COP 4530: Data Structures
Project 2

You are not allowed to use the Internet. You may only consult approved references*.
This is an individual project.
This policy is strictly enforced.

You must submit a **hard copy** of all of the items requested below. You must also submit your *code*† to Canvas.

For full credit, the code that is submitted must:

- Use the specified signature, if applicable.
- Be implemented in a file using the specified file name, if applicable.
- Be correct (i.e., it must always return the correct result).
- Be efficient (i.e., it must use the minimum amount of time and the minimum amount of space necessary to be a correct implementation).
- Be readable and easy to understand. You should include comments to explain when needed, but you should not include excessive comments that makes the code difficult to read.
  - Every class definition should have an accompanying comment that describes what is for and how it should be used.
  - Every function should have declarative comments which describe the purpose, preconditions, and post conditions for the function.
  - In your implementation, you should have comments in tricky, non-obvious, interesting, or important parts of your code.
  - Pay attention to punctuation, spelling, and grammar.
- Follows ALL coding guidelines from section 1.3 of the textbook. Additional coding guidelines:
  - No magic numbers. Use constants in place of hard-coded numbers.
  - No line of the text of your source code file may have more than 80 characters (including whitespace).
  - All header files should have #define guards to prevent multiple file inclusion. The form of the symbol name should be <FILENAME>_H_
  - Do not copy and paste code. If you need to reuse a section of code, then write a function that performs that code.
  - Define functions inline only when they are small, say, 10 lines or less
  - Function names, variable names, and filenames must be description. Avoid abbreviation.
  - Use only spaces (no tabs), and indent 3 spaces at a time.
- Compile and run on the C4 Linux Lab machines (g++ compiler, version 4.8.2). *The shell script and makefile that I will use to compile and run your code will be posted on Canvas. Please note that I may use my own* `main.cpp` *file to test the code you submit.*
- Have no memory leaks.

*The list of approved references is posted on Canvas. You must cite all references used.
†Your code must compile and run on the C4 Linux Lab machines

Project Description

Develop a recursive function to generate all $n!$ permutations of a set of n elements.

Hint: The permutations of $\{1, 2, \ldots, k\}$ can be obtained by considering each permutation of $\{1, 2, \ldots, k-1\}$ as an ordered list and inserting k into each of the k possible positions in this list, including at the front and the rear. For example, the permutations of $\{1, 2\}$ are $(1, 2)$ and $(2, 1)$. Inserting 3 into each of the three possible positions of the first permutation yields the permutations $(3, 1, 2)$, $(1, 3, 2)$, and $(1, 2, 3)$. Inserting 3 into each of the three possible positions of the second permutation yields the permutations $(3, 2, 1)$, $(2, 3, 1)$, and $(2, 1, 3)$

Project Tasks

1. Create a file named `permutations.h` with the following functions to generate and output all permutations of an array:
   (a) [5 points] A generic function to output an array.
   `template <class T> void outputArray(T* items, const int& size, ostream& out)`
   An array is output on a single line with one space between each element.
   You may assume that the type T has overloaded the << operator.
   (b) [5 points] A recursive function compute $n!$.
   `long factorial(const int& n)`
   (c) [25 points] A generic recursive solution to output all permutations of an array.
   `template <class T> void outputPermutations(T* items, const int& size, ostream& out).`
   - There are $n!$ permutations of an array - you should use your factorial function from above to assert that all $n!$ permutations have been created.
   - Use your output array function from above to output each permutation.

   For example, the output for all permutations of the array [0, 1, 2] would be output as:

```
2 1 0
1 2 0
1 0 2
2 0 1
0 2 1
0 1 2
```

   You should implement additional helper functions as needed. Helper functions must also be located in the ***`permutations.h`*** *file.*

2. [5 points] Write a main function to test each of the functions from Question 1 above.
   Note that I will be creating my own ***`main.cpp`*** *file to test your code from Question 1 above.*