**CS173 Intermediate Computer Science**
**Project 9: List with Dynamic Array**

## OVERVIEW

This is an individual project. You should complete this work entirely on your own. You should not use IDEAS or CODE from anyone else or from the internet or other non-approved resource. You may get ideas and code help from:

- The course professor
- Any example code we have done in class
- Anything posted on our Canvas page
- The class TA
- Our textbook

This project is one of two "stop-gap projects" mentioned on the syllabus. The skills in this project and the next are so essential that the CS department faculty have decided that students may not pass CS173 unless this project and the next are completed satisfactorily. If a student does not meet this requirement on their first submission, the professor will allow the student to continue to work on and resubmit the project to meet this requirement. **These projects are not assigned a point value used in the final grade, but completing them satisfactorily is required to pass the course. Deep understanding of these projects and their solutions will be critical to do well on Quiz 6 and the final exam, so use these projects as practice opportunities to learn what you need to re-study.** Come to office hours. Ask questions.

For this project, you will implement a List class in C++. Start by reading the List ADT to see what the List is supposed to do and how it is supposed to work. Our List will differ a good bit from the textbook List and from the python List.

The List we create will never be full. To do this, we need to create a dynamically allocated array for the List class. We can add things to the array as necessary, but once the array fills up, then we will have to create a new, bigger array and then be sure to copy things from the original (smaller) array to the new (bigger) array. We also need to prevent memory leaks and to do proper deep copying.

## DYNAMIC DATA

Your list uses a dynamically allocated array to hold the actual values. By using a dynamic array, the list can be made larger if it is full.

Notice the DEFAULT_ARRAY_SIZE declaration in List.h. Use this to create the initial array size for your list. It is private and only used internally.

If the user attempts an insert or append to a “full” list, then you will have to do the following steps:

- Create a new array that has twice the capacity of the current array.
- Copy the items from the current array into the new array.
- Delete the current array.
- Attach the new array to the List object properly.
- Complete the insert/append operation into the new array.

I suggest you create a private method that accomplishes all but the last step. You can add this private method to your List.h private section. Then call it from append/insert or any other appropriate place. You can use the "reallocate" method in the private section of the List.h file for this purpose.

Any List created from the copy constructor or assignment operator should have the same capacity as its source list.

This list capacity never shrinks for any reason, even if you remove items or clear the whole list.

You should be sure to fully test your dynamic array to see that it works correctly. You should also avoid memory leaks by making sure you remove (delete) memory appropriately.

Since you now have dynamic data, you must perform a deep copy instead of a shallow copy. Be sure you understand the difference between deep and shallow copying from our class discussion. You can also read the section in the textbook on deep copy (14.5). I have also posted YouTube videos to Canvas on deep vs shallow copying.

You will have to implement:

- default constructor
- copy constructor
- destructor
- assignment operator

All of these are specified in the .h file I am giving you, so you must implement them anyway. But these always must be implemented when you are using dynamic data because the default definitions for them do not appropriately handle dynamic data.

## TEMPLATE

Your list class will be a templated class. That means it will hold a yet-to-be specified datatype. Follow the example template class we walked through in class.

You will need to include the List.cpp file inside the List.h file. This is already done for you in the List.h file I have provided. You can no longer compile the List.cpp file separately because it is not yet a full class until the template type is specified.

## LOGISTICS

I have given you a complete List.h file. **You should not change the public part of this class**. In the private section, I have given you three variables. **You should not change or remove these variables.**

- The **arr** is the pointer to the dynamically allocated array.
- The **capacity** is the number of total spots in the array (used or unused).
- The **size** is the number of currently used spots in the array.

Note that *capacity* indicates the current size of the array whereas *size* keeps track of how many of these array spots are currently used. Do not confuse the two distinct variables.

You may add more private variables and/or private methods to List.h.

I have given you a driver file main.cpp and a makefile. You may change these files if you would like, but I suggest keeping an original copy somewhere. You will want to comment out the list operations until you implement them. Then uncomment them back in, one at a time, and test each one.

You are to create your own List.cpp file. **You are to submit BOTH your List.h and List.cpp** files as part of this assignment. **Your List.h file must have exactly the same public interface** – you cannot change any of this. You must submit both files together in the same submission. Do not submit one file in one submission and then the other file in a different submission.

## TESTING

I also suggest you add more extensive testing to your main program. Be sure to think about ways your list class might "break" and then try hard to break it by adding more code to the main program. The main.cpp file I gave you only does cursory testing. I will use a more extensive test file to see if your code works. Most of the advanced testing comes from testing the dynamic array, looking for memory leaks, and making sure deep copying is working correctly.

## LEVELS OF ACHIEVEMENT

Because of the nature of the stop-gap projects, the goal is to get as close to a perfectly implemented List class as possible, following the specifications in the ADT. Memory leaks, incorrect logic, and deviation from the behavior described in the ADT and this assignment document are serious concerns. If these issues are present in more than a couple of spots, that

can require resubmission. Use the resources available to you to make sure you understand how each method should work.