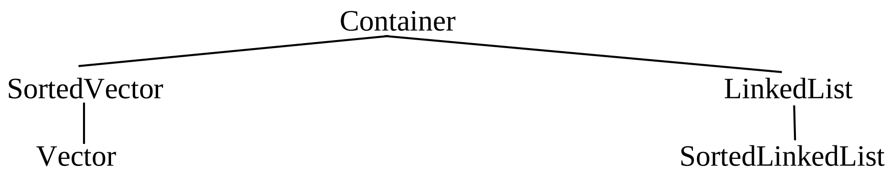
ECS 40 **Program #6 (50 pts, 2 hours)** **Spring 2016**

Due:Wednesday, May 11th, 11:59 PM Executable name: containers.out Handin to cs40a p6
Filenames: authors.csv, Makefile, main.cpp, container.cpp, container.h, linkedlist.cpp, linkedlist.h, sortedlinkedlist.h, sortedlinkedlist.cpp, sortedvector.h, sortedvector.cpp, vector.h, and vector.cpp.
New concepts: doubly linked list, inheritance, polymorphism

For this assignment, you will be creating six classes. There are five container classes that have the hierarchical relationship depicted below. Container is an abstract class. The other four container classes, are concrete classes that implement all the methods promised by the Container as well as other methods. The ListNode class will serve both the LinkedList and SortedLinkedList class.

main.cpp is designed to allow you to do incremental development. The original main.cpp will not compile until you have implemented the Container class. To test your SortedVector class you need only uncomment the lines indicated in main.cpp. You can find main.cpp, my containers.out, and testing files in ~ssdavis/40/p6.



Specifications, and order of development. I implemented them in the order in which they are presented.

1. Container class
   1.1. Contains only a protected int named size.
   1.2. Concrete public methods
      1.2.1. Container (int siz) is the only constructor. Both directly derived class constructors will have to call this function from their initialization list.
      1.2.2. int getSize(); which returns the number of items currently stored in the container, i.e. the value of size.
   1.3. Virtual public method
      1.3.1. Destructor that does nothing. A virtual destructor is required whenever a class has any virtual method.
   1.4. Pure virtual public methods (not implemented in container.cpp)
      1.4.1. int* insert(int num) which inserts the num into the container, and returns a pointer to the int in the container.
      1.4.2. int* erase(int num) which tries to remove the num from the container. Returns a pointer to the int after the sought num if there is one, else NULL. If the num is not found, then return NULL.
      1.4.3. int* find(int num) prints the list of the values confronted on the way to finding num. Note the last number printed is followed by a space before the endl. If num is found, then it returns a pointer to the found int, else it returns NULL. If there are duplicated values, it will return a pointer to the first found.
   1.5. Your code should now compile with no errors, nor warnings. Don't run it though.
2. SortedVector class that is publicly derived from the Container class.
   2.1. Uncomment the lines indicated in main.cpp.
   2.2. The class has a private int capacity, a protected int* named array, and a public getCapacity() method.
   2.3. The array is maintained in sorted order, from smallest to largest.
   2.4. A good way to save time is to copy the pure virtual function prototypes from the Container class into the SortedVector class in sortedvector.h.
      2.4.1. Now get rid of the three " = 0".
      2.4.2. Now copy the prototypes from sortedvector.h into sortedvector.cpp to ensure that the signatures all match, and add {} to make them functions.
      2.4.3. Add "return NULL;" to each of the functions that you copied. Your code should now compile without warnings.
   2.5. The default constructor of SortedVector initializes the Container constructor, and initializes array to NULL.
   2.6. The destructor removes the array.
   2.7. The linear searches of insert(), erase(), and find() should rely on the fact that the array is sorted. There is no need to use a binary search for this program. You may not use "==", nor "!=" anywhere in sortedvector.cpp. You need to use only the "<" of int to determine equality and inequality.
   2.8. insert() inserts the int at the proper sorted position in the array. If the size == capacity before inserting the item, then it calls the resize() method of the SortedVector class.

2.9. void resize() is a protected method of the SortedVector class. If the capacity is zero, then it dynamically allocates the array to hold one int. If the capacity is not zero, then double the capacity of the array, and copy the old values into the new array. Therefore, the capacity would be 0, 1, 2, 4, 8….
2.10. erase() moves all succeeding items one position closer to the zeroth index.
2.11. Write the find() code and have the search start at array[0].
2.12. Add a const overloaded [] operator that return int&. If the index is outside the range of size, then the functions should print "`Virtual seg fault.\n`" to the screen, and return array[0].
2.13. Your code should run sortedVertex.txt without problem.

3. Vector class is publicly derived from the SortedVector class that is unsorted, and contains no new data members.
3.1. Uncomment the two lines dealing with Vector in main().
3.2. You may wish to copy sortedvector.h and sortedvector.cpp to become vector.h, and vector.cpp, though that will require a lot of editing once you've done it. I did that.
3.3. Only insert(), erase(), and find() need to exist in Vector. There is no need for a constructor, nor destructor.
3.4. These three methods in the class are not virtual.
3.5. insert() inserts the int at the end of the array.
3.6. The linear searches can no longer rely on the array being sorted.
3.7. Add a non-const overloaded [] operator that return int& that permits a value in the array to be changed. If the index is outside the range of size, then the functions should print "`Virtual seg fault.\n`" to the screen, and return array[0].

4. ListNode class contains an int data, ListNode* previous, and ListNode *next. It's only method is a private "standard" constructor. The LinkedList class, and the SortedLinkedList class are friends of ListNode.

