PONDER 05 : GO FISH!

Due Saturday at 11:59 PM MST

The fifth programming assignment is to write a program to play the game of “Go Fish.” This, and all future assigments for CS 235, will consist of two parts: the first part is to implement the data structure of the week, the second is to use the data structure to solve a problem.

Set

Create a class encapsulating the notion of a set. This will work exactly like the [std::set](http://www.cplusplus.com/reference/set/set/) class. Of course, any data-type will need to be supported so your class will be a template class. It will need to be defined in its own header file (set.h). The class name must be Set. Your class will need to support the following operations:

* **Constructors**: Default constructor (with the capacity set to zero initially), a non-default constructor (taking a capacity value as a parameter), and the copy constructor. If there is insufficient memory to allocate a new buffer, then the following exception is thrown:  
  ERROR: Unable to allocate a new buffer for Set.
* **Destructor**: When finished, the class should delete all the allocated memory.
* **operator=**: Assignment operator. This method takes a Set as a parameter and copies all the elements to this. If the current buffer size is sufficient, no allocation is made. If the current buffer size is not sufficient, enough space is allocated to accomodate the new data. If there is insufficient memory to allocate a new buffer, then the following exception is thrown:  
  ERROR: Unable to allocate a new buffer for Set. It also returns \*this by-reference as all assignment operators should.
* **empty()**: Test whether the set is empty. This method takes no parameters and returns a Boolean value.
* **size()**: Return the set size. This method takes no parameters and returns an integer value.
* **capacity()**: Return the current capacity of the set. That is, the number of elements the set is able to store without reallocating.
* **clear()**: Clear the contents. This method takes no parameters and returns nothing.
* **insert()**: Add an item to the set. This method has a void return type. One parameter will be expected, the item to be added to the set. Note that the order in the set must be preserved. This means that the correct location in the set must be found before insertion. Also, if the item already exists in the set, then nothing is done. Of course, if the capacity is insufficient to hold the newly inserted item, then it will need to double the buffer size just like Container.
* **erase()**: Remove an element from the set. This method takes an iterator as a parameter, referring to the element to be removed from the set. Note that the pseudocode in the reading takes an index so some modification to that pseudocode will be required. There is no return value.
* **find()**: Takes a template item as a parameter and returns an iterator pointing to the corresponding item in the set. If the item does not exist, it returns Set::end(). Note that the pseudocode in the reading returns an index so some modification to that pseudocode will be required.
* **operator&&()**: The intersection operator. The parameter is a Set object and the return value is a new Set object containing all the elements that are in both this and in the parameter.
* **operator||()**: The union operator. The parameter is a Set object and the return value is a new Set object containing all the elements that are in either this or in the parameter.
* **begin()**: Return an iterator to the first element in the set. It takes no parameters and returns a SetIterator.
* **end()**: Return an iterator referring to the past-the-end element in the set. The past-the-end element is the theoretical element that would follow the last element in the container. It does not point to any element so it must not be de-referenced.

Most of the work for this class already exists in the Container class from Week 01. It would be a good idea to start from there.

Iterator

Additionally, create an iterator class that will traverse the set. Call this class SetIterator. Note that this iterator will work much like the iterator from Week 01.

for (SetIterator <int> it = s.begin(); it != s.end(); it++) cout << \*it << endl;

This class will need to support the same iterator operators as Week 01.

Driver Program

A driver program (/home/cs235/week05/week05.cpp) is provided. It will pound-include your header file (set.h) and expect a template class Set as well as the iterator class SetIterator to be defined therein. This program will exercise your class, filling the set with user input and displaying the results.

Additionally, a makefile (/home/cs235/week05/makefile) is provided, looking for the class driver program (week05.cpp) and your header file (set.h) to be in the current directory. It will build an executable (a.out) and also create a TAR file for submission (week05.tar).

If your class is properly implemented, then testBed will pass through this driver program.

Go Fish

In addition to passing the four test functions for the Set class, you will also need to use the Set class to implement the game Go Fish. The game of Go Fish will work the following way:

1. You will need to use the Card class defined in /home/cs235/week05/card.h and/home/cs235/week05/card.cpp. This class will contain all the methods and operators needed to manipulate a Go Fish card.
2. Read a hand (a Set object) from a file. This will be the hand we will play the game with. An example hand is at /home/cs235/week05/hand.txt. When you load the hand from a file, use this absolute path instead of copying the file into your own directory.

