Author: Andressa Olindo Wu

Completion Date: 05/10/23

classname.cpp: node.cpp

Description of the role or purpose of objects of this class goes here.

Node class is designed to represent a single node in a linked list data structure. Each node in the linked list contains a unique identifier, redid, and some associated data, represented by the name string. Each node contains a pointer to the next node in the linked list, which is used to create a sequence of linked nodes that can be traversed in order.

Linked List class includes various methods to manipulate the list such as inserting, deleting, traversal and sorting nodes. The linked list is composed of node objects, which contain two member variables, redid and name, that hold an integer and a string respectively. The LinkedList class has a private pointer to the head and tail nodes of the list and an integer variable to store the length of the list.

PUBLIC MEMBER VARIABLES

data_type Variable_Name; - Purpose of the variable

int redid;
 Purpose: This variable is of type int and likely represents some unique identifier or key associated with a particular node in the linked list.

string name;
 Purpose: This variable is of type string and likely represents some data or information associated with the node in the linked list. It will be used to store student names.

Node *next;
 Purpose: This variable is a pointer to another Node object, representing the next node in the linked list. This pointer is used to create a chain of linked nodes that can be traversed in sequence.

CONSTRUCTORS

Brief description of the purpose of this constructor; for example, default constructor, copy constructor

Destructor if any.

The constructor used in the Node class is the default constructor.

Default Constructor:

Node(int redid, string name)

The constructor for the Node class takes two parameters: redid and name. These parameters are used to set the values of the redid and name member variables for each new Node object.

Default Constructor:

LinkedList(int redid = 0, string name = "")
This is a constructor for a LinkedList class that creates a new Linked List object. It takes two optional parameters; an integer redid and a string name.

Destructor:

~Linkedlist()

OTHER PUBLIC MEMBER FUNCTIONS

Function_Name: brief description of the purpose of this function ,it's input, and its output.

Also include a brief description of how the function works if the algorithm is non-trivial.

return_type Function_Name (parameter_list)

Precondition: State of Program before member function is run

Postcondition: State of Program after member function is run

Function_Name: printList()
Brief description: This function is used to print the linked list. It takes no input parameters and has no output.
Return type: void
Parameter list: none
Precondition: The linked list should have been properly initialized with valid data.

Postcondition: The function iterates through the linked list using a temporary pointer called 'temp', printing the values of each node in the list to the console until the end of the list is reached. The algorithm used is a simple while loop that checks if the temporary pointer is not null and then prints the redid and name of the current node before moving on to the next node using the 'next' pointer.

Function_Name: getHead()
Brief Description: This function prints the current head node of the linked list. Input: None
Output: None
Algorithm:
Check if the head node is null or not.

Function_Name: getTail()

Brief Description: This function is used to print the ID and name of the tail node of the linked list, if it exists.

Input: None

Output: None (prints the ID and name of the tail node if it exists)

Algorithm:
Check if the tail is null.If it is, print "Tail: nullptr" and return.

Function_Name: getLength
Return type: int
Parameter list: none
Brief description: This function returns the length of the linked list.

Input: This function doesn't take any input parameters.

Output: This function returns an integer value which represents the length of the linked list.

Function_Name: append()

Return type: void

Parameter list: none

Purpose: To add a new node with the given redid and name at the end of the linked list.

Input: Two parameters- an integer redid and a string name, which represent the data of the new node.

Output: None.

Algorithm:
Create a new node with the given redid and name.

Function_Name: deleteLast()

Return type: void

Parameters: None

Input: None
Output: None
Brief description: This function deletes the last node in the linked list.

How it works: The function first checks if the list is empty. If it is, it simply returns without doing anything. If the list has only one node, it sets the head and tail pointers to nullptr, effectively deleting the node. If the list has more than one node, it traverses the list using two pointers: temp and pre. Temp traverses the list until it reaches the last node, while pre stays one node behind. When temp reaches the last node, pre becomes the new tail node, and its next pointer is set to nullptr. The last node is then deleted, and the length of the list is decremented by 1.

