# Assignment 2: Create a Vector

## Learning Outcomes

* Implement a dynamically resizable array and analyze its performance.

# Instructions

Our last program was to maintain a list of wizards the STL's vector class. In this assignment, you will be writing your own vector class. Implement class vector consisting of an array of a generic type.

Unlike a typical class with a header file (.h) and an implementation file (.cpp), everything will go into one file called "vector.hpp" file.

This class will require the following fields:

* mData: An array of items of a generic type.
* mCapacity: An integer representing the capacity of the vector, which is the number of items the array can contain. Hereafter known as the **capacity**.
* mSize: An integer representing the size of the vector, which is the number of items actually stored in the array. Hereafter known as the **size**. The capacity is always greater than or equal to the size.
* mAssignments: An integer representing the number of assignments that have been done in the course of adding elements to and resizing the vector. You only need to keep track of assignment statements that store an item in the array or copy an item into a new array.

Your vector class must also support the following public member functions. (Please define everything into a single vector.hpp file!):

* vector(): A default constructor which should call the initialization constructor with a value of 10.
* vector(const vector&): A copy constructor that takes a vector object and copies its contents into a dynamically allocated array with the same capacity.
* ~vector(): A destructor that deletes the data array.
* operator=(const vector&): vector& An overloaded assignment operator that deletes the current array and replaces it with a dynamically allocated array with the same capacity and contents as the provided vector object.
* getCapacity(): int Returns the capacity of the vector.
* size(): int Returns the size of the vector.
* getAssignmentCount(): int A constant member function that returns the number of assignments made. Note that the only assignments we will be keeping track of are ones that involve the underlying array directly. This will include push back operations (see below) and copies.
* operator[]: T This will allow the user to write or to read a particular element in your vector. You probably haven't done anything like this before and I will give you the code for this.
* push\_back(const T): void Adds an item in the next open position if there is room (update assigment count by 1). If there is no room, the application should allocate a new array that is twice the size of the current capacity and then copies all existing elements from the existing array to the new array. Then we delete the existing array and point our array field to the new array. This method should update the assignment count each time it performs and assigment. (This behavior of resizing the array should be its own private helper method called resizeArray(int) (where the passed int is the size of the new array).

## How to implement operator[]

The operator[] requires that you add TWO methods your vector class (one which is used for reading and the other for writing). You probably haven't implemented this before, so you'll be given the code. Here are the prototypes used in the vector class definition. The int passed to each method represents the index desired by the user.

T& operator[](int); // Write operation.  
const T& operator[](int) const; // Read operation.

The implementation of these two methods is identical: return the element in your array at the desired index.

template <class T>  
T& vector<T>::operator[](int index) {  
 return mData[index];  
}  
  
template <class T>  
const T& vector<T>::operator[](int index) const {  
 return mData[index];  
}

Once these two methods are added to your vector class, you may now do things such as this (provided that everything else in your vector is written correctly):

vector<int> list;  
list.push\_back(10);  
list.push\_back(20);  
list.push\_back(30);  
std::cout << "Last element: " << list[list.size()-1] << std::endl;

## Return to your Wizard.

In order to create a dynamically allocated array of Wizard objects, the Wizard class needs a default constructor. This constructor doesn't need to do anything, but it does need to be there.

This is my default wizard constructor code. I put this into my wizard.h file.

wizard() {}

## Return to your fellowship library.

* In your fellowship.h file, remove the import statements to <vector> and replace those with the local version of "vector.hpp" that you made.
* You may have to update parts of your fellowship.cpp file. For example, I had to rewrite my for-loop to account for a data structure that doesn't use iterators.
* You will have to add a method int getAssignmentCount() to your fellowship library to return the number of assignments used in your vector class.

## Return to the main function.

Next, return to your original main function.

* Add a new option to your menu to add a wizard to the listing times. For example, this option will still prompt a player for a new wizard and then a number, it will add that wizard times.
* Finally, add an option that will display the current assignment count of the vector (using the fellowship's getAssignmentCount method that you had to add to your fellowship).

Play with this application.

* How many assignments are made if you insert 100 elements?
* How many assignments are made if you insert 1,000 elements?
* How many assignments are made if you insert 10,000 elements?
* What can you infer about the application if you insert 100,000 elements? (Your computer might not be able to handle this, but you should predict what it would do if you could.)

# What to Hand In

Upload a zip file containing the following files to the drop box on D2L:

* vector.hpp - the header file containing the entire class declaration and external definitions of the vector class. Normally, the definitions would be in a separate file, but class templates need to be handled differently in order for them to be compiled properly by the C++ compiler.
* All files should be updated from the last homework assignment to fit this new vector and included.
* Make sure your name, CSCI 3250, and Programming Assignment 2 appear in comments in all of your files (updated) from the last homework assignment.
* Note: NO CREDIT will be given to programming assignments that do not compile.
* Make sure you have compiled and tested your program before handing it in.