

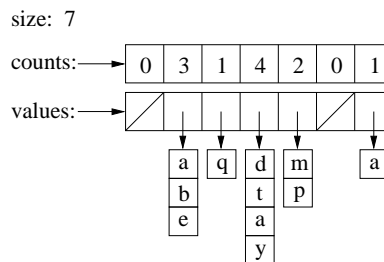
CSCI-1200 Computer Science II — Fall 2006

Homework 8 — Undo Array

In this assignment you will build a custom data structure named `UndoArray`. Building this data structure will give you practice with pointers, dynamic array allocation and deallocation, and writing templated classes. The implementation of this data structure will involve writing one new class. You are not allowed to use any of the STL container classes in your implementation or use additional classes or structs.

The Data Structure

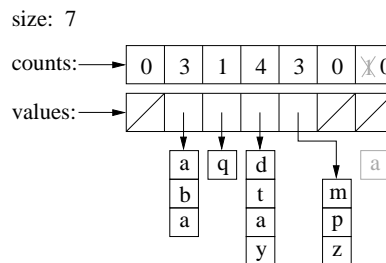
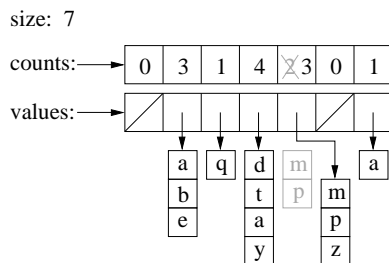
The `UndoArray` works like an ordinary fixed-size array that stores n values of template type `T`. Like ordinary C/C++ arrays, you can `get` and `set` individual entries in the array. What is unusual is that you can sequentially `undo` the calls to `set`. To make this work, we'll need to store the history of all values that were assigned to each index. Below is a diagram of the data structure you will implement. In this example `T` is type `char`:



The `UndoArray` class has 3 member variables: `size`, an unsigned integer representing the size of the `UndoArray`; `counts`, an array that stores the number of times each position in the `UndoArray` has been set; and `values`, an array of arrays that store the history of values for each position in the `UndoArray`. The current value of a position in the `UndoArray` is the last entry in the history. In the above example, a call to `get(1)` returns 'e'. The length of each history array is equal to the number of times that value has been `set` (minus the number of times it has been `undone`). If a position in the `UndoArray` is uninitialized, the corresponding position in the values array stores a NULL pointer (indicated with a slash). Attempting to read an uninitialized value in the `UndoArray` is an error. The `initialized` function can be used to verify that the value is initialized.

Modifying the Data Structure

When a position is `set`, the counts array is appropriately incremented, a new history array is allocated that is one longer than the previous history array, all values in the history are copied with the new value written at the end, the old history is deleted, and the corresponding values pointer is changed. For example, a call to `set(4, 'z')` will result in the new data structure diagram below left.



When a position is `undone`, the counts array is decremented, a new history array is allocated that is one shorter than the previous history array, the history values are copied (except for the most recent value), the old history is deleted, and the corresponding values pointer is changed. It is an error to undo an uninitialized value. If the history contains only one value, that position becomes uninitialized; for example, a call to `undo(6)` results in the data structure diagram above right.

Testing

We provide a main function that will exercise your data structure; however, the tests are not thorough. It is your responsibility to add test cases where the template class type `T` is something other than `char`, and test cases to ensure that your copy constructor, assignment operator, and destructor are working correctly.

You must also implement an output stream operator for your class which outputs an ascii representation of the private data structures stored by the class. Use the diagrams in this handout as a guide. This output operator will be helpful as you debug your class. Include examples of the use of this function in your new test cases.

Looking for Memory Leaks

The data structure for this assignment (intentionally) involves lots of allocation & deallocation. Certainly it is possible to design more practical and efficient alternate data structures that implement the same functionality. For this assignment, please implement the data structure *exactly* as described.

To verify that your data structure does not contain any memory leaks and that your destructor is correctly deleting everything, we include a batch test function that repeatedly allocates an `UndoArray`, performs many operations, and then deallocates the data structure. To run the batch test case, specify 2 command line arguments, a file name (`small.txt`, `medium.txt`, or `large.txt`) and the number of times to process that file. If you don't have any bugs or memory leaks, this code can be repeated indefinitely with no problems.

On Windows, open the Task Manager (Ctrl-Shift-Esc). Select "View" → "Select Columns" and check the box next to "Memory Usage". View the "Processes" tab. Now when your program is running, watch the value of "Mem Usage" for your program (it may help to sort the programs alphabetically by clicking on the "Image Name" column). If your program is leaking memory, that number will continuously increase (and eventually crash).

Alternately, on Cygwin/Linux/FreeBSD, open another shell and run the `top` command. While your program is running, watch the value of "RES" (resident memory) for your program. If your program is leaking memory, that number will continuously increase.

Extra Credit: Pointer Arithmetic

For extra credit, rewrite your `UndoArray` implementation to use pointers and pointer arithmetic for access to the internal data arrays *instead of* the subscripting operator, `[]`.

Submission

Do all of your work in a new folder named `hw8` inside of your CSII homeworks directory. Use good coding style when you design and implement your program. Be sure to make up new test cases and don't forget to comment your code! When you are finished please zip up your folder exactly as instructed for the previous assignments and submit it through the course webpage.