SuperVector Project

In this project, you will extend and build on the IntVector data type that we built in class. You should refer to that code during this project (either the paper handout or the code from the class website).

You will create a SuperVector data type that has all the same capabilities as an IntVector but more as well:

- SuperVectors are a templated class, which means they can store any data type, not just integers.
- Individual items can be appended, prepended, or inserted anywhere into a SuperVector.
- One SuperVector may be appended, prepended, or inserted anywhere into a second SuperVector.
- An individual item, or an entire range of items, may be removed from a SuperVector.
- A SuperVector may be "sliced" much like a Python list, returning a sub-portion of the original SuperVector as a new SuperVector.

What you need to do:

- Modify the SuperVector.h file to fill in all the functions that are blank right now.

  Note: Templated classes work slightly differently than "regular" classes, which results in a few issues:

  - Every class and function must be preceded by template<class T>. This is entirely taken care of for you in the SuperVector.h file, however.

  - For technical reasons surrounding templates, it's more complicated than normal to split a templated class into a .h and a .cpp file, so traditionally the whole class (both the class declaration and the method bodies) are kept in the .h file. So do not create a SuperVector.cpp file for this project.

- Modify main.cpp which is a "driver" program that demonstrates the functionality of the SuperVector class.

  The driver program maintains two SuperVectors, called "a" and "b," and allows the user to manipulate them in various ways by reading commands from a text file. The commands your driver program must handle are: (in the descriptions below, "letter" will either be "a" or "b")

  - append `letter value`: This command, e.g., `append a 10`, appends an integer to the end of either vector a or b. So append a 10 should call a.append(10).

- `append` *`letter letter`*: This command, e.g., `append a b`, appends the second vector to the first. So `append a b` should call a.append(b).

- `prepend` *`letter position`* or `prepend` *`letter letter`*: Similar to `append`.

- `insert` *`letter position value`*: A command such as `insert a 4 6` should call a.insert(4, 6).

- `insert` *`letter position letter`*: A command such as `insert a 4 b` should call a.insert(4, b).

- `clear` *`letter`*: Clears either vector a or b (calls the clear method).

- `remove` *`letter position`*: A command such as remove a 3 should call a.remove(3).

- `remove` *`letter start end`*. Removes all positions from the given vector from start, start+1, start+2, ...end-1. Position end is not removed. So `remove a 3 5` would remove positions 3 and 4, but not 5.

- `slice` *`letter start end letter`*. Slices the first SuperVector from start to end and puts the result in the 2nd SuperVector. So the command `slice a 2 4 b` should do:
  b = a.slice(2, 4).

Mostly all you have to add to main.cpp is more if/else statements to handle the other commands. Append is already done for you; you can use it as a guide.

Guidelines

- All operations should be as efficient as reasonably possible, especially in terms of big-O. Watch out for situations where you make extra loops over the SuperVector that you don't need.

- The only place you should be using regular C++ vectors (from #include <vector>) is in the split() function in main. Otherwise, everything should be done with SuperVectors.

- For adding single elements at a time to a SuperVector, follow the same rules as we did for IntVector (expanding by units of SIZE_INCREMENT).

- For adding multiple elements at a time (inserting/appending/prepending another SuperVector), expand (if needed) just enough to fit the new items in, no more, no less.

- When removing elements from a SuperVector, do not reduce the capacity.

Grading

- Your output should match mine.