CMPS 2143: Object Oriented Programming
Programming Assignment 5: 100 points **DUE: Friday, Dec 6, 2019 11am**

Purpose: To use turn a class into a template class and use it. To reinforce inheritance and polymorphism.

Problem: Read in a list of Shapes and their dimensions from a file. Then print out each shape and its area and the total area of *all* the shapes.

Method: Your instructor will give you some code, but it is incomplete. You need to modify the application to convert the List class to a template class, complete all the Shape classes in the Shape.h file (do them all inline) and then declare a List of pointers to Shapes in your main program.

Input: Input the names of the input and output files from the keyboard. The input file will consist of a list of shapes: Rectangle, Oval, Square, Circle with the appropriate number of dimensions.

Input file Sample:

```
Oval  3  5
Square 5
Square 3
Rectangle 2 3
Circle 6
Circle 5
Oval 1 1
Square 0
Square 10
Rectangle 2 5
Rectangle 5 6
```

Output file Sample:

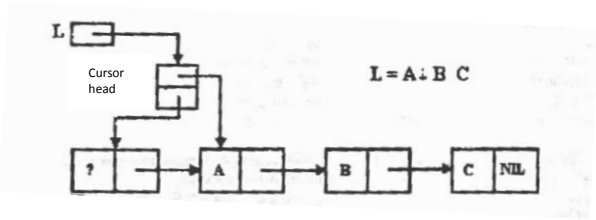
```
Catherine Stringfellow
Program 5

Shape is Oval.        Area is 47.100000.
Shape is Square.      Area is 25.000000.
Shape is Square.      Area is 9.000000.
Shape is Rectangle.   Area is 6.000000.
Shape is Circle.      Area is 113.040000.
Shape is Circle.      Area is 78.500000.
Shape is Oval.        Area is 3.140000.
Shape is Square.      Area is 0.000000.
Shape is Square.      Area is 100.000000.
Shape is Rectangle.   Area is 10.000000.

Total area of all shapes is 391.78
```

Output: Output is to a file.

Turn in: Printouts of:

- List.h, List.cpp files
- Shapes.h
- Prog5.cpp
- 2 input files and corresponding output file

Un-templated List.h




```
class List
{
  struct ListNode
  {   ItemType item;
      ListNode * next;
  };

public:

// requires : List is not created
// ensures : List is created and empty
List ( );

 // requires: this list and other are created
// ensures:  other is copied to this list
List (List & other);

// requires : List is created
// ensures : List is not created
~List ();

//requires: List is created
//ensures:  List is empty
void ClearList ();

//requires: this list and other are created
//ensures:   this = #other and other = #list, except that cursors are at heads
void SwapLists(List & other);

// requires : List is created
// ensures : List's cursor is placed before the first item in the sequence,
//         and the sequence is unchanged
void ResetCursor ();

// requires : L is created
// ensures : returns true iff L's cursor is located after
//         the last item in the sequence.  L is not changed
bool CursorAtEnd ();

// requires : List is created and the cursor is not at the end of the sequence
// ensures : Return in Item the item referenced by L's cursor.  The sequence
//         is unchanged and the cursor is not moved
// checks :  if Cursor is at the end of the list, write error message
void GetCurrentItem (ItemType & Item);

// requires : L is created and the cursor is not at the end of the sequence
// ensures : the item referenced by L's cursor is set to the value of I;
//          The sequence is unchanged otherwise and the cursor is not moved
// checks : if Cursor is at the end of the list, write error message
void UpdateCurrentItem (const ItemType & Item);
```

```
// requires : List is created and the cursor is not at the end of the sequence
// ensures : List's cursor is advanced one position in the sequence and the
//         sequence is unchanged
// checks : if Cursor is at the end of the list, write error message
void AdvanceCursor ();

// requires : List is created
// ensures : Item is inserted at the position of L's cursor. Otherwise the
//         sequence is unchanged and the cursor is located before the
//         inserted item
void InsertItem (const ItemType & Item);

// requires : List is created and the cursor is not at the end of the sequence
// ensures : the item at the position referenced by L's cursor is deleted.
//         Otherwise the sequence is unchanged and the cursor is located
//         before the next item in the sequence
// checks : if cursor is at the end of the list, write error message
void DeleteItem ();

//requires: List is created
//ensures: the number of items in the list is returned
int getCount();

private:
  ListNode * head;
  ListNode * cursor;
  int count;
};
```