Vector Class

Background Information

Static arrays in C++ pose a very common problem—the need to know exactly how big the array should be at compile time. Frequently, we do not know this information because the values come from a variable input source, such as a file of unknown size. Dynamic arrays solve this problem by allowing the number of values stored to be determined at run time.

A vector serves as a wrapper for a dynamic array—a wrapper in that it surrounds the array with a class, creating an interface that provides certain functionality, such as accessing, removing, or counting the underlying elements. A further benefit is that vectors handle the dynamic memory allocation and resizing of arrays as necessary under the hood.

In this assignment, you will be creating your very own vector class. Please note that this is OPTIONAL. See **Grading** for more information.

Base Requirements

You should be able to fulfill these requirements using what you learned in CSCI 135. If you would like more of a challenge, you may continue on to the **Intermediate Level Requirements**. Your **IntVector** class will only be able to store integers. All instances initially start with a dynamically created array of size 10. Make sure to deallocate your data in an appropriate location. The class interface should provide the following functionality:

1. `void push_back(int data)` — Adds a new element, **data**, at the end of the vector, after its current last element. This effectively increases the container size by one. If the array is full, a new array of double the size will be created and all of the elements will be copied over.
2. `int pop_back()` — Removes and returns the last element in the vector, effectively reducing the container size by one. If the vector is empty, write **throw -1;** in the code.
3. `int at(int pos)` — Returns the element at position **pos** from the array. If **pos** is not a valid position, write **throw -1;** in the code.
4. `int size()` — Returns the current number of elements in the vector.
5. `int capacity()` — Returns the size of the storage space currently allocated for the vector
6. `bool empty()` — Returns whether the vector is empty (contains 0 elements)

Intermediate Level Requirements

These requirements may require a bit more thinking. Completing them means you're in good shape for this course. For even more of a challenge, continue on to the **Advanced Level Requirements**.

1. `void insert(int data, int pos)` — Adds a new element, **data**, at position **pos**.
2. `int remove(int pos)` — Removes and returns the element in position **pos**. If pos is not a valid position, write **throw -1;** in the code.
3. `int count(int data)` — Returns the number of times **data** exists in the vector.

Advanced Level Requirements

Keep in mind that the following requirements are outside of the scope of what you learned in CSCI 135; you are not expected to know how to do them as you begin this assignment. They will require you to do some extra research on your own, related to operator overloading and templates. Be sure to look at these topics in the ZyBook, as well.

1. Overload the [] operator for your vector class, which retrieves or overwrites (depending on whether it's **vec[4] = 100** or **int x = vec[4]**) elements at the specified position. The functionality is similar to element access for arrays. If it is not a valid position, write **throw -1;** in the code.
2. Overload the == operator, which determines if one vector is exactly the same as another—in other words, the vectors have elements with the same value in the same order.
3. Overload the = operator, which copies all the data in one vector to the other.
4. ***SPECIAL REQUIREMENT*** Instead of only allowing integer storage, make your vector class "general," so that a user can create a vector of any data type (e.g., strings, chars, doubles). Note that this is probably the most difficult requirement. It will require the most substantial changes. You will benefit from making a second class altogether, **TemplatedVector**. I've provided a second main file main function for you should you choose to attempt this task.

Things to Think About

- Some requirements are unsaid, but are still necessary for a properly written class (e.g., including a constructor).
- The provided main files run tests to ensure valid functionality. Use them to your advantage as you write code. Do not attempt to do everything at once. Code small pieces and test iteratively.
- What data members are needed for a vector to maintain information about its current state?
- Why is inserting/removing elements from the front/middle of a vector fundamentally different from inserting/removing from the end?
- Templating is often difficult to master because of the precise syntax required and the cryptic errors that appear at compile time. However, all this aside, what additional changes might you have to make to your function signatures (function return type + function name + function parameters) that may not have been necessary for **IntVector**?

Grading

This is an optional, ungraded assignment. However, you should use this project to gauge where you stand going into this course, especially in the context of what you are required to already know. I encourage you to challenge yourself and do more than you think you can do. This gives you the opportunity to make mistakes and learn from them without fear of being penalized. I will provide extended feedback on your code, so you can do better next time on the graded assignments.

Due Date

Even though this assignment is optional, the due date is Sunday, February 13th, at 11:59 pm. A sample solution will be provided after the due date. See the provided **GitHub Submission Guide** on Piazza for information about how to submit your assignment.