Linked List Report

Dr. Rouse

Ancel Carson

9/26/2018

Overview

A linked list is a data structure that is used to collect data of an unknown quantity in a potentially non-linear way. A linked list does not store data in a sequence and does not allocate space it does not yet need. This contrasts with an array which has a set number of data points it can hold, stores its information in a sequence, and uses memory for spaces that not have been filled. The benefit of using a linked list is that when memory is limited, it does not use more of it than it absolutely needs allowing for faster and smoother programming

A linked list functions on an idea of nodes and references. A node can be thought of as a box and the references as directions to the next box. Inside each box is a specific piece of information such as a number or a phrase and the directions on how to get to the next box. The boxes can be stored in any order and any location in the computer’s memory, so the references are very important. Going from box to box is much like a scavenger hunt the computer must run every time a specific box is being looked for.

Program

The aLinkedList program is designed to add nodes to the list and update the references for all the nodes around it. This allows the list to not have a set size and add information in any order.

Variables

There are some key variables inside of the aLinkedList program are the two reference nodes and the size variable. The two reference nodes are start and end. These two nodes allow the computer to remember where the scavenger hunt starts and ends hence the naming. These two nodes are very important to keep track of, especially the start node. If the program loses the start node it does not know where to start looking and the information is lost, and all data is gone. The size variable keeps track of how long the list is. There are more variables that are used by specific methods and they will be talked about later.

aLinkedList()

Right below the variables is what is called the constructor. That is the part of the code that groups the start, end, and size variables together for another program to use them. Inside of the constructors the references are set to null and the size is set to 0. In terms of coding, both things mean that the variables are empty and waiting to be given information. It is a way of giving a clean slate to the program that will be using the aLinkedList.

isEmpty()

This method is used to ask the aLinkedList if there is anything in it. It asks if the start reference is still equal to null and returns true or false value depending on the answer. If it is still equal to null, the result is true. If there is a node being referenced by start, then the result is false.

getSize()

This method is used to return the length of the list to another program when it is called.

insertAtStart()

This method begins by taking in a string given by another program. It then creates a new node with a blank reference to hold the string. The size variable is then increased by 1 to coincide with the list increasing in size. If the created node is the first node to be added to the list both the start and end references are set to the new node. This is because as the only node in the list, it is both the first and last node. If the created node is not the first node, then its reference is set to the current starting node and the start reference is set to the created node. This allows nodes to be added to the front of the list without losing all the information behind it.

insertAtEnd()

This method also begins by taking in a string given by another program. It then creates a new node with a blank reference to hold the string. The size variable is then increased by 1 to coincide with the list increasing in size. If the created node is the first node to be added to the list both the start and end references are set to the new node. This is because as the only node in the list, it is both the first and last node. If the created node is not the first node, then the reference of the current ending node is set to the created node. This allows nodes to be added to the end of the list and for the linked list to keep track of the last node in series.

insertAtPos()

This method begins by taking in not only a string, but a position variable given by another program. It then creates two nodes, one that contains the string that was given and another that references the same node as the start reference. The position variable is then reduced by one. This is done so that when going down the list, the program stops at the node before the desired position of the created node. Once that has been done the program runs a loop a number of times equal to the size of the list. On each iteration, the number of times the loop has been run is compared to the desired position of the created node. If the numbers are not equal, the reference is shifted down one node until it reaches the node before the desired position. When the number of iterations is equal to the desired position, the created node is added to the list. To do this a temp node is created to hold the node the reference points to. After that, the reference is connected to the created node and the created node is connected to the temp node. This allows a node to be added in any location without losing any data in other nodes.

deleteAtPos()

This method can be broken into three parts, deleting the node at the start of the list, deleting the node at the end of a list, and deleting a node somewhere in the middle of the list. Each one getting more complicated than the one before it.

Beginning of the list

To delete a node at the start of the list, the program simply shifts the start reference down one node. Once the original start node is lost, it can be considered deleted as there is no way to get it back. After removing the node, the size is decreased, and the method is ended.

End of the list

To delete a node at the end of the list, the program starts by making two temporary nodes. Until they reach the end of the list the two temp nodes are shifted one after another down the list. The result is that temp node1 is referencing the last node and temp node2 is referencing the node just before the last node. Once they both have the correct node, temp node2 is set to be the end node and its reference is changed to null. This breaks the link between the previous end node and the current end node effectively deleting the previous end node. After completing this change of reference, the size is decreased, and the method is ended.

Middle of the list

To delete a node in the middle of the list, the program begins by creating a single reference node and setting the position value to the node directly before the desired node. Once that has been done the program runs a loop a number of times equal to the size of the list. On each iteration, the number of times the loop has been run is compared to the desired position of the created node. If the numbers are not equal, the reference is shifted down one node until it reaches the node before the desired position. When the number of iterations is equal to the desired position, the created node is deleted from the list. To do this a temp node is created to hold the node the reference points to. The temp node is then shifted down the list by 1 so the reference is before the desired node and the temp node is after the desired node. The reference node is then told to point to the temp node. This breaks the link to the desired node effectively deleting it. After deleting the chosen node, the size is decreased, and the method is ended.

Suggestions

* ptr and nptr are vague variable names. Would recommend finding more descriptive names to help understanding
* aLinkedList class does not have a function to retrieve data from anywhere in the list. Would recommend including a function that does so.