# Submission

- MyVector.h, VectorItr.h, ConstVectorItr.h, any other.h files that make up your solution
- Time it took Matthew: 1 hour and 15 minutes

# Description

You will be implementing the vector class as well as the both constant and regular iterators over it.

## My Vector

The class you will be creating is called MyVector. I have given you a .h file that describes the methods that I want you to implement in the class.

## VectorItr and ConstVectorItr

I have given you VectorItr and ConstVectorItr .h files that contain the methods I want you to implement. The difference between the two is that the ConstVectorItr only has a single operator* method that returns a const& to the item that it is pointing to whereas VectorItr has both a const and non-const version of operator*.

## Errors

If the user ever attempts to access an element of the vector that is out of bounds you should raise an `std::out_of_range` exception that has the following error string: ”`Out of bounds exception. Attempted to access the vector at " << pos << " but there are only " << size() << " elements. "`

# Restrictions

- You may not make any instances of or references to std::vector in your solution
    - There is a function that accepts std::vector& as a parameter and this is ok. You cannot however form a reference to this vector in your code to use later
- You cannot use any other container from the standard library to store your elements
    - For example map, set, vector, etc

# Hints

- Get the constructors, destructor, and overloaded outstream working right away
    - These are used in almost every test case so you'll need to get them working if you want any points
- One annoying thing about implementing the vector class is that you'll have to be able to create instances of objects that have **no default constructor** (the 0 argument constructor). This is a problem because `new` when called to create an array of elements always uses the default constructor. There is joke in CS that all problems can be solved with enough indirection and it turns out that that is the solution here. So instead of making an array of Ts you can make an array of T* or even better an array of unique_ptr<T>.
- Insert can be used to implement pushBack
- Erase can be used to implement popBack
- At can be used to implement [], front, and back
- To really increase your learning, try using and extending the generic iterator code that I gave you