5. LinkedList class is a doubly linked list publicly derived from the Container class.
5.1. The class contains three protected ListNode*: head, tail, and curr.
5.2. The constructor sets the three members to NULL.
5.3. The destructor removes all of the ListNodes of the list.
5.4. Try to apply what you learned in writing SortedVector to making the writing of LinkedList easier.
5.5. insert() inserts at the tail of the list, curr is set to point at the inserted int, and size is incremented.
5.6. erase() curr starts at the head of the list. If the parameter is found, remove the ListNode to which it points, have curr point to the next ListNode, and decrement size.
5.7. find() curr starts at the front of the list, and traverses the list until it finds the int, or reaches the end of the list.
5.8. It has a pre-increment operator that moves curr one forward, and returns a pointer to int to which curr is newly pointing, or NULL if curr is NULL.
5.9. It has a pre-decrement operator that moves curr one backward, and returns a pointer to int to which curr is newly pointing, or NULL if curr is NULL.

6. SortedLinkedList is publicly derived from LinkedList, and sorted from smallest to largest.
6.1. The linear searches of insert(), erase(), and find() should rely on the fact that the linked list is sorted.
6.2. You may not use "==", nor "!=" anywhere in sortedlinkedlist.cpp. You need to use only the "<" of int to determine equality and inequality.
6.3. insert() places the int at the correct sorted position in the list.

7. All methods must have the appropriate const declarations. Please list private, then protected, then public in classes.

8. Code must be submitted by exactly one member of each team. Double submissions, or errors in the authors.csv format will result in the team losing five points. Your handin command line will be:
handin cs40a p6 authors.csv Makefile container.cpp container.h linkedlist.cpp linkedlist.h sortedlinkedlist.h sortedlinkedlist.cpp sortedvector.h sortedvector.cpp vector.h vector.cpp

```
[ssdavis@lect1 p6]$ cat main.cpp
// Author: Sean Davis

#include <iostream>
#include <fstream>
#include "container.h"
#include "vector.h"                      // uncomment for Vector
#include "sortedvector.h"                // uncomment for SortedVector
#include "linkedlist.h"                  // uncomment for LinkedList
#include "sortedlinkedlist.h"            // uncomment for SortedLinked List
```

```
using namespace std;

int main(int argc, char** argv)
{
  char operation;
  int num, *intPtr, containerNum, index;
  Container *containers[4];
// SortedVector *vectors[2];                       // uncomment for SortedVector
// containers[0] = vectors[0] = new SortedVector;  // uncomment for SortedVector
// Vector *vectorPtr = new Vector;                 // uncomment for Vector
// containers[1] = vectors[1] = vectorPtr;         // uncomment for Vector
// LinkedList *lists[2];                            // uncomment for LinkedList
// containers[2] = lists[0] = new LinkedList;       // uncomment for LinkedList
// containers[3] = lists[1] = new SortedLinkedList; //uncomment SortedLinkedList

  ifstream inf(argv[1]);

  while(inf >> containerNum >> operation)
  {
    switch(operation)
    {
/*      case 'C' :   // uncomment for SortedVector
        cout << "Capacity: " << vectors[containerNum]->getCapacity() << endl;
        break;
*/                  // uncomment for SortedVector
      case 'E' :
        inf >> num;
        intPtr = containers[containerNum]->erase(num);
        cout << "Erase " << num << " : ";

        if(intPtr)
          cout << *intPtr << endl;
        else // nothing after num
          cout << "NULL\n";

        break;
      case 'F' :
        inf >> num;
        cout << "Find " << num << " : ";
        intPtr = containers[containerNum]->find(num);

        if(intPtr)
          cout << *intPtr << endl;
        else // not found
          cout << "NULL\n";

        break;
      case 'I' :
        inf >> num;
        intPtr = containers[containerNum]->insert(num);
        cout << "Insert " << num << " : " << *intPtr << endl;
        break;
      case 'S' :
        cout << "Size: " << containers[containerNum]->getSize() << endl;
        break;
      case 'R' :
        inf >> index;
//       num = (*vectors[containerNum])[index]; // uncomment for SortedVector
        cout << "Read [" << index << "] : " << num << endl;
        break;
      case 'W' :
        inf >> index >> num;
        cout << "Write [" << index << "] = " << num << endl;
//      (*vectorPtr)[index] = num;  // uncomment for Vector
        break;
/*    case 'M' :
        intPtr = --(*lists[containerNum - 2]);
        cout << "-- : ";

        if(intPtr)
          cout << *intPtr << endl;
        else  // moved beyound front of container
          cout << "NULL\n";

        break;
      case 'P' :
        intPtr = ++(*lists[containerNum - 2]) ;
        cout << "++ : ";

        if(intPtr)
          cout << *intPtr << endl;
        else  // moved beyond end of container
          cout << "NULL\n";

        break;
*/  // uncomment for LinkedList
    }  // switch
  }  // while more in file

  return 0;
} // main())
```

```
[ssdavis@lect1 p6]$ containers.out vectorTest.txt

Size: 0
Capacity: 0
Insert 13 : 13
Size: 1
Capacity: 1
Insert 8 : 8
Size: 2
Capacity: 2
Insert 2 : 2
Size: 3
Capacity: 4
Insert 9 : 9
Capacity: 4
Insert 4 : 4
Capacity: 8
Find 4 : 13 8 2 9 4
Read [2] : 2
Read [0] : 13
Virtual seg fault.
Read [6] : 13
Virtual seg fault.
Read [5] : 13
Virtual seg fault.
Read [-1] : 13
Erase 8 : 2
Size: 4
Find 9 : 13 2 9
Erase 2 : 9
Size: 3
Find 2 : 13 9 4 NULL
Write [1] = 1
Find 13 : 13
Write [4] = 3
Virtual seg fault.
Write [-3] = 5
Virtual seg fault.
Find 4 : 5 1 4
Size: 3
Capacity: 8
[ssdavis@lect1 p6]$
```