CSCI 2270 – Data Structures

Instructor: Asa Ashraf

Assignment 4

Due: Sunday, September 26 2021, 11:59PM

Linked List - Part 2

Communication Between Social Media Profiles

OBJECTIVES

1. Delete node(s), readjust a sublist, and detect loop in a linked list
2. Get practice implementing classes

This assignment is an extension of Assignment 3.

Background

In this assignment you're going to extend our SocialMediaNetwork model from Assignment 3. In particular, you will need to implement operations to: (i) delete a SocialMediaProfile node, (ii) delete the entire network, (iii) readjust the network, and (iv) both create and detect a loop.

Building your own communications network

You will be implementing a class to simulate a linear communication network between social media profiles. There are three files in Canvas containing a code skeleton to get you started. *Do not modify the header file or your code won't work in Canvas!* You will have to complete only the class implementation in SocialMediaNetwork.cpp . The driver file main.cpp is implemented for you.

The linked-list itself will be implemented using the following struct (already included in the header file):

```
struct SocialMediaProfile
{
    string name;                    // name of the person
    string socialMediaStatus;       // status set by the profile
    int numberMessages;             // no. of messages in the inbox
    SocialMediaProfile *next;       // pointer to the next profile
    int totalFriends;               // number of friends of this profile
};
```

CSCI 2270 – Data Structures (Fall 2021)

Instructor: Asa Ashraf

Class Specifications

The **SocialMediaNetwork** class definition is provided in the file *SocialMediaNetwork.hpp* in Canvas. *Do not modify this file or your code won't work on Coderunner!* Complete the TODOs in the file *SocialMediaNetwork.cpp* according to the following specifications.

SocialMediaProfile* head; *// private data member: same as Assignment 3*

- ➔ Points to the first node in the linked list

SocialMediaNetwork(); *// same as Assignment 3*

- ➔ Class constructor; set the head pointer to NULL

bool isEmpty(); *// same as Assignment 3*

- ➔ Return true if the head is NULL, false otherwise

void addProfileInfo(SocialMediaProfile* previous, string profileName, int totalRoom);
// same as Assignment 3

- ➔ Insert a new profile with name **profileName** and **totalRoom** in the linked list after the profile pointed to by **previous**.
- ➔ If **previous** is NULL, then add the new profile to the beginning of the list.
- ➔ Print the name of the profile you added according to the following format:

```
// If you are adding at the beginning use this:
cout << "adding: " << profileName << " (HEAD)" << endl;

// Otherwise use this:
cout << "adding: " << profileName << " (prev: " << previous->name <<
")" << endl;
```

void loadDefaultSetup(); *// modified from Assignment 3*

- ➔ First, delete whatever is in the list using the member function **deleteEntireNetwork**.
- ➔ Then add the following six profiles, in order, to the network with **addProfileInfo**: "Marshall", "Lily", "Ted", "Robin", "Barney", "Ranjit". Friend numbers are 10, 8, 6, 5, 4, 15 respectively. Statuses are respectively:
  - ◆ "There is a great sandwich place on the 24th."
  - ◆ "I want to move to the suburbs."
  - ◆ "Working on a building design right now."
  - ◆ "The Canucks won today."
  - ◆ "Suit up."
  - ◆ "NYC is an interesting city."

CSCI 2270 – Data Structures (Fall 2021)

Instructor: Asa Ashraf

SocialMediaProfile* searchForProfile(string profileName); *// same as Assignment 3*

➔ Return a pointer to the node with name **profileName**. If **profileName** cannot be found, return NULL

void printNetwork(); *// same as Assignment 3*

➔ Print the names of each node in the linked list. Below is an example of correct output using the default setup. (Note that you will **cout << "NULL"** at the end of the path)

```
== CURRENT PATH ==
Marshall(10) -> Lily(8) -> Ted(6) -> Robin(5) -> Barney(4) -> Ranjit(15)
-> NULL
===
```

➔ If the network is empty then print *"nothing in path"*

void deleteSocialMediaProfile(string profileName); *// Beware of edge cases*

➔ Traverse the list to find the node with name **profileName**, then delete it. If there is no node with name **profileName**, print *"Profile does not exist."*

void deleteEntireNetwork();

➔ If the list is empty, do nothing and return. Otherwise, delete every node in the linked list and set **head** to NULL. Print the name of each node as you are deleting it according to the following format:

```
cout << "deleting: " << node->name << endl;
```

After the entire linked list is deleted, print:

```
cout << "Deleted network" << endl;
```

void readjustNetwork(int startIndex, int endIndex);

➔ Manipulate **next** pointers to readjust the linked list. Here, ***startIndex*** and ***endIndex*** are respective indices of node(s), calculated from the first node (head) - denoting a chunk of the linked list. The function will send the chunk of the link list between start index and end index at the end of the linked list. Consider the node at head as index 0.

