COMP 215 Algorithms

Lab #4

In this lab, we will review (well, you will anyway, I'm not helping you in this description) file input and output in C++. You will also be doing funny stuff with large amounts of data you read from the file. You only have to hand in the .cpp file containing your code (and the answers I ask in comments at the beginning of the file). Do NOT include the file with the random numbers. It is too big. I don't want to see it. Hand in everything before Friday, September 25, 11:59pm.

The instructions for this lab are given at a slightly higher level than before, I now expect you to be able to figure some stuff out on your own. Remember that if you have trouble with C++ syntax, or you do not remember the exact way to use standard functions, http://www.cplusplus.com/ and Google are very useful resources.

First we write a large file full of random numbers. The file must contain 10,000,000 (ten million) lines, each line contains three random integers.

1. Write a function `generateFile` that takes no input, returns no output, but writes the file with 10,000,000 lines, each containing three random integers. The file should be called `InputNumbers.dat`. You can use any method you like for writing to the file (old style `fprintf`, or file output streams), but you should use the `rand()` function to generate the 30,000,000 random numbers you will need for the file. Write three random number per line, and 10,000,000 lines. (you can reduce that number to 1,000,000 if ten million takes too long – it could take a couple minutes)

Why did we write three numbers per line? Because we will be using them in triples of course! For this, we will need a data structures that stores triples of numbers.

2. Write a class `integerTriples` that contains three public integers: `value1`, `value2` and `value3`.

3. Write a constructor for this class that takes three integers, and gives the these integers to `value1, value2` and `value3` in the obvious way.

4. Overload the operators ==, >, <, <= and >= so that we have that if (a, b, c) and (d, e, f) are triples, then $(a, b, c) == (d, e, f)$ if and only if $a == d$, $b == e$ and $c == f$, and $(a, b, c) < (d, e, f)$ if and only if $a < d$ OR if $a = d$ and $b < e$ OR if $a = d$, $b = e$ and $c < f$. The other operators should implement the natural extensions of these two. All these functions should be public.

Now, we need a function that will read the big file we created before, and store it in an array of `integerTriples`.

5. Write a function `readFile` that takes an integer `nbLines` as input, allocates an array of `integerTriples` of size `nbLines`, reads the first `nbLines` from `InputNumbers.dat`, puts them in the array and returns a pointer to that array. You can assume that `nbLines` will always be less than or equal to 10,000,000.

Great, now that we have an array, we can start doing stuff with it.

6. Write a function `findMax` which, given an array (in the form of a pointer) of `integerTriples` and its size `n`, returns the largest `integerTriples` together with the index of the largest `integerTriples`. Since the function returns two outputs, it should do so by passing some of its input by reference.
7. Write a function `print` which, given an array (in the form of a pointer) of `integerTriples` and its size `n`, prints the content of the array on screen, one triple per line. This function should not return anything.
8. Write a function `ourSort` which, given an array (in the form of a pointer) of `integerTriples` and its size `n` sorts the values of the arrays as follows:
   (a) First find the maximum element among all `n` elements of the array and its index `i`. Then, swap the $\texttt{i}^{th}$ and $\texttt{n}^{th}$ element of the array, so that the maximum element of the array is now at the end.
   (b) Then, find the maximum element among the first `n-1` remaining elements of the array and its index `i`. Then, swap the $\texttt{i}^{th}$ and $\texttt{(n-1)}^{st}$ element of the array, so that the two maximum element of the array are now at the end.
   (c) Then, find the maximum element among the first `n-2` remaining elements of the array and its index `i`. Then, swap the $\texttt{i}^{th}$ and $\texttt{(n-2)}^{nd}$ element of the array, so that the three maximum element of the array are now at the end.
   (d) etc, all the way down to element 1.
9. Remember the function `clock()` that we used in a previous lab to determine the time taken to execute some code? We will use it again to determine how long it takes to sort arrays of various sizes. Write a function `ourSortTiming` which takes an integer `size` and does the following:
   (a) Use the function `readFile` to get a pointer to an array of `size` integerTriples.
   (b) Starts the clock.
   (c) Uses the function `ourSort` to sort the element of the array in increasing order.
   (d) Stops the clock and prints on screen the number of clock ticks taken to sort the array.
   (e) Delete the array of integerTriples.
10. Run the function `ourSortTiming` on arrays of size 100,000, 200,000, 300,000, etc. In rough terms, how much longer does it take when the size of the array doubles? triples? quadruples? Write your answer in comments at the beginning of the file. If you cannot tell the difference easily, try sizes 1,000,000, 2,000,000, 3,000,000 etc.