Lab 4 – Dynamic Array

Overview

In this assignment you are going to create a class called DynamicArray. The purpose of this class is to store an array that will make use of the C++ concept of templates to store any type of data. In addition, you will write functions to expand or shrink the array as needed.

Description

All programs need storage of some kind as they execute. Arrays are the most basic, and probably the most common form of storage. However, because of their fixed nature they can be problematic to use, particularly if you need to store data of varying lengths as your application's needs change over time. One of the first data structures people start to use in any programming language is some form of an expandable array. Java has the ArrayList class, C# has the List class, and even C++ has the vector<T> class.

The DynamicArray class you create will be very similar to the vector<T> class. The concepts behind allocating and freeing memory will be applicable when implementing numerous other data structures and algorithms, so you aren't reinventing the wheel here. You're learning how wheels are made, so you can build your own, custom version of them as needed in later projects.

Templates

Templates in C++ can be a bit confusing at first, but they allow you to reuse code very easily. By defining a class as a template, you can create multiple instances of that class to use different types. For example:

```
Foo<int> a;   // Instance of Foo which uses ints
Foo<float> b; // Instance of Foo which uses floats
Foo<Bar> c;   // Instance of Foo which uses Bars
```

The DynamicArray class you write is going to be a storage container, and while you may create storage containers in the future that are custom-built to solve a single problem, reusable code can make your life much easier.

DynamicArray

Storage

Internally, a class like this does not store much data. There are only three data members that you are required to store (though you may add any additional variables you see fit to help you write this).

- Data - A pointer to the data you are storing. This will be based on the ***type-id*** you define for the template. This pointer is the core of this class. It's where the magic happens.
- Size – How many objects are being stored currently?
- Capacity – How many objects COULD be stored? This is not the same as the size. Think of a 2-car garage. Its capacity is 2, but at any given point it may have 0, 1, or 2 cars parked in it.

Functions

A class like this is all about its functionality. Basic arrays do absolutely nothing. A class like this becomes much more useful when it can DO something. First, the basics—data access. Then, the mutators. Last, but not least, constructors and the Big Three.

```
public:
    /*====== Accessors ======*/
    unsigned int GetCapacity() const;
    unsigned int GetSize()  const;
    const T *   GetData() const;

    // Return a specific element from the internal array
    const T &operator[](unsigned int index) const;
    T & operator[](unsigned int index);

    // Just like the brackets operator, but with a function
    const T &At(unsigned int index) const;
    T &At(unsigned int index);
```

Why use a function AND an overloaded operator to do the same thing? Truthfully, you don't NEED to (in your own projects, that is—you DO need to for this assignment), but it's good experience to see that there is more than one way to accomplish something in code.

```
/*====== Mutators ======*/
// Add an item onto the end of the array, if there is room.
// If not, the array will have to be resized to make space for one more.
void Add(const T &data);
// This is the beefcake of a function that does the resizing.
void Resize(unsigned int newSize);
// The data killer, the silent assassin, the slayer of array elements.
// Removes an element from an array, and then shrinks the array to fill
// the gap.
void Remove(unsigned int index);
```

The Resize() function should print out the old capacity, as well as the new capacity. You wouldn't normally have this in the final version of a class like this, but this can be helpful to illustrate what's happening. You can print this with a line such as this:

```
cout << "Resizing... old capacity: " << (TheOldCapacity) << " New
capacity: " << (TheNewCapacity) << endl;
```

And of course, the Big Three, or Trilogy of Evil, whichever you prefer. Plus, a pair of constructors. They might be evil. They might not be. Hard to say. But keep an eye on them, just in case.

```
// Default constructor
DynamicArray();
// Constructor with an initial capacity
DynamicArray(unsigned int);

// Trilogy of evil...
DynamicArray(const DynamicArray &d);
DynamicArray &operator=(const DynamicArray &d);
~DynamicArray();
```

Exceptions

For functions which attempt to do something with a particular index, you will want to check to see if the indicated index is in a valid range. (Arrays have a valid range from 0 to arraySize-1.) If it isn't, you want to **throw** an exception--there are many ways you can handle exceptions, but in this assignment just throwing an exception of type runtime_error("Error! Invalid Index") will be acceptable. There is a section in your textbook which has examples of throwing and catching exceptions. You will also have to include the file <stdexcept>, and be sure you are using std::runtime_error.

