Assignment 2

Due: October 11th, 2015 before 11:59pm

Objectives

- Introduction to a list ADT in C++.
- Practice reading and implementing the specification of an ADT.
- Practice with both array-based and linked-list-based data structures.
- Introduction to simple sorting algorithms and algorithm efficiency.

Introduction

The goal of this assignment is to implement two different versions of a list data structure, which stores an ordered sequence of elements. In this assignment, the elements of the list will always be integers. Later in the course, we will see how to implement data structures with the flexibility to use any element type (using C++ templates).

Part I of this assignment is to implement an array-based list structure, with the ability to add an arbitrary number of elements to the list. In Part II, you will implement a linked-list-based structure, which has the same set of behaviors as the array-based version. Although the implementations of the two data structures will be internally very different (and may have different performance characteristics), both data structures will support the same set of operations.

Part III of the assignment involves rewriting a sorting algorithm to sort the elements of a linked list more efficiently

You are not permitted to change anything inside the `public` **section of** `array_list.h` **and** `linked_list.h`. **If your submissions for this assignment contain modifications to the public definitions in** `array_list.h` **or** `linked_list.h`**, you will receive a mark of zero. You are permitted to add new definitions to the** `private` **section of either file.**

Quick Start:

1. Read this entire document
2. Download all of the A2 source files from conneX. For convenience, a zip file containing all of the assignment source files has been posted.
3. Complete Part I:
   a) Read the entire `array_list.h` file carefully.
   b) Compile and run the array list tester with the following commands:
      Compile: `g++ -Wall array_list.cpp array_list_tester.cpp -o array_list_tester`
      Run: `./array_list_tester`
   c) If all tests pass, move on to Part II.
   d) If a test fails, fix the implementation of the tested function and go back to (b).
4. Complete Part II:
   a) Read the entire `linked_list.h` file carefully.
   b) Compile and run the linked list tester with the following commands:
      Compile: `g++ -Wall linked_list.cpp linked_list_tester.cpp -o linked_list_tester`
      Run: `./linked_list_tester`
   c) If all tests pass until the tester displays `Part II Complete`, proceed to Part III. You may need to use Ctrl-C to terminate the tester program if it enters the Part III test.
   d) If a test fails, fix the implementation of the tested function and go back to (b).
5. Complete Part III:
   a) Rewrite the `bubble_sort` function in `linked_list.cpp` such that the sorting test finishes in less than 30 seconds.
   b) Compile and run the linked list tester as in Part II.
   c) If the sorting test does not finish in 30 seconds, use Ctrl-C to terminate the program and go back to (a).

Part I

Part I requires implementing the methods of the array_list class in array_list.cpp. The definition of the class is located in array_list.h, and the specifications for each method can be found in a comment before the method declaration in array_list.h. The pre-conditions for each method have been designed such that your code does not need to perform any error-checking on input values.

The supplied testing program array_list_tester.cpp tests each of the methods in the array_list class. After all methods have been successfully implemented, the output of the tester program will be:

```
test_constructor passed.
{1}
{2,1,3,4}
{2,6,1,3,4}
{5,2,6,1,3,4}
test_insert_and_output_and_get passed.
test_in_list_and_expand passed.
test_remove_and_empty passed.
{101,102,103,104,105,106,107}
test_remove_value passed.
test_clear passed.
stress_test passed.
Attempting to sort the list {679,-3640,-917,1336,-3315,-4884,-4181,1560,4049,-4500}
Sort successful.
Attempting to sort a list of 10000 elements
Sort successful. Total time: 0.351999 seconds
sort_test passed.
Passed: 8
Part I complete.
```

Note that the total time of the sorting test may vary between machines.

Compiling and running Part I:

To compile Part 1, type:

```
% g++ -Wall array_list.cpp array_list_tester.cpp -o array_list_tester
```

To run Part 1 type:

```
% array_list_tester
```

Part II

Part II requires implementing the same ADT as Part I using a linked list instead of an array. Specifically, you will implement the methods of the linked_list class in linked_list.cpp. The definition of the class is located in linked_list.h, and the specifications for each method can be found in a comment before the method declaration in linked_list.h, similar to Part I.

The supplied testing program linked_list_tester.cpp tests each of the methods in the linked_list class. The linked_list_tester.cpp program is designed to test both the Part II and Part III methods. Once the tester displays “”, Part II is complete and you can start on Part III. Note that you can terminate a running program with Ctrl-C (on Windows and Linux) or ⌘-C (on Mac). For example, after completing Part II, the tester should display the following:

```
test_constructor passed. test_constructor passed.
{1}
{2,1,3,4}
{2,6,1,3,4}
{5,2,6,1,3,4}
test_insert_and_output_and_get passed.
test_in_list_and_size passed.
test_remove_and_empty passed.
{101,102,103,104,105,106,107}
test_remove_value passed.
test_clear passed.
stress_test passed.

Part II complete
Starting Part III tests
...
```

Part III

When you have finished Part II, your linked_list implementation will support all of the methods of the list ADT. In the original template, the bubble_sort method has already been implemented. However, since it uses repeated calls to get_node, the implementation is very slow, and the sorting test in the linked_list_tester program may require more than an hour with the provided implementation.

Your task for Part III is to rewrite the bubble_sort method to avoid the use of get_node and pass the sorting test in the linked_list_tester program in less than 30 seconds. Note that a correct implementation will normally take less than one second.

Submission

Submit your array_list.cpp, and linked_list.cpp files using conneX.

As usual, it is acceptable (and encouraged) for you to talk about your assignment with your classmates, and you are encouraged to design solutions together, but each student must implement their own solution. Plagiarism detection software will be run on all assignment submissions.

Grading

If you submit something that does not compile, you will receive a grade of 0 for the assignment. It is your responsibility to make sure you submit the correct files.

Requirement	Marks
Part I:	
array_list.cpp compiles with no warnings	1
Each of the 8 tests in array_list_tester passes (one mark per successful test)	8
Part II:	
linked_list.cpp compiles with no warnings	1
Each of the 7 Part II tests in linked_list_tester passes.	7
Part III:	
The sorting test in linked_list_tester passes, and the implementation of the bubble_sort method uses the $O(N^2)$ bubble sort algorithm.	2
Total	19