For example, if you have linked list like this: **"A -> B -> C -> D -> E-> NULL"**, and ***startIndex*=1 and *endIndex*=3**, then the linked list after readjustNetwork should be "**A -> E -> B -> C -> D-> NULL**", since the sublist "**B -> C -> D**" was extracted and moved towards the end.

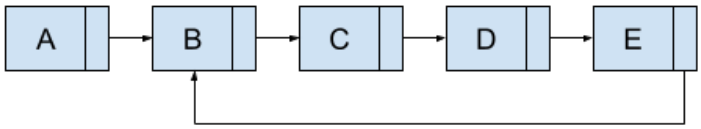
If you have linked list like this: `"A -> B -> C -> D -> NULL"`, and ***startIndex=0 and endIndex=2***, then the linked list after readjustNetwork should be "`D -> A -> B -> C -> NULL`". Here, "D" is the new head.

Validate against the following conditions in order:

- ➔ If the linked list is empty, print "*Linked List is Empty*".
- ➔ If ***endIndex*** is greater than or equal to the index of the last node of the linked list or smaller than ***0***, then print "*Invalid end index*". *(Can you reason why?)*
- ➔ Similarly, if ***startIndex*** is greater than or equal to the index of the last node of the linked list or smaller than ***0***, then print "*Invalid start index*".
- ➔ If ***startIndex*** **>** ***endIndex*** print "Invalid indices".
  [NOTE: Change the order of the "node" (by manipulating the next pointers of each node), not the "value of the node"]

bool detectLoop();

- ➔ Traverse through the linked list (pointed to by **head**) to detect the presence of a loop. A loop is present in the list when the tail node points to some intermediate node (possibly even itself) in the linked list, instead of pointing to NULL value. For example, in the following list with `"A"` at the head has a loop from tail (`"E"`) to node `"B"` :



  This means that the last unique node E is connected back to the node B that appears before it in the linked list, thus creating a non-terminating linked list.
- ➔ This function should return `true` if the list contains a loop, else return `false`.
- ➔ Refer to the following links for the algorithm of loop detection:
  https://www.youtube.com/watch?v=apIw0Opq5nk
  https://www.youtube.com/watch?v=MFOAbpfrJ8g

SocialMediaProfile* createLoop(string profileName);

- ➔ As a way to test the detectLoop() function, develop a createLoop() function that adds a loop to the linked list pointed to by **head**.
- ➔ You'll achieve this by creating a link from the last node in the list to an intermediate node. The function takes as argument the profile name of that intermediate node to loop back into.
- ➔ **IMPORTANT:** The function should return the last node of the linked list before creation of the loop. This will be needed by the driver function to break the loop.
- ➔ For example, consider the linked list: `"A -> B -> C -> D -> E -> NULL".` Suppose the function is called as: `createLoop("B");`

After execution of the function the linked list should be `"A -> B -> C -> D -> E -> B -> ..."` *(as shown in figure above)* and it will return a pointer to the node `E`. **NOTE:** node `E` was the last node before creation of the loop.

➔ If the profile is not present in the linked list, the function should return without creating a loop. A pointer to the last node should still be returned.

~SocialMediaNetwork();

➔ Class destructor; delete the entire list and set the head pointer to NULL.

➔ You should print the nodes' data as you delete them, similar to **deleteEntireNetwork()**.

TODOs

1. **void deleteSocialMediaProfile(string profileName)**
2. **void deleteEntireNetwork()**
3. **SocialMediaProfile * createLoop(string profileName)**
4. **bool detectLoop()**
5. **void readjustNetwork(int startIndex, int endIndex)**
6. **~SocialMediaNetwork();**

To test the entire implementation below, you will also need to plug-in your answers from Assignment 3 in the provided starter file.

Main driver file

The *main* driver file is already implemented for you. Your program will start by displaying a menu by calling the **displayMenu** function included in main.cpp. The user will select an option from the menu to decide what the program will do, after which, the menu will be displayed again. The specifics of each menu option are described below.

Option 1: Build Network

This option calls the **loadDefaultSetup** function, then calls the **printNetwork** function. You should get the following output:

```
deleting: Marshall
deleting: Lily
deleting: Ted
deleting: Robin
deleting: Barney
deleting: Ranjit
```

```
Deleted network
adding: Marshall (HEAD)
adding: Lily (prev: Marshall)
adding: Ted (prev: Lily)
adding: Robin (prev: Ted)
adding: Barney (prev: Robin)
adding: Ranjit (prev: Barney)
== CURRENT PATH ==
Marshall(10) -> Lily(8) -> Ted(6) -> Robin(5) -> Barney(4) -> Ranjit(15) ->
NULL
===
```

Option 2: Print Network Path

Calls the **printNetwork** function. Output should be in the format below:

```
// Output for the default setