Integer Set

Background Information

Static arrays in C++ pose a very common problem—they need to know exactly how much data you need to store at compile time. Frequently, we do not know this information ahead of time because the values come from a variable input source, such as a file of unknown size or keyboard user input. Dynamic arrays solve this problem by allowing the number of stored values to be determined at run time. Despite this benefit, dynamic arrays can be cumbersome to deal with directly.

A vector serves as a wrapper for a dynamic array—a wrapper in that it surrounds the array with a class and creates an interface that provides well-defined functionality, such as accessing, removing, or counting the array elements. A further benefit is that the vector handles the dynamic memory allocation and resizing of its underlying array as necessary under the hood, without the user of the vector needing to know anything about it. Abstraction, for the win!

In this assignment, you will be creating your very own integer vector class and use it to implement an Integer **Set**. At any given moment, all of the elements must be unique. That is to say, the Set may not contain any duplicates. This is the same as the mathematical concept of a set**.**

Vector Requirements

You must implement a class called **Vector** that uses a dynamic array as its underlying data structure. You may not use the STL vector as the underlying data structure (duh!). All instances initially start with a dynamically created array of size three. Make sure to deallocate your array when necessary. The Vector class interface must provide the following functionality:

1. `unsigned int capacity() const` — Returns the size of the space currently allocated for the vector.
2. `unsigned int size() const` — Returns the current number of elements in the vector.
3. `bool empty() const` — If the vector contains 0 elements, return **true**; otherwise, return **false**.
4. `void push_back(const int& data)` — Add **data** to the end of the vector, doubling the size of the underlying dynamic array if necessary
5. `bool remove(const int& data)` — If the vector contains **data**, removes the *first* instance of **data** from the vector, and returns **true**; otherwise, returns **false.** You will need to shift elements if the element being removed is not the last element in the vector.
6. `void clear()` — Empties the vector of its elements and resets the capacity to 3.
7. `Overload operator=` — Copy Assignment Operator
8. `Overload operator[]` — returns element in pos requested. You may assume that the given position is valid
9. `bool at(unsigned int pos, int& data) const` — If the position **pos** is valid set the value of **data** to the element in the vector at position **pos.** Return true on success, false on failure. You may NOT assume the position is valid

Set Requirements

You must implement a class called **Set** that uses your vector underlying data structure. The **Set** class interface must provide the following functionality:

10. `unsigned int size() const` — Returns the current number of elements in the set.
11. `bool empty() const` — If the set contains zero elements, returns **true**; otherwise, returns **false**.
12. `bool contains(const int& data) const` — If the set contains **data**, returns **true**; otherwise, returns **false**.
13. `bool insert(const int& data)` — If the set does not already contain **data**, adds a new element, **data**, and returns **true**; otherwise, returns **false**.
14. `bool remove(const int& data)` — If the set contains **data**, removes **data** from the set, and returns **true**; otherwise, returns **false.**
15. `void clear()` — Empties the set of its elements.

Things to Think About

- What data members are needed for a set to maintain information about its current state?
- Why is the insert function the key to maintaining the condition of uniqueness in Set?
- You'll notice that some parameters are passed as constant references (`const string&`) as opposed to just the bare type (`string`). Do some research to figure out why this is useful. Do we really need to do this for primitive data types? Why or Why not?
- This project will have a series of extensions to it in which you may need to add new functions, change data structures, or change things that may effect many functions, how can you design your Set in such a way that these changes require the least amount of effort on your part?
- Although not listed specifically, you should be certain to have a constructor, destructor and copy constructor if necessary.
- Why do we need a copy assignment operator for the Vector class but not the Set class?

Testing Driver

You will be provided a testing driver (main.cpp) file which will help guide you in your development process and will also give you an idea of how you are doing. You can use this main file to compile your code with and run the executable to see if it crashes. You SHOULD NOT edit or even touch the testing driver – you should look at it to see what is being tested. The nature of the testing driver may help guide what you work on first and it may be more useful to take on a Vertical Development Approach. Along with the testing driver you will be given the header files for the classes we expect you to implement. The public members should not change (since you have already made this contract with users) but you may add/change private members since that is abstracted away. You are expected to change the header files to properly document/comment it.

Grading

The assignment grade is broken down as per the project grading rubric which is on Piazza. Please take a look at that for submission instructions and expectations.

Due Date

The due date for this project is February 15th, 2017. Remember, no late submissions are accepted.