Project Goal:

The goal of this project is to design and implement a program that generates all permutations of a set of n positive integers, where the user provides a number, n, within the range of 1 to 25. The program will use the Johnson-Trotter algorithm to generate permutations and display them to the user.

Struct Name: **Data** - Creates objects to indicate value and direction

- Private members: int data, bool direction
  - By default, direction is true. true = ← , false = →

Functions:

- Data(int) - Constructor
  - Using the int value passed in, creates an object that contains that value.

Functions:

- float Total(float size) - Takes in float size, which is the # of objects to create from 1 → n, where n is the size. Using that, the function calculates n!, which is the total # of permutations.
- void get_Set(ostream&, vector<Data>) - Prints the passed permutation (all the values that the objects in vector<Data> contains).
- int FindLargestMobile(vector<Data>&) - Parses through the vector<Data> to find the index of the largest mobile element and then returns it. Returns -1 if no largest mobile element is found.
- void Johnson_Trotter(vector<Data>&, vector<vector<Data>>&) - Using the index of the largest mobile element from FindLargestMobile, swaps it to where it is pointing to. Reverses the direction (make false) of all other **Data** objects of vector<Data> that contain a value larger than which the largest mobile element contained. Stores the permutation into vector<vector<Data>>.

Main File:

Creates objects of type **Data** containing numbers from 1 → n. These objects are then stored into a vector that holds objects of type **Data**. Then all permutations of these objects containing these numbers are found using Johnson-Trotter. Each permutation is outputted to the console.

Test Driver:

Takes a text file with various integers, finds every permutation and stores them into a vector list. Outputs the expected amount of permutations and the total number of permutations stored in the vector list.

Test Cases:

- Test if get correct output when n = 25 → Should get 1.55112e+25 Permutations
- Test if get correct output when n = 1 → Should get 1 Permutation
- Test if get correct output when n = 5 → Should get 120 Permutations
- Test if get correct output when n = 3 → Should get 6 Permutations
- Test if get correct output when n = 10 → Should get 3628800 Permutations
- Test if get correct output when n = - 1 → Should be invalid as n < 0
- Test if get correct output when n = 30 → Should be invalid as n > 25
- Test if get correct output when n = T → Should be invalid a T is not an integer