudo code for the main step(s) needed to implement this method.

   (b) For the method `pop_front`, show the before and after picture.

   Write the pseudo code for the three to six main steps needed to implement this method.

   (c) For the method `erase_after`, show the before and after picture.

   Write the pseudo code for the three to six main steps needed to implement this method.[3]

3. Which of the following code snippets are valid? If the code snippet is invalid, state why.

   (a)
```
list<int> l;
list<int>::iterator lIter;

l.push_back(200);
lIter = l.begin();

for (int i = 1; i < 100; ++i)
  l.push_front(i);
for (int i = 1; i < 100; ++i)
  l.push_back(-i);

cout << *lIter << endl;
```

[3]Don't forget to update the member variable holding the size.

(b)
```
list<int> l;
list<int>::iterator lIter1;
list<int>::iterator lIter2;
list<int>::iterator mid;

for (int i = 0; i < 100; ++i)
   l.push_back(i);

lIter1 = l.begin();
lIter2 = l.end();
mid = lIter1 + (lIter2 - lIter1)/2;
cout << *mid << endl;
```

(c)
```
vector<int> v;
vector<int>::iterator vIter1;
vector<int>::iterator vIter2;
vector<int>::iterator mid;

for (int i = 0; i < 100; ++i)
   v.push_back(i);

vIter1 = v.begin();
vIter2 = v.end();
mid = vIter1 + (vIter2 - vIter1)/2;
cout << *mid << endl;
```

4. Add `%` to the `PREC_TABLE` table and to the `TokenType` so that `%` precedence can now be determined using `PREC_TABLE`. [4]

```
enum TokenType { EOL, VALUE, OPAREN, CPAREN, EXP,
                 MULT, DIV, PLUS, MINUS };
// PREC_TABLE matches order of Token enumeration
struct Precedence
{
    int inputSymbol;
    int topOfStack;
} 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] % has the same precedence as * and /.

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 book.)

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

// PREC_TABLE matches order of Token enumeration
struct Precedence
{
int inputSymbol;
int topOfStack;
} 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(DIV);
    opStack.push(EXP);
    opStack.push(EXP);
    TokenType topOp;
    TokenType lastType = MULT;
    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 `LList class`, what if the following code for the method `remove` was used. Would it work correctly? Explain.

```
// Remove the first occurrence of an item x.
template <class Object>
void LList<Object>::remove( const Object & x )
{
    LListNode<Object> *p = findPrevious( x ).current;
```

```
        if( p->next != NULL )
        {
            LListNode<Object> *oldNode = p->next;
            delete oldNode;
            p->next = p->next->next;  // Bypass deleted node
        }
    }
```

7. 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.)  8 4 2 / 3 3 ^ - +

b.)  2 4  ^  2  6 * -

c.)  4 2 3 * - 3 2 ^ - 6 +

d.)  9 3 /  2 * 1   -

e.)  2  5  /  8 +  7  *  4  +
```

8. 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.)  (5 -  2)/3  +  3  ^  2
b.)  8 + (2 ^ 3) ^ 2
```

```
c.)  2 ^ 3 ^ 2 + 8
d.)  (2  +  6)  /  3 - ( (32  +  4)  *  7) * 2
e.)  3 * 2 / 4 * 5
```