Lab 5 – C++ and Basic Sort
CPSC 1021- Fall 2017

Introduction

In today's lab, we will read in a file using C++ and then perform a basic Bubble Sort.

Lab Objectives

- C++ File I/O
- Reading a file with no length marker
- Write a bubble sort routine

Prior to Lab

Make sure you can read in and write out to files in C++. This involves using C++ style streams. Read more here: http://www.cplusplus.com/doc/tutorial/files/

Lab Instructions

Create two files, **lab5.cpp** and **lab5.h** (NOTE: **.hpp** is also a valid header extension, and could be useful for distinguishing C headers and CPP headers).

You will be given an input file of variable length that contains integers. Your program should take **2 command line arguments**, the input file of integers and the name of an output file where the sorted list of integers will be written.

You may only use C++ style file I/O to manipulate data. **You may not use printf or scanf in your program**. Additionally, if you use the C++ sort() function, you will receive a zero ☺

Your Objectives

- Read through the input file and determine how many integers you need to store
- Read through the file a second storing the integers in an array
- Pass this array to a bubble sort function, sorting the integers in **ascending order**
- Write the sorted integers to the output file, **one integer per line**

For example, given the input file **number.txt**:
3
6
2

Your program should produce an output file:
2
3
6

Bubble Sort

Bubble sort is a simplistic **sorting** algorithm which allows us to rearrange the elements of our array into sorted order in **$O(n^2)$ time**. This **big-oh notation** tells us the **upper bound time complexity of our algorithm.** All that really means is, given some problem which has a size **n**, the time it takes to solve the problem is no worse than order of **n^2.**

For example, if we had 10 numbers to sort, the time complexity of bubble sort tells us that it will require somewhere in the ball-park of 100 operations to completely sort them. This description is simplistic and incomplete, but it does capture the main idea: higher time complexity means higher worst case time to solve a problem. Notice how big n^2 becomes as n grows large ($n = 1{,}000 \rightarrow n^2 = 1{,}000{,}000$). The best comparison based sorting algorithm **quicksort** has time complexity O(n log(n)), which is much fast as n grows large. We'll implement quicksort later this semester!

For **bubble sort**, the idea is we scan through the array and allow large values to "bubble up" to the end of the list.

See: https://en.wikipedia.org/wiki/Bubble_sort

The basic pseudo-code for this algorithm is as follows:

```
bubble_sort( array, length ) {
 while ( number_swaps != 0 ) {
   number_swaps = 0
   for ( index, index < length - 1, index++ ) {
     if array[index] > array[index+1] → swap; number_swaps++
   }
 } }
```

We keep looping through the array until we do not perform any swaps. At the end of the algorithm, the numbers are sorted from left to right, smallest to largest.

Write a function that implements this algorithm. It should not have any hard coded values!

Header Files and Header Guard

Your bubble sort function should have its prototype defined in the lab5.h file. At this point, we should be following best practice for maintaining our source code, and using **header guards**.

https://en.wikipedia.org/wiki/Include_guard

This protects us from multiple definition errors when compiling.

What to turn in

- **lab5.cpp** which reads in a file of integers to an array, sorts them, and writes the sorted list out
- **lab5.h** which has the prototype for your bubble sort function

SOME HINTS!
There is a swap() function in the C++ standard library that can make your code a little cleaner. Google it for more info.

To reset a file and read from it a second time, use the clear and seekg functions:

in_file.clear()
in_file.seekg(0, std::ios::beg)

Compile and Execute

Use G++ to compile your code as follows:

gpp -Wall -g lab5.cpp -o bubble

Execute the program

./bubble <input> <output>

FORMATTING:

1. Your program should be well documented
2. Each file should include a header:
3. Your program should consist of proper and consistent indention
4. No lines of code should be more than 80 characters

```
// Sample Header
/*******************
 your name
 username
 Lab 1
 Lab Section:
 Name of TA
*******************/
```

5 – 10 points will be deducted for each of the above formatting infractions.

Submission Instructions

- Test your program on the School of Computing server prior to submitting.
- Use the tar utility to tar.gz all source files. **Do not tar an entire directory! When I untar your archive, I should see all the files you included, not a top-level directory! Failure to correctly tar may result in up to a 25-point penalty!**

  EX. tar –czvf nglyder-lab5.tar.gz lab5.cpp lab5.h

- Name your tarred file **<username>-lab<#>.tar.gz** (ex. nglyder-lab5.tar.gz)
- Use handin (http://handin.cs.clemson.edu) to submit your archive