CMPT 225 D2
Fall 2020
T.Shermer

Assignment 3
ArrayList with Iterators

Due Oct 26 at 23:59

You are to write a generic (templated) C++ ArrayList class, along with an Iterator nested class. There will also be some code for testing this implementation.

The principal class will be called **ArrayList,** and it should be templated on the type of the object it holds. (E.g. ArrayList<int> or ArrayList<string>.) It will have a private array in which it keeps the elements. The array is required to grow when it reaches its capacity, so there is never a problem inserting elements. The array should also be used in a **circular fashion** in order to provide for fast insertFront() and removeFront() methods. (The circular use of arrays has been covered in lecture.)

ArrayList should have the following member functions (T is the template parameter).

```
type    name

T&      operator[](int i)    // returns the element at index i of the ArrayList.
                             // note that this is not index i of the underlying array.
T&      front()              // returns the element at the front of the ArrayList.
T&      back()               // returns the element at the back of the ArrayList.
void    insertFront(T& e)            // insert at the front of the ArrayList.
                                     // Grows underlying array if necessary.
void    insertBack(T& e)             // insert at the back of the ArrayList.
                                     // Grows underlying array if necessary.
void    insert(Iterator p, T& e)     // insert before the iterator p.  This could involve growing
                                     // the underlying array and/or copying elements forward in it.
void    removeFront()                // removes the front element of the ArrayList.
                                     // throw an EmptyListException if ArrayList is empty.
void    removeBack()                 // removes the back element of the ArrayList.
                                     // throw an EmptyListException if ArrayList is empty.
void    remove(Iterator p)           // removes the element at the iterator p from the ArrayList.
                                     // may involve copying of elements.
                                     // throw if ArrayList is empty.
int     size()                       // number of elements in ArrayList
bool    empty()                      // is size = 0?

Iterator begin()                     // an iterator to the first element

Iterator end()                       // an iterator to the element after the last element.
```

It should also have a constructor and a destructor. As a constructor argument, it should take a capacity, which is the size of the array that it makes to hold the elements. This argument should default to 4.

If there is a insert-type operation and the underlying array is at capacity (i.e. it's full), then make a new array that has twice the size of the old capacity and copy the elements from the old array to the new array. Delete the old array.

If an iterator argument is used in a function, throw an InvalidIteratorException if the iterator is not valid (too small or too large).

EmptyListException and InvalidIteratorException are new classes that you should create. They can be created in the same file as the ArrayList or in their own files.

Iterators should have the following member functions:

```
T&        operator*()              // gives the reference to the object at the iterator's position.
Iterator  operator++()             // moves the iterator one forward in the ArrayList
Iterator  operator—()              // moves the iterator one backwards in the ArrayList
bool operator==(Iterator& i)       // compares this iterator with the iterator i.  Works even if i
                                   // is an invalid iterator.
```

Iterator should be a nested class within ArrayList.

Note that the functions above (in both classes) are given approximately. Add references (&) where necessary or sensible, and add **const** to arguments and functions if they are const. I also haven't defined all the data members that the classes should have.

To exercise your ArrayList functionality, your main routine should call three test functions. These are: testArrayListUnderflow(), testIntegerIterator(), testStringIterator().

testArrayListUnderflow() will check to see if your exception is thrown when the ArrayList underflows. In it, you should use a three try-catch statements that catches EmptyListException. Inside the first try block, create an ArrayList of ints, insert an element onto it, call removeFront() twice, and then print "did not catch exception". Inside the catch block, print "caught EmptyStackException". Inside the second try block, do the same thing except call removeBack() twice instead of removeFront(). Inside the third try block, do the same thing with remove(p), where p is the iterator begin().

testIntegerIterator() should set up an int ArrayList of 4 elements, putting six elements on it with insertBack(). (This tests the ArrayList growth). Then it should print the six elements individually, by looping through the ArrayList with an iterator using the standard idiom:

```
for(ArrayList::Iterator iter = L.begin(); iter != L.end(); iter++) {
        cout << *iter << " ";
}
cout << endl;
```

Next, testIntegerIterator should removeFront() three times, then insertBack() three times. Now the underlying array should be at size 8 and have the head after the tail in the underlying array. Again print the items of the ArrayList by looping using the standard idiom.

testStringIterator does the same thing as testIntegerIterator does, except it uses string data.

Add any other tests you feel like adding, but call and print them after the required tests.

Print a blank line between each pair of tests in main. Add any output that helps clarify what the testing is doing.

All classes should be in separate files (.cpp and/or .h) named with the class name. (The exception classes are exceptions to this.)

You will be judged on correctness of your code and on code style, so don't forget to keep your code clean as you develop it! (Or at the very least, clean it up before submission. We don't want to see untidy code.)