Assignment 3: Sets and Stacks

Due: Friday February 16th, 23:59. Worth 3% of your course grade.

Web stuff: See the course web site for sample input and output, and a summary of any hints and announcements about the assignment. There is no starter code for this assignment.

Purpose: To give you experience designing an interface for an ADT and writing code with templates. You will also use a new data structure called a self-organizing list.

Stack: Your first task is to write a template for a stack class. You may define the interface as you see fit, as long it follows the traditional concept of a stack.

Your implementation should use a linked list of chunks, each of which stores several stack elements. The default size of these chunks is 10 elements, but include a constructor that lets the client choose a different chunk size. Whenever you push, if the top chunk of the stack is full, just construct another chunk and link it in.

You will probably find your stack template useful when you implement set.

Set: Next, write a template for a set class. It should include the following operations:

- Insert a new object into the set, if there is not already one that equals it. Assume the objects in your set have overloaded the “==” operator, and use this to test for equality.
- Search the set for an object that equals a given one. Again, use the “==” operator.
- Clear the set, that is, make it empty. Do not delete the memory used by the objects themselves; leave that to the client to do, if desired.
- Return the number of elements in the set.
- Given an object, delete from the set the one (if there is one) that equals it. Again, use the “==” operator.
- Undelete the last thing that was deleted, if there was one. This operation can be used repeatedly to undo deletions, in reverse order.
- Print the set. Overload the “<<” operator to do this. Assume that the objects in your set have themselves overloaded “<<”. Use this to print them, and print your own newline between each.

Again, you may define the interface as you see fit.
You must implement your set with a linked list of nodes, each of which stores a single set element. You may make the linked list singularly or doubly linked.

To be sure that everyone's output is exactly the same, I am imposing these requirements: Your insert function must add new set items at the end of the list, and your print function must print the list in order from the beginning to the end. Your undelete must put the item-to-be-undeleted at the end of the list.

Now here comes the “self-organizing” part. Search in a linked list requires visiting half the nodes on average, if the searches are uniformly distributed among the list elements. If however, the elements most often searched for are nearer the front of the list, the average behaviour improves. How can you make sure that the items most often searched for are near the front? One strategy is this: Every time you search for an element (and find it), move that element to the front of the list. This is a gamble; perhaps it will never be searched for again. But then it will slowly sink to the end of the list.

A list that uses this strategy is called self-organizing. You must use a self-organizing linked list to implement your set.

You will write a simple test driver, called Driver.cpp. It should create a set of ints, and then read and process commands. To make the input simpler, each token is on a separate line. Make sure that there are no prompts, and that the only output is that indicated below.

These are the commands:
- "i" followed on the next line by a number: Insert the number.
  Print “inserted” or “not inserted” (if the number was a duplicate of something in the set).
- "s": search for the number.
  Print “yes” or “no” to indicate whether it was in the set.
- “d” followed on the next line by a number: Delete the number.
  Print “deleted” or “not deleted” (if the number was not in the set).
- “u”: undelete the last thing that was deleted.
  Print “undeleted” or “nothing to undelete” (if there are no more items to undelete).
- “z”: print the size of the set
- “c”: clear the set. Print “cleared”.
- “p”: print the set, in the format described above for the set printing operation.

See the web for sample input and output.

Details on how to submit your assignment will be available on the web site. You will be required to submit a make file.