Practice for Exam

The best way to practice for the exam is to write code that uses the ideas we have been studying:

- Classes
- Linked lists
- Constructors
- Dynamic Memory Allocation
- Overloaded functions

For this lab you will add some more functions (aka methods) to the SortedList class you worked with in lab2.

For the lab work on the first few on the list. For practice, do a one or two more each day until the exam. As you do more and more, the ideas will get clearer, and your speed and accuracy will increase.

The Files

Start with the lab2 files you already worked on.

☞ Make a folder called lab4.

☞ Then copy the new files from

```
/comp/15/public_html/files/lab4
```

☞ Then copy the files from lab2 to lab4. On the linux Desktop, you can just open the two folders and copy the files with the usual Ctrl-C Ctrl-V sequence.

Or from the command line, you can type:

```
cd
cp Desktop/comp15/lab2/* Desktop/comp15/lab4
```

assuming those directory names and paths are what you used in your account. If these are not so, adjust to fit your locations.

You should now have the files:

```
main.cpp
main2.cpp
sortedlist.h
sortedlist.cpp
input1.txt
input2.txt
```

Compiling

To use the new main, compile with:

```
clang++ -g -Wall -Wextra main2.cpp sortedlist.cpp
```

and run it with

```
./a.out
```

Adding New Methods

There are lots of things you can do with a sorted list. Here is a long list of new methods you can write:

`int length()`
: length returns the number of elements in the list.

`int findMax()`
: findMax() returns the largest value in the list. If the list is empty, the function prints an error message and calls exit(1);

`double findMedian()`
: findMedian() returns the median value in the list. This function is declared to return a double. The reason is that the median value in a list with an even number of elements is the average of the two middle values. The list values are all ints, but the mean may have a fractional part. If the list is empty, the function prints an error message and calls exit(1);

`int freq(int val)`
: freq() returns the number of times the specified value appears in the list. If the value never appear, freq() returns 0. If the list is empty, freq() returns 0.

`SortedList whereis(val)`
: whereis() returns a new sorted list that contains the index values of all elements where val appears.

  For example, if the list `l` is 2 3 3 3 5 8
  then `l.whereis(3)` returns the list 1 2 3

`void merge(SortedList& l2)`
: merge() merges a second list into the original list. For example if the lists are:

  `l1` = 1 5 8
  `l2` = 1 2 5 10

  then after doing: `l1.merge(l2)`

  `l1` will be 1 2 5 5 8 10

`void removeVals(int v)`
: removeVals(int) takes a single integer value as an argument and modifies the list by deleting all elements that contain that value.

`void removeVals(SortedList& l2)`
: removeVals(SortedList&) is an overloaded version of removeVals(). If the argument is a SortedList, then this function removes from the original list all the values that appear in the list passed to it. For example if the lists are:

`l1` = 1 5 8
`l2` = 1 2 5 10

then after doing: `l1.removeVals(l2)`

`l1` will be 8

`SortedList findDuplicates()`

findDuplicates looks through a SortedList and returns a new sorted list that contains a list of which values appear more than once in the original list. For example given this list and this assignment:

`l1` = 1 2 2 3 4 4 4 5
`l2 = l1.findDuplicates()`

then

`l2` will be 2 4

`void removeDuplicates()`

removeDuplicates() modifies the calling array by removing all duplicate values, leaving only one instance of each value.

For example, given this list and this function call:

`l` is 2 3 3 3 5 8 8
`l.removeDuplicates()`,

then

`l` will contain 2 3 5 8

`SortedList findCommonValues(SortedList&)`

findCommonValues() is passed a second list and returns a new list that contains the values both lists contain. There are no duplicates in the result. For example:

`l1` = 1 5 8
`l2` = 1 2 5 10

then after doing:

`l3 = l1.findCommonValues(l2)`
`l3` will 1 5

`int getMaxFrequency()`

getMaxFrequency returns the largest number of times any value appears in the list. If the list is empty, this function returns 0. For example given this list:

`l1` = 1 2 2 3 4 4 4 5

then

`l1.getMaxFrequency()` returns 3

`int* toArray()`

toArray() returns a dynamically allocated array of ints that contains the values stored in the SortedList.

Adding Functions

Add one function at a time. To add a function, you need to do this:

[a] Add the function declaration to sortedlist.h
[b] Add the code to sortedlist.cpp
[c] Turn on the test in main2.cpp
  See main.cpp for how to do that
[d] Compile
[e] Fix syntax errors
[f] run
[g] debug

Provide What You Do

At the end of the lab time, please provide what you have.

```
provide comp15 lab4 main2.cpp sortedlist.{h,cpp}
```

Keep Working

These functions exercise all sorts of skills involving linked lists, loops, passing values, returning values. If you do one or two a day, you will be in good shape for the exam.