Function Name: prepend()

Input:
int redid: an integer representing the unique identifier of the new node to be added to the beginning of the linked list.
string name: a string representing the name associated with the new node.

Output: none (void)
Brief Description: The prepend function adds a new node to the beginning of the linked list. If the list is empty, the new node becomes both the head and the tail of the list. Otherwise, the new node is added before the current head, and the head is updated to point to the new node. The length of the list is

incremented after the new node is added.
Algorithm:
Create a new node using the input redid and name.

Function_Name: deleteFirst()
Purpose: This function deletes the first node in the linked list.

Input: None
Output: None
Algorithm:
Check if the length of the list is 0, if it is then return.
Return Type: void
Parameter List: None
Precondition: The linked list may or may not have any nodes.
Postcondition: If the list was not empty, the first node of the list is removed and the length of the list is decremented by 1. If the list was empty, no changes were made to the list.

Function_Name: insert()

Purpose: This function inserts a new Node into the linked list at a given index with a given redid and name.

Input: Index (an integer representing the index at which to insert the new Node), redid (an integer representing the redid of the new Node), name (a string representing the name of the new Node).

Output: A boolean indicating whether or not the insertion was successful.

Algorithm:
• Check if the index is valid (i.e. between 0 and length inclusive). If not, return false.
Precondition: The linked list exists.
Postcondition: The linked list has a new Node inserted at the specified index with the specified redid and name.

Function_Name: deleteNode()
Purpose: This function deletes a node at a given index in a linked list.

Input: An integer representing the index of the node to be deleted.

Output: None.
Algorithm:
If the index is less than 0 or greater than or equal to the length of the linked list, return without doing anything.
If the index is 0, call deleteFirst() function and return.
Precondition: A linked list exists and has at least one node.
Postcondition: The node at the given index is removed from the linked list. If the index is invalid, the linked list remains unchanged.

Function_Name: sortByName()

Description: This function sorts the linked list by the names of the nodes in alphabetical order using the Selection Sort algorithm.

Input: None Output: None

Working: The function works by iterating over each node in the linked list (current pointer), and for each node, finding the minimum node among the remaining unsorted nodes (using the min and temp pointers). If the minimum node is not the current node, their name and redid fields are swapped. This effectively moves the minimum node to its correct position in the sorted list. The process repeats until all nodes have been sorted.

Precondition: The linked list should be initialized and have at least one Node object in it. The name and redid fields of each Node object should be properly initialized and comparable using the < operator.

Postcondition: The linked list is sorted in ascending order by the name field of each Node object. The name and redid fields of

each Node object are updated according to their new positions in the sorted list. The original order of the nodes in the list is lost. The function outputs a success message to the console.

Function_Name: reverse()
Purpose: The purpose of this function is to reverse the linked list.

Input: This function does not take any input parameters.

Output: This function does not return any value.

Precondition: The linked list exists and has at least one node.

Postcondition: The linked list is reversed.

Function_Name: sortByREDID()

Input: None

Output: Outputs a message indicating that the sorting was successful.

Brief Description: This function sorts a singly linked list of nodes by the redid field in ascending order using the selection sort algorithm.

Return type: void
Input parameters: None

Algorithm:
The outer loop iterates over each element in the array, from left to right.
Precondition: The program must have a valid linked list with at least one node. Each node must have a name and redid field.
Postcondition: The linked list is sorted in ascending order based on the redid field. No new nodes are added or removed from the linked list.

Main Function:

This is the main function for a linked list program. It provides a menu of options for the user to interact with the linked list, including creating a linked list, adding nodes to the beginning and end of the list, adding a node at a specific index, deleting nodes from the beginning and end of the list, deleting a node at a specific index, sorting the list by name or REDID, reversing the list, and exiting the program. The program uses an object of the LinkedList class to perform these operations. The user is prompted for input depending on the selected option. The program continues to prompt the user until they choose to exit.

Preconditions: None, but a linked list object must be created before any other options can be selected.

Postconditions: The program exits when the user chooses to exit. The state of the linked list object may be modified depending on the options selected by the user.