Examples

A few examples of things you might want to do with an array:

```
// Get some names from the user
DynamicArray<string> names;
for (int i = 0; i < 5; i++)
{
    string userInput;
    cin >> userInput;
    names.Add(userInput);
}

// Store the letters of the alphabet.
DynamicArray<char> ABCs;
for (char letter = 'a'; letter <= 'z'; letter++)
    ABCs.Add(letter);

// Split 1000 numbers into even and odd groups
DynamicArray<int> evens(500);   // Create with pre-allocated capacity
DynamicArray<int> odds;
for (int i = 0; i <= 1000; i++)
{
    if (i % 2 == 0)
        evens.Add(i);
    else
        odds.Add(i);
}
```

Tips

A few tips about this assignment:

- Remember the "Big Three" or the “Rule of Three”
    - If you define one of the three special functions (copy constructor, assignment operator, or destructor), you should define the other two. The entire purpose of this class is based on the concept of dynamic memory allocation.
- Don't try to tackle everything all at once. Work on one function at a time. Be sure one thing works before moving on—that one thing could very well break a lot of other things later!
- Create your own tests! While the assignment submission on zyBooks has a lot of tests you will ultimately be graded on, you can/should be writing your own code to test functionality as you go. This also helps you to better understand your own code. Plus, it’s just good practice.
- Refer back to the recommended chapters in your textbook as well as lecture videos for an explanation of the details of dynamic memory allocation or templates
    - There are a lot of things to remember with memory allocation and deallocation
- If you think some of these tips seem familiar, you aren’t going crazy… it’s almost as if they are applicable to lots of situations in programming…

Program output:

```
**** Testing the Dynamic Array ****

Integer container: Initial capacity of 0
Adding 5 items...
Resizing... old capacity: 0 New capacity: 1
Resizing... old capacity: 1 New capacity: 2
Resizing... old capacity: 2 New capacity: 3
Resizing... old capacity: 3 New capacity: 4
Resizing... old capacity: 4 New capacity: 5
1
2
4
8
500
Capacity: 5
Size: 5

Float container: Initial capacity of 10
Adding 5 items...
50.5
33.6667
25.25
20.2
16.8333
Capacity: 10
Size: 5

Shrinking container to 2 elements...
Resizing... old capacity: 10 New capacity: 2
50.5
33.6667
Capacity: 2
Size: 2
Attempting to access index[50]...Error! Invalid index
```

```
Resizing... old capacity: 0 New capacity: 1
Resizing... old capacity: 1 New capacity: 2
Resizing... old capacity: 2 New capacity: 3
Resizing... old capacity: 3 New capacity: 4
Resizing... old capacity: 4 New capacity: 5
100
999
200
300
400
Removing '999' from the list...
100
200
300
400
Creating a copy of the list (testing the copy constructor)...

Printing copy of dynarray...
100
200
300
400

Attempting to remove index[100]...
Error! Invalid index
Resizing... old capacity: 0 New capacity: 1
Resizing... old capacity: 1 New capacity: 2
Resizing... old capacity: 2 New capacity: 3
Resizing... old capacity: 3 New capacity: 4

Printing heroes from dynarray...
Conan Hitpoints: 100 / 120
Thor Hitpoints: 150 / 150
Merlin Hitpoints: 80 / 85
Bob Hitpoints: 25 / 26

Cloning our heroes (testing the assignment operator)...
Printing cloned heroes from dynarray (and changing their names)...
Evil Conan Hitpoints: 100 / 120
Evil Thor Hitpoints: 150 / 150
Evil Merlin Hitpoints: 80 / 85
Evil Bob Hitpoints: 25 / 26
```