Project 6 Documentation

Purpose of the Program

==================

The core objective of the project involved creating our own unique ArrayList and linked list classes complete with member variables and functions, with both pointer-based and bracket notation used for array manipulation. The source code consisted of six files, with three pairs of header and that contained all the details for implementing all the functions of the ArrayList and NodeList classes as well as the DataType and Node classes. The last of the source code files served as a test driver for both of the two class functions, ArrayList and NodeList. Finally, a Makefile was created to compile the source code files together to form the program.

Program Design

=============

The program works to implement the functions and member variables of a unique array list and linked list classes designated as ArrayList and NodeList respectively. The ArrayList source code consists of two separate files, one of which is a header file called ArrayList.h while the other file is called ArrayList.cpp. The ArrayList.h header file stores all specifications that are required for the ArrayList, mostly being function prototypes and initialized member variables. The NodeList source code consists of two separate files, one of which is a header file called NodeList.h while the other file is called NodeList.cpp. The NodeList.h header file stores all specifications that are required for the NodeList, mostly being function prototypes and initialized member variables.

The ArrayList class contains the private data members m\_array, a DataType-type pointer which always points to the dynamically allocated data for any ArrayList object, m\_maxsize, an int that denotes the total number of spaces currently allocated for m\_array, and m\_size, an int that describes the current number of spaces allocated for m\_array that are occupied. In addition, every time the dynamically allocated memory is changed, m\_size is initialized and updated. The ArrayList class also has two private helper methods of grow() and trim(). Grow will deallocate the memory currently occupied by m\_array and then reallocate a new chunk of dynamic memory for m\_array that is large enough to accommodate an increased memory size for the addition of new elements that make m\_size exceed the old value of m\_maxsize. Trim does the opposite, responsible for deallocating the dynamic memory pointed to by m\_array and reallocating new memory that is just enough to fit in m\_size number of elements that m\_array currently has. The NodeList class contains the private data member m\_head, a Node-type pointer which always points to the first Node element of the NodeList data structure.

The ArrayList class has the following public member functions: default constructor, parameterized constructor, copy constructor, destructor, insertAfter, insertBefore, removeForward, removeBackward, find(), first(), last(), size(), the operator=, operator[], empty, and clear. The default constructor will instantiate new ArrayList objects with no valid data, while the parameterized constructor instantiates new ArrayList objects that are initialized to hold a copy of the C-string passed as a parameter. The ArrayList copy constructor will instantiate a new object that acts as a separate copy of the data of the ArrayList object being copied, while the destructor destroys the instance of the m\_array. Size() will return the size of the currently allocated m\_array, first() returns the first element of the ArrayList object, and last() will return the last element of the m\_array; find() will return a pointer to the first element of m\_array that has the same value as the passed target. The operator= assigns a new value to the calling object’s DataType data based on the other ArrayList object passed as a parameter. The bracket operator will allow by-reference accessing of a specific DataType value at the index passed in within the allocated m\_array. Empty() will check to see if the calling list object is an empty list object, while clear() clears the m\_array of an ArrayList object. InsertAfter inserts a value into m\_array after a target value and returns a pointer to the inserted value while insertBefore does the same thing except insert the value before a target value. RemoveForward removes a value and returns a pointer to the element after the one removed; function removeBackward also removes a value but returns a pointer to the element before the one removed. Finally, ArrayList has a friend function operator<< that outputs the data for a ArrayList object to the file or terminal depending on the type of output stream variable passed in as its parameter.

The ArrayList.cpp file is solely responsible for the implementation of the ArrayList class member functions that are first specified in the ArrayList.h header file, responsible for the actual mechanics of how all the functions will carry out in the program itself. In addition to these source code files, the proj5.cpp source code file acts as a test driver that demonstrates the functionalities of the ArrayList class. The test driver does this by using a variety of strings and calls to each of the public member functions in the ArrayList class one-by-one in code sections tagged by the appropriate number corresponding to each function.

The NodeList class essentially has the same functionality as the ArrayList class, with identical functions that are responsible for identical tasks. The key difference is that all these functions have been rewritten to perform the same operations on linked list data structures that use Nodes instead of allocated arrays. The removeBackward function for NodeList class also is modified in that it returns a pointer to the value after the value it removes from the linked list NodeList object rather than the one before.

Problems of Program

================

During the process of coding for the program, there were some issues that turned up concerning exact details of implementation for the copy constructors of the ArrayList and NodeList class that had been responsible for creation of object copies. In addition to this, there were some problems and concerns about the function of the parameterized constructor for the NodeList class as well as other functions in the ArrayList and NodeList class such as the insertAfter and insertBefore functions along with removeBefore. However, with time all these issues with implementations of the class functions were resolved in due time with diligence and persistence in the debugging process.

Changes to Program

================

I would have liked to double check the parameterized constructor for the NodeList class once more, just to make sure that it could have worked and been tested properly using the test driver code, despite the fact that the logic behind the parameterized constructor for NodeList is sound and works in theory. With the repairs to the code of the parameterized constructor for NodeList, I could also have made some kinks in the rest of the program disappear since some of the test driver code relied on a NodeList object that was called using the parameterized constructor method.