Shark AngleFish Crab SeaHorse Dolphin Cod Cod Cod Crab

1. The game will consist of five turns. With each turn:
   * Prompt the user for a card (using the Card class' extraction operator)
   * Test to see if the user's card it is the hand.
   * If it is in the hand, remove it and display the message “You got a match!” If it is not found, display the message “Go Fish!”
2. After the five turns are finished, display the number of matches and the cards remaining in the hand. Note that there is a tab before “You got a match!” and “Go Fish!”

We will play 5 rounds of Go Fish. Guess the card in the hand

round 1: Shark

You got a match!

round 2: Cod<

You got a match!

round 3: Cod

Go Fish!

round 4: Cod

Go Fish!

round 5: Cod

Go Fish!

You have 2 matches!

The remaining cards: AngleFish, Crab, Dolphin, SeaHorse

To get full credit for this program, you must use your own Set class. If your class does not work, use the standard template library std::set from #include <set>. If you do this, you will loose points for the first half of the assignment, but not the second.

Common Mistakes

The most common mistakes students make with this assignment include the following:

* **Failing to allow the set to grow to handle any number of elements**. As with Week01, you will need to use a dynamically allocated array.
* **Using an iterator withing a method**. Remember that iterators are designed to make the client's life easier, so they do not have to be aware of implementation details. Methods within the Set class already are aware of implementation details because they have access to the member variables. You should therefore use an integer index to loop through the data in this case.
* **Failing to keep the set sorted**. If you choose to use the un-sorted implementation, then you will need to sort the set when an iterator is instantiated with Set :: begin(). This implementation is problematic and very expensive. It is much easier (and has better performance) to follow the sorted implementation.
* **Use a relative path for hand.txt.** Use the absolute path or testBed will not be able to read your file. An example of a relative path is "hand.txt" while an absolute path is "/home/cs235/week05/hand.txt".
* **Duplicate binary search.** Two functions in the Set class will need to do a binary search:find() and insert(). Do not have two copies of the binary search algorithm; have only one!
* **Biting off too much.** Start by implementing the game with a "hand" of integers usingstd::set as the container. Once this works, substitute int with Card from/home/cs235/week05/card.h. Once this works, then substitute the standard template library set (std::set) with the Set class you created at the beginning of this assignment. This technique is much easier than trying to do it all at once.

Test Bed

The testBed for this assignment is:

testBed cs235/week05 week05.tar

You can also run testBed on the executable:

testBed cs235/week05 a.out

Of course, you will need to pass testBed to get full credit on the assignment.

Submitting

You will submit this assignment using the Linux submit command. Please:

1. Create a TAR file built from the makefile, which will contain seven files:
   * makefile: Directly from /home/cs235/week05/makefile except with your edits on the comment block.
   * set.h: Your class definition for Set and SetIterator.
   * goFish.h: Containing the prototype for playGame() and any other function you may need.
   * goFish.cpp: Implementation for all the functions and classes necessary to play Go Fish.
   * card.h: Provided, containing the class definition for Card.
   * card.cpp: Provided, containing the method definitions for Card.
   * week05.cpp: Unmodified from /home/cs235/week05/week05.cpp.
2. Run the program by hand a few times through all four test cases as well as the game Go Fish.
3. Verify your solution with testBed.
4. Submit your file using the submit command. The submit command will prompt you for your instructor, the class (cs235), and the assignment (week05). You submit your file with:

submit week05.tar

Your program will be graded according to the following rubric:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Exceptional 100% | Good 90% | Acceptable 70% | Developing 50% | Missing 0% |
| Set Interface  20% | The interfaces are perfectly specified with respect to const, pass-by-reference, etc. | week05.cppcompiles without modification | All of the methods in Set and SetIterator match the problem definition | Both Set andSetIteratorhave many of the same interfaces as the problem definition | The public methods in theSet class do not resemble the problem definition |
| Set Implementation  20% | Passes all fourSet testBed tests | Passes three testBed tests | Passes two testBed tests | Passes one testBed test | Program fails to compile or does not pass any testBed tests |
| Go Fish  30% | The code is elegant and efficient | Passes the Go Fish testBed test | The code essentially works but with minor defects | Elements of the solution are present | The game Go Fish was not attempted |
| Code Quality  20% | There is no obvious room for improvement | All the principles of encapsulation and modularization are honored | One function is written in a "backwards" way or could be improved | Two or more functions appears "thrown together." | The code appears to be written without any obvious forethought |
| Style  10% | Great variable names, no errors, great comments | No obvious style errors | A few minor style errors: non-standard spacing, poor variable names, missing comments, etc. | Overly generic variable names, misleading comments, or other gross style errors | No knowledge of the BYU-I code style guidelines were demonstrated |

Please make sure to fill out the program header in the makefile. Failure to do this will result in a loss of 10% on the assignment.