**Spencer Davis**

**Analysis of Linked List**

A brief summary of how a linked list works and its methods.

**Assignment 4**

09-26-18

Dr. Ken Rouse

Spencer Davis

09-25-18

Data Structures

How the Linked List class works:

This class uses a variable called a node. The node can hold a string value and point to another point in memory where another node is stored. The linked list class will manage all of these nodes into a single list, much like an array. There are six methods that this class can use: isEmpty, getSize, insertAtStart, insertAtEnd, insertAtPos, and deleteAtPos. Each of these methods will be gone into in detail.

isEmpty:

The first method that is in the class is the isEmpty method which returns a boolean. It will check if the starting node is null and will return a true or false statement.

getSize:

All this method does is return the size variable as an integer.

insertAtStart:

This method is passed the value (string) that is going to be put into the list. This is done by creating a new node called nptr with the value that is passed to the method. There is an if statement to check if the list is empty by checking if the start node has nothing inside it. If there is nothing in the list, then the start node references the ntpr node and the end references the start node because it is the only element in the list. If the list isn’t empty the ntpr node is linked to point to the start node, and the start node now references the ntpr node. This method doesn’t return anything.

insertAtEnd:

When a new value (string) is going to be added to the end of the list, this method is called. The method receives the value in the parameters and creates a new node ntpr with the given value. The size is incremented right after this variable is created. Just like the insertAtStart method this method will check if the list is empty before continuing. An if statement is checking if the start node is null and if it is, it the start node will reference the nptr node that was created and the end will reference the start. If the list is not empty then the end not will now point to the ntpr node that has been created and the end will now reference the ntpr node since it is at the end. This method is a void so it returns nothing.

insertAtPos:

The insertAtPos method is used to insert a value (string) at a specific position in the list. It is passed two variables in the parameters, the value to be added and the position to add the value to. Three variables are created at the beginning of this method. The first is a node called nptr that now stores the value (string) that was passed to the method. The second is a node called ptr that references the start variable. This node is going to be used to traverse through the list. Lastly the position variable. This variable is reassigned because it must be one less that the position that is going to be inserted. This is because it needs to stop at the node that is going to be pointing to the new node. A loop then begins that runs less than the size of the list. At the end of the loop the ptr node references the node that it was pointing at. This jumps from node to node in the list. Inside the loop it is checking if the iteration of the loop is equal to the position desired. If the iteration is equal to the position then a temp node is created, and it references the node that ptr is pointing to. The ptr node is then set to point at the nptr node and the ntpr node is set to point at the temp node. Outside the for loop the size is incremented. This method is also a void, so it returns nothing.

deleteAtPos:

The delete at position method can be broken into three separate parts: removing the first position, removing the last position, and removing the position given to the method. The delete at position method is passed one variable in the parameters, the position to be removed. The first part of the delete at position method, removing the first position, is done by checking if the position is one. If the position is one, then the start node is set to reference the node that is was pointing to and the size is decremented. The second part of the delete at position is deleting the node at the end of the list. There is an if statement checking if the position is equal to the size of the list. If this is true then the following will happen. Two nodes will be created, s and t. Both will reference the start node. A loop is then started that runs while s is not referencing end. In the loop the t is set to s and s is set to the next node. The allows both nodes to traverse through the list and s will be the end and t will be right before the end. The end variable is the set to reference the node t, and what it is referencing is now null. The size is then decremented, and the method is exited. The final thing this method can do is to remove the position that was given. This starts by subtracting one from the position variable that was given for the same reason that is was done when adding something to a position. A node is created called ptr that references the start node, it will be used to traverse through the list. A loop is started very similar to the loop that was done in insertAtPos method. It checks if the iteration of the loop is equal to the position. If it is the same then a new temp node is created that references the node that ptr is pointing at. The temp node then references the node that it is pointing at, and ptr is set to that node. This removes the node completely from the list. The size of the list is then decremented.