Assignment 1

COMP-322, Winter 2017

Due: March 9th, 2016 - 11:55pm

Please read the entire pdf before starting. You must do this assignment individually.

Implementing Vector class with iterators

The purpose of this question is gain a better understanding of the standard library methods by implementing a mini version of them. You will also get practice defining and using classes in C++

All the code in this assignment should be placed in a file `DynamicArray.cpp`. (It is better practice to use separate files with include statements, but having everything in just one file will simplify the grading process.)

As a general guideline, if you wish to add further utility methods to either of your two below classes (such as making the iterator a two way iterator), that is encouraged, as long as you have the expected functionality. Note though that one method that you may not add, is a method to return the private property of the backend array in the dynamic array class.

DynamicArray class (60 points)

First define a new type `DynamicArray`. The type should be templated, so that it can work with all types. We will see this in lectures, but to do so, you must insert the following statement before both the class header and all of your methods:

```
template <class T>
```

(T can be replaced by anything else). You also will need to include the ¡T¿ inside of your method headers when defining them. Such as `void DynamicArray<T>::add(T item)`.

See the lecture notes or `http://www.tutorialspoint.com/cplusplus/cpp_templates.htm` for more details.

Your new type should have the following methods defined on it:

- add(T item) : adds an element to the end of the DynamicArray. T is the type used in the template above. It should return void.
- size: returns how many elements are in the DynamicArray. (Note: this is different than the capacity. See below for further discussion.)
- remove(int i): removes the element at position i of the DynamicArray. All elements at positions greater than `i` should be shifted down by one at this point. It should return the removed element.
- DynamicArray() : This should initialize the backend array (see below) to have a size of 10 and set the capacity to be 10 and current position to be 0.
- $\sim$ DynamicArray() : This destructor should handle any memory that has not been freed yet.

- Overload the [] operator so that you can access any arbitrary spot of the dynamic array. Hint: Remember to return a reference to allow for setting values this way.
- begin() : This method returns a `DynamicArrayIterator` object with a state that is at the beginning of a `DynamicArray` set up to iterator forward.
- end() : This method returns a `DynamicArrayIterator` (see the next question) object with a state that is one past the end of a `DynamicArray` set up to iterator forward.
- r_begin() : This method returns a `DynamicArrayIterator` object with a state that is at the beginning of a `DynamicArray` set up to iterator forward.
- r_end() : This method returns a `DynamicArrayIterator` object with a state that is one past the end of a `DynamicArray` set up to iterator forward.

Some hints are below:

- Remember that the way a dynamic array such as the vector class (and now our class) works is it keeps as a private property a "regular" non-resizable array as well as an int for the capacity of the regular array and an int for the next available free spot of the array. This array is always larger than or equal to the "virtual" dynamic array. As we add to the dynamic array, using the `add` method, we insert into the next available spot of the array (using the private property). After inserting the element, we update the next available spot.
- Occasionally, the back end array will be full before you add something, and it will be necessary to create a bigger array, copy values from the old array into the new one, and then point the private property at the new one. As a good balance between wasting space and wasting excessive time copying, you should double the size of the array each time. [1] Make sure to use proper memory management to deal with the old array!
- Since the DynamicArray class depends on the DynamicArrayIterator class which is not yet defined (and vice-versa), you will need to add a prototype statement at the top of your file, before your class. If you choose to put DynamicArray class first and then your definition of DynamicArrayIterator, you should insert the following statement before the DynamicArray class. (If you choose to put DynamicArrayIterator first, you will need to put a similar statement but with DynamicArray instead. You also can just put both :))

  ```
  template <class T> class DynamicArrayIterator;
  ```

- For all of the methods that return iterators, you will need to think very carefully about values needed to instantiate a DynamicArrayIterator object such that you can start with the iterator returned by the begin method, and continue to use the plus plus operator until eventually you will reach a state where the object has the same contents (i.e. matches true when using the == opertaor) to the object returned by end().

DynamicArrayIterator (40 points)

Here you should define a new type `DynamicArrayIterator` which can be used to traverse an array either forwards and backwards based on the value of the private properties. This type will also need to be templated to work fully with the other one.

You should have the following private properties:

- int currentPosition
- DynamicArray* array
- int direction - This is used to store how to move each time. A value of 1 indicates it is moving forward and a value of -1 indicates it is moving backwards. This value never changes after its initialization.

[1] See https://en.wikipedia.org/wiki/Dynamic_array#Geometric_expansion_and_amortized_cost for more on why this is.

First, write the following methods/overloaded operators

1. A constructor that takes as input a pointer to the DynamicArray, the initial current position, and the direction it is moving.
2. Overload the * operator so that values can be set and fetched.
3. Overload the increment operator. For simplicity, make this a void function so that it does not matter if you do pre or post fix –though you will need to ensure you call it properly from your main method. Depending on the direction of the iterator, this should either increase or decrease the current position.
4. Overload the == operator to allow a comparison with another DynamicArrayIterator.
5. Overload the != operator to allow a comparison with another DynamicArrayIterator.

Main program (0 points)

To test all of this out, you should create a main function which creates a DynamicArray as well as a DynamicArrayIterator and makes sure they work correctly. This will not be graded though.

Here is some sample code to help demonstrate the usages of your types, along with the expected output.

```
int main() {
  DynamicArray<int> foo;
  foo.add(3);
  foo.add(2);
  cout << foo[1] << endl;
  foo[1] = 10;
  cout << foo[1] << endl;

  for(DynamicArrayIterator<int> a = foo.begin(); a != foo.end(); ++a) {
    cout << *a << endl;
  }

  for(DynamicArrayIterator<int> a = foo.r_begin(); a != foo.r_end(); ++a) {
    cout << *a << endl;
  }
}
```

Expected output:

```
2
10
3
10
10
3
```

What To Submit

You should submit one file called `DynamicArray.cpp`.