# CSCI 2312-001: Object Oriented Programming

# Assignment 6

For this assignment, you will use pointers with dynamic memory, templates, binary files and exception handling to build a multi-purpose, generic, Single Linked List. Anytime you are using dynamic memory, you should have a Copy Constructor, overloaded Assignment Operator, and a Destructor. Though this will be similar in functionality to the C++ Standard Template Library (STL) forward\_list, you will create your own using pointers (instead of forward\_list, list or vector).

Create a Student struct with a name[20] and a student id.

Overload the << for a Student.

Create a Node with templated data (so that you can have Nodes of any legitimate data type). Nodes also have pointers to other Nodes (that must be templated). Structs are acceptable means of describing Nodes.

Create a templated List class (templated with the same type of data that is used in your Node). This class should maintain a head pointer to templated Nodes. This is the start of your linked list. This class should have the following member functions:

**Appropriate Constructors**

**Copy Constructor**

* Used to give the values of an already declared list to another list
* Deep Copy
* Set the calling object’s headPtr to nullptr
* Copy every node from what is in the parameter to the object created with the Copy Constructor. Consider using your push\_back function.

**Overloaded Assignment Operator**

* Used to assign the values of one already declared list to another already declared list
* Deep Copy
* Delete anything that might be pointed to by headPtr (Considering using your pop\_back function)
* Copy every node from what is in the parameter to the called object created by the assignment (again consider using your push\_back function)
* Return the value of what is pointed to (return \*this). Remember the “this” pointer refers to the object that called the member function

**clear**

* Delete ALL of the nodes in the linked list
* If you are not concurrently enrolled in Data Structures, you can send your instructor an email asking for the code to the clear function

**Destructor**

* Get your clear working first…then AND only then should you call clear in your Destructor

**push\_front** and **push\_back**: Pushes data entered as input parameter (Nodes are not input parameter, but nodes will be created and added in the two push functions) into the front or back of the list. Remember that this is an array of characters, so you will #include <cstring>, then use functions like strcpy to set values of character arrays.

**pop\_front** and **pop\_back**:

* Removes data from the front or back of the list
* Should throw an exception if trying to pop an empty list.

**printList**: Function should be called printList and implemented with a loop and the << operator for the data. (<< does not work well when templated, so you can’t overload that operator for the list)

In main:

* Create a pointer to a list and assign it to a new pointer
* Demonstrate all functions with multiple ints and use cout statements and the printList to describe each the functions.
* Create a pointer to a second list and construct it with the first list
* Print out the second list to show that your copy constructor worked.
* Create a pointer to a third list, then use = to assign it to the first list
* Print out the third list to show that your assignment operator worked.
* Demonstrate exception handling of pop\_front.
* Delete both pointers
* Declare a list of type Students
* Read in Students from provided binary file students.dat.
* Use push\_front to create a linked list of students. Note: if you don’t get 3 records read in from students.dat, you could try writing some binary students before reading them in…then comment that out once you get the students.dat to read in properly.

Remember your ROCkET Analysis. You really need to draw out this functionality on paper. When dealing with pointers, it is also critical that you code just a little at a time, then evaluate that functionality. Refer to HW2, where you used the debugger. If you get a run time error, you might be trying to access a null pointer, or perhaps you are not setting deleted links to null pointer.

**Template reminders**: Templates must be defined in the same compilation unit, or you will get a linking error. You should name your definition of a templated class with a .h file. Name your implementation with a .hpp. Then in main, you need to #include both the .h and .hpp files. Because you did a #include of the .hpp file, you do NOT add this to your makefile (if you are using a Linux system)…just the .cpp files. Likewise, if compiling using Visual Studios or XCode, you must remove that code from the project, or add it to the header files(remove not delete!). If you do not you will get a linking error. Since we are templating the Node type, both struct and Class definitions for the Node and list and implementations should have template <typename DATA> at the top of each member function or definition. Then to declare an object you would call like a vector with Node<DATA> or List<DATA>

**Submission**: Like the other homeworks, you must turn in a single zipped file called LastnameHW6 with your .h, .hpp, .cpp files, a readme.txt, and a makefile…all tested on the csegrid.ucdenver.pvt. Remember, if you include a file, you do NOT compile it.