CST 238 Spring 2020, Dr. Gross, CSUMB

Homework 7. Dynamic Lists

Due: Tuesday, April 14, 11:55pm

Objectives:

- Use dynamic arrays to implement a list
- Resizing the underlying dynamic array

How to turn in:

- Submit the List.cpp file ONLY
- If you fail to follow instructions, your code could be either incomplete or have incorrect code (e.g., the class declaration); this will cause you to lose points, so *follow instructions carefully*

Part 1. Getting Started (5 pts)

Fork the Starting Code - List with Dynamic Array, and split into three files:

- List.h (header or interface file)
- List.cpp (implementation or body file)
- main.cpp (runner)

Make sure that the appropriate code is in the appropriate files:

- All files will need to have
    - Appropriate includes, based on what is used in the file
    - Your name
    - The date
- The interface file should contain
    - A header guard, using #ifndef/#define/#endif, to prevent the declarations from being invoked twice
    - The class declaration
    - The header information required for all assignments (name, ID, date, synopsis)
- The implementation file should include the class definitions
- The runner file should only contain a main() function

Part 2. Big 3 (15 pts)

Continue with the same code, and implement:

- A destructor, which should free all dynamic memory
- A copy constructor, which should
    - Copy all non-dynamic variables
    - Allocate memory for all dynamic variables
    - Copy the contents of any dynamic variables
- An overloaded assignment operator, which should
    - Verify that we are not doing self-assignment

- Copy all non-dynamic variables
- Check if the dynamic memory allocated is the correct size
  - If not, free the current memory and re-allocate the correct size
  - Copy the contents from the righthand operand's dynamic memory to the local object
- Return a reference to local object

Part 3. Resizable List (20 pts)

Extend the List to resize automatically when it needs to. Right now, this List uses dynamic memory, but it will only hold a set number of items. In order to grow, it must be changed so that, when we attempt to insert or append a new item into a full list, it resizes itself.

For this part, you have to implement the five member functions:

- `resize()`, which takes an integer and changes the capacity of the list to that amount. This *will* change its `myCapacity`, and *might* change its `mySize`. It should retain any elements in the underlying array that it can (e.g., calling `resize(5)` on a list of 10 elements should keep the first five elements).
- `getCapacity()`, which returns the value of `myCapacity`
- `getSize()`, which returns the value of `mySize`
- `append()`, which takes an item value and adds it to the end of the list
- `removeLast()`, to delete the last item in the list, returning true if successful, or false if the list is empty
- `setCapacity()`, which takes a new capacity and grows or shrinks `myArray` as necessary, keeping all possible elements
- You will also need to update the `insert()` function; right now, it won't allow inserts past the limit of the underlying array, but now it needs to resize as necessary

One example main function and run:

```
int main() {
   // Create a small array with capacity is 1.
   List intList (1);

   cout << "=== Solution: 1 0 ===\n";
   cout << intList.getCapacity() << " " << intList.getSize();

   intList.append(10);
   intList.append(20);
   cout << "\n=== Solution: 2 2 ===\n";
   cout << intList.getCapacity() << " " << intList.getSize();

   intList.append(30);
   cout << "\n=== Solution: 4 3 ===\n";
   cout << intList.getCapacity() << " " << intList.getSize();
```

```
   cout << "\n==== Solution: 10 20 30 ========\n";
   intList.display();

   intList.removeLast();

   cout << "\n==== Solution: 10 20 ========\n";
   intList.display();
   return 0;
}
```

And the test run:

```
./main
=== Solution: 1 0 ===
1 0
=== Solution: 2 2 ===
2 2
=== Solution: 4 3 ===
4 3
==== Solution: 10 20 30 ========
10  20  30

==== Solution: 10 20 ========
10  20
```

A second example:

```
int main() {
   List intList (5);
   intList.append(10);
   intList.append(20);
   intList.append(30);
   intList.append(40);
   cout << "=== Solution: 5 4 ===\n";
   cout << intList.getCapacity() << " " << intList.getSize();
   cout << "\n=== Test setCapacity() ===\n";
   intList.setCapacity(2);
   cout << "\n==== Solution: 10 20 ========\n";
   intList.display();
   intList.append(77);
   cout << "\n=== Solution: 4 3 ===\n";
   cout << intList.getCapacity() << " " << intList.getSize() << endl;
   cout << "\n==== Solution: 10 20 77 =====\n";
   intList.display();
   return 0;
}
```

and the test run

```
./main
=== Solution: 5 4 ===
5 4
```

```
=== Test setCapacity() ===

==== Solution: 10 20 ========
10  20

=== Solution: 4 3 ===
4 3

==== Solution: 10 20 77 =====
10  20  77
```

Final Notes:

- Read the sample run carefully to understand the requirements of the problem correctly
- Only the implementation file for this program should be uploaded to iLearn as `List.cpp`