Bryan Kline
Project 11 Description
11/26/15

Project 11 is the last in a series of projects on data structures which is focuses on lists. There are two types of lists that are the focus of this project, an array-based list and a node-based list. Unlike stacks or queues, lists have a cursor that can move and items in the list can be added or removed where the cursor is located. As before, the array-based list has a predetermined maximum size and an actual size while the node-based list has only a cursor and a head, both pointers to type Node, which are containers that hold a value and a pointer to the next Node in the list. This project consists of only the implementation files (no header, main driver, or makefile) for the two lists which were written in C++ using gedit and complied and run in Ubuntu/Xubuntu with Valgrind and are free of memory leaks. The documentation for these files will be broken into two parts corresponding to the files, list1.cpp and list2.cpp, the array-based and Node-based List class implementations, respectively.

Array-based List (list1.cpp):

The array-based List class has a default constructor, a copy constructor, and a destructor, member functions gotoBeginning(), gotoEnd(), gotoNext(), gotoPrior(), insert(), remove(), empty(), full(), and clear() as well as an overloaded assignment operator and is a friend of an overloaded insertion operator function. It's data members are three ints “size”, “actual”, and “cursor”, and an int pointer “data”. As with stacks and queues the default constructor allocates memory for “data” of the size passed into it as a parameter and sets other data members to default values, and the copy constructor allocates memory of the size of the List passed in as a parameter, other data members are set to the values of the List passed in, and then a counter controlled loop goes through the List and copies in its values into the calling List. The destructor frees the memory “data” points to if it's not pointing to NULL and sets all data members back to default values.

The function gotoBeginning() sets “cursor” to zero if the List isn't empty, gotoEnd() sets “cursor” to “actual” if the List isn't empty, gotoNext() increments “cursor” if the List isn't empty and “cursor” isn't at the end of the List, and gotoPrior() decrements “cursor” if the List isn't empty and “cursor” isn't at the beginning of the List. All the functions that move the cursor return bools corresponding to whether or not the move was successful as do the functions empty(), full() and clear(). The function empty() checks if either “size” is less than one or “actual” is negative one and returns true if either are true, the function full checks if “actual” is equal to “size” minus one and returns true if it is, and the function clear() first checks if the List is empty, if it is it returns false otherwise “cursor” and “actual” are set to negative one.

The function insert() first checks that the List isn't full, then if it's not it checks that it's empty and if it is then “cursor” and “actual” are incremented and the char passed in as a parameter, “letter”, is set to the element that cursor is at, otherwise a counter controlled loop moves everything in List toward the end by one first. Finally, true is returned if this was successful and false returned otherwise. The function remove() first checks that the List isn't empty and if not it then checks whether or not “cursor” is at “actual” in which case “letter” is set equal to the char at cursor“ and both “cursor” and “actual” are simply decremented otherwise a counter controlled loop moves the elements in the array towards the beginning first. True is returned if that was successful and false returned otherwise.

The List class overloaded assignment operator first checks that “data” is not NULL and if not then the memory “data” points to is freed and then new memory is allocated to it of length “size” from the List passed into it as a parameter and then a counter controlled loop goes through that List and copies in the chars from its array into the calling List's array and data members are set to those of the List passed into the function and “this” dereferenced is returned. The overloaded insertion operator that is a friend of the List class first checks that the List isn't empty, if it is it simply prints a new line to the screen, and if it isn't then a counter controlled loop goes through the List and prints each element to the screen, putting brackets around the element at cursor.

Node-based List (list2.cpp):

The node-based List class has all the same functions and overloaded operators as the array-based List class but its data members are instead two Node pointers, “head” and “cursor”, and the Node class has a default constructor and data members “data” and “next”, a char and a Node pointer respectively. The default constructor simply sets “head” and “cursor” to NULL while the copy constructor takes in a List object by reference and, if that List isn't empty, goes through that List and creates new Nodes with the values from the List passed in as a parameter used in the Node's copy constructor and “cursor” is set to the same position as the List passed in. The destructor, if the List isn't empty, goes through Node by Node deleting the Nodes and then sets “head” and “cursor” to NULL. The Node class copy constructor takes in a char and a Node pointer and simply assigns its data members “data” and “next” to those passed in as parameters.

The function gotoBeginning() sets “cursor” equal to “head” if the List isn't empty and returns true, otherwise false is returned and the function gotoEnd(), if the List isn't empty and if “cursor” isn't already at the end, moves through the List with an event controlled loop until it reaches the last Node, at which point “cursor” is set to that Node and true is returned, otherwise false is returned. The function gotoNext() moves “cursor” to the next Node if the List isn't empty and if it isn't at the end and true is returned, otherwise false is returned, while the function gotoPrior() makes a temporary Node pointer, sets it to “head”, and moves through the List until its “next” pointer is “cursor”, at which point “cursor” is set to the temporary pointer and true is returned, otherwise false is returned. The function empty() checks that “head” and “cursor” are not NULL and true is returned if they are not, false otherwise while the function full() simply returns false as a linked List can never be full and the function clear() is identical to the destructor.

The function insert() takes in a char and checks that the List isn't full, if it is the false is returned and if not then if it's empty then “head” has a Node allocated to it and the char is passed to the Node's copy constructor and “cursor” is set to head, otherwise a temporary Node pointer has a Node allocated to it, the char is passed into the copy constructor as is the “next” pointer of “cursor”, “cursor” is linked up to the new Node and then “cursor” is set to the temporary pointer and true is returned. The function remove() takes in a char by reference and checks that the List isn't empty, if it is then false is returned and if not then the char passed in, “letter” is set to the char at the Node being removed and then if there is only one Node in the List clear() is called, otherwise gotoPrior() gets the prior Node, it's “next” pointer is set to the “next” of “cursor”, the Node “cursor” is at is deleted and “cursor” is set to the next Node. Finally, true is returned.

The List class overloaded assignment operator works exactly like the copy constructor except that it first checks that both the Lists aren't the same List and that it returns “this” dereferenced. The overloaded insertion operator which is a friend of the List class takes in an ostream object and a List, both by reference, and returns an ostream object by reference and first checks that the List isn't empty. If it is then a new line is printed to the screen, otherwise an event controlled loop goes through the List and prints out each char to the screen, putting brackets around the char at “cursor”. Finally, the ostream object is returned.