

CS 14: Data Structures and Algorithms

Lab 5

Peter H. Fröhlich
phf@cs.ucr.edu

Wagner Truppel
wagner@cs.ucr.edu

Out on: February 3, 2003

Due by: February 9, 2003

Remember to follow the instructions and policies for assignments given on the course website. This week's assignment consists of **three** separate parts:

- an **in lab** programming exercise (1)
- an **at home** programming exercise (2)
- an **at home** written exercise (3)

The “in lab” part is due at the **end** of your **lab section**. The “at home” parts are due **before 8:00 pm** on the date given above. You have to use the department's electronic turnin service for **everything**, including the **written** part. Put your solution for the latter into the **text file** `solution-5.txt` and turn it in together with the other files.

Important Notes: If you get warnings in the standard library when using `-Werror`, please don't worry about them; we are **not** asking you to fix the library, just your own code. Also, the test cases for lists are **not** very comprehensive, and you should add more to make sure **all** the list operations work!

1 In-Lab: Lists as Arrays

You can download an archive for this lab from the course website again. Among other things, you will find a simplified version of the familiar `array` file defining the interface for ADT `Array`. There's also a file `singlelist` with a fairly complete implementation of singly-linked lists in the form discussed in the lecture (i.e. nested classes, iterators, etc). Furthermore, there's a `README` file which contains some additional comments on the implementation, and a file `testsingle.cpp` with a number of **basic** test cases.

Your first task is to develop an implementation of the `Array` interface in terms of `SingleList`. That is, develop a class `cs14::ListArray` (of course in a file `listarray`) that implements the `Array` interface, but which performs **all** the `Array` operations in terms of a private `SingleList` instance. Use the file `testlistarray.cpp` to “sanity check” your implementation.

Hints: You will only need the operations `add_first` and `first` from `SingleList`, but you have to correctly use `SingleList::Iterator` to traverse the list. For iterators, remember that `valid` returns `true` as long as you are **within** the list, that `get` and `put` are used to change the element at the current position, and that `next` advances your position by one. Also, it can be helpful to write a private member function `moveTo` that just **positions** an iterator correctly...

2 At-Home: DoubleLists

The archive for this week's lab also contains the file `testdouble.cpp` with some test cases for a class `cs14::DoubleList` in a file `doublelist`.

Your task is to develop the implementation of a doubly-linked list following the example of singly-linked lists. Note that while some operations on singly-linked lists take $O(n)$ time, **all** operations on your doubly-linked list should take $O(1)$ time.

Hints: A simple way to get started is to copy `singlelist` to `doublelist` and try to compile `testdouble.cpp` (which won't work). Next make the minimal extensions and modifications to get the test program to compile. Only **then** start changing the internals to

make the operations work and more efficient. “Test early, test often!” should be your “mantra” for this.

3 At-Home: List Tradeoffs

For this exercise, you will need to refer to the books assigned for the course and maybe surf the web as well. Answer the following questions in **one or two sentences** each (the shorter the better):

1. What is a **circular** list implementation?
2. What is the major **advantage** of circular lists?
3. What is a **sentinel** (or **dummy**) element in connection with lists?
4. What is the major **advantage** of using **sentinel** elements?
5. Does it make sense to **combine** circular lists and sentinel elements? Why or why not?

If you use sources **other** than the assigned books, be sure to include a reference!

Hint: You should start working on this early, just in case you need to ask us for clarifications.