Lab 2: ADT Design & Implementation

By Dr. Yingwu Zhu, Seattle University

1. Goals

In this lab, you are required to use a dynamic array to design and implement an ADT **SortedList** that maintains a ***sorted list*** of integers. This lab aims to further your understanding of the basic concepts we've discussed in class. The **SortedList** differs from the **List** class we've discussed in class in that it stores its elements in a non-decreasing order.

2. Design

In the design stage, you need to think about two questions: ***(1) What data members will be in this ADT? (2) What operations (or member functions) will be provided by this ADT?***

(1) Data member:

(2) Operations:

Stop here! Answer the two questions above!

==========

Three data members:

- Array capacity
- Array pointer
- # of elements

Basic operations:

- Constructor (with a parameter specifying the array capacity)
- Destructor
- *bool empty()* which return true if the list is empty; false otherwise
- *void insert(ElementType x)* which inserts a new element x into the list and maintains the non-decreasing order
- *void remove(ElementType x)* which deletes an element x if it exists
- *bool found(ElementType x)* which returns true if *x* exists on the list; false otherwise
- overload output operator<< to display the elements on the list

Note: **typedef int ElementType;**

3. Implementation

In this stage, you need to think about data structures for storing the data members and algorithms for implementing the operations.

For the data structures, it is quite straightforward: we use a dynamic array to store the elements.

Now, it is time for you to come up with the class declaration for **SortedList** in **sortedlist.h**

Under your **lab2** directory, please complete the header file **sortedlist.h**

Stop here! Complete the header file! (10 points)

==========

```
/*
   Author: xxxxx (your name)
   Program: sortedlist.h
   Last Modification: xxx (date)
*/

/*
   Description: this header file is for SortedList
*/
#ifndef   _SORTED_LIST_H
#define   _SORTED_LIST_H
#include <iostream>
using namespace std;
typedef    int    ElementType;
class SortedList {
   private:
            int   capacity;
            int   size;
            ElementType*  array;
   public:
            SortedList(int  c);   //constructor
            ~SortedList();   //destructor
            bool  empty();
            bool  found(ElementType x);
            void  insert(ElementType x);
            void  remove(ElementType x);
            void  display(ostream& out);
};

ostream&  operator<<(ostream& out, const SortedList& slist);
#endif
```

Checklists:

- Redundant declaration?
- Data members properly declared?
- Member functions properly declared?
- Operator<< overloading properly declared?
- Semicolon in the end of class declaration?

If all your answers to the checklists are YES, congratulations!

Next, you are going to implement the **SortedList** in **sortedlist.cpp** step by step!

Step 1: Implement constructor and destructor (10 points)

Thinking: What should be done in constructor and destructor? When will they be called?

Step 2: Test the constructor and destructor (10 points)

In this step, you need to create a client program **test.cpp** and a **Makefile**. In **test.cpp**, you should test **SortedList**'s constructor and destructor.
Note: You should comment out those member functions and operator<< overloading function that you have NOT implemented, in order to compile your program successfully. Also reuse your Makefile in Lab1 by modifying the involved file names.

```
/*
   Author: xxxxx (your name)
   Program: sortedlist.h
   Last Modification: xxx (date)
*/

/*
   Description: this header file is for SortedList
*/
#ifndef   _SORTED_LIST_H
#define   _SORTED_LIST_H
#include <iostream>
using namespace std;
typedef    int    ElementType;
class SortedList {
   private:
            int   capacity;
            int   size;
            ElementType*  array;
   public:
            SortedList(int  c);   //constructor
            ~SortedList();   //destructor
            /*
            bool  empty();
            bool  found(ElementType x);
            void  insert(ElementType x);
            void  remove(ElementType x);
            void  display(ostream& out);
            */
};

// ostream&  operator<<(ostream& out, const SortedList& slist);
#endif
```

Step 3: Implement insert() (10 points)

Step 4: Overload operator<< (10 points)

Step 5: Test if insert() and operator<< overloading work properly (10 points)

Step 6: Implement remove() and test it (10 points)

Step 7: Implement found() and test it (10 points)

Step 8: Revisit found() (10 points)

What search algorithm is used in this function? Can you use a more efficient algorithm if your algorithm is linear time?

Stop here! Try to come up with a more efficient search algorithm!

==========

Yes. You should use a binary search algorithm. Improve your found() with a binary search algorithm.

How many points have you got?

4. Extra lab exercises

If you have successfully completed the above steps, you may implement **SortedList** using a different data structure **linked-list**.