CPSC 2120/2121
Lab 10
List Permutation
Due date/time are on the CPSC 2121 Canvas website

Learning Objectives

- To improve our ability to implement algorithms and data structures in C++
- To implement a random algorithm in C++
- To use simple data structures to implement algorithms
- To become proficient in fundamental data structures used throughout computer science
- To improve our analytical skills while gaining familiarity with mathematical tools used in the analysis of algorithms
- To implement an algorithm and understand its intricacies

Problem/Exercise

For this lab, you will find three distinct ways to randomly permute the elements in a list. Your input will be a list of elements with the same data type, and your output should be a random permutation of that list.

At least one of your approaches should make use of a data structure as a fundamental part of the process, and all approaches should make some use of randomness. Testing will be accomplished by running the program multiple times on the same input. Points will be awarded based on how few times the same permutation comes up. Completely deterministic algorithms (e.g. shifting elements to right/left) will be given very little credit.

You are allowed to use any data structures in the C++ STL for this assignment. Your final output will be in a single file, named permute.h, which will compile without errors using the provided Makefile. In order to thoroughly test your program, you will also submit a single input file named lab10.txt which has map information and queries in the same format as the provided lab10.txt.

Additional Requirements

- Your permute.h file must compile and run with unmodified versions of the provided lab10.txt and Makefile.

Examples

Your C++ source code should be in a file called permute.h, and it should be compiled into an executable called lab10.out using our provided Makefile.

./lab10.out

First:	4 2 5 1 3
Second:	2 4 3 5 1
Third:	5 2 3 4 1

./lab10.out

First:	4 1 3 2 5
Second:	3 1 4 2 5
Third:	4 2 3 5 1

./lab10.out

First:	3 5 2 4 1
Second:	3 2 4 5 1
Third:	1 4 5 2 3

./lab10.out

First:	5 2 3 1 4
Second:	1 2 4 3 5
Third:	4 3 2 1 5

Source Code Requirements

- Put a comment at the top of your source code file with your name (first and last), the date of your submission, your lab section, and the assignment's name.
- All functions should be commented. Use inline documentation as needed to explain ambiguous, tricky, or important parts of your code.
- All source code must follow good programming style standards such as properly indenting source code; and all variables, functions, and classes should be well-named.
- Your class must use dynamic memory allocation and deallocation properly, which means your class cannot contain a memory leak. You may use valgrind to check for memory leaks. Refer to the Unix manual and valgrind's online documentation for more information on valgrind.

- Your program will read a list of space-separated elements from a file, and must output a permutation to stdout.

Submission

Before the date/time stated on the CPSC 2121 Canvas webpage, you need to submit your code to handin under the correct lab assignment. Make sure to submit all of the following.
1. All source files required for this lab (permute.h)
2. Your testing file (lab10.txt)

After you submit, always double check that the file(s) you submitted were the correct version. To double check, download the submitted file(s), put them on one of our Unix machines, and make sure they compile and run correctly.

Grading

If your class does not compile on our Unix machines or your assignment was not submitted on time, you will receive a grade of 0 on this assignment. Otherwise, your class will be graded using the criteria below.

Your class works correctly with our testing program(s)	6 points
Proper variable names, documentation, and code organization	2 points
Your class uses dynamic memory allocation/deallocation properly (no memory leaks)	1 point
Alternate testing file containing map and query data you created for testing which is distinct from that provided.	1 point

You must test, test, and retest your code to make sure it compiles and runs correctly and efficiently on our Unix machines with any given set of valid inputs. This means that you need to create many examples on your own (that are different from the provided lab10.txt) to ensure you have a correctly working class. We will only test your program with valid sets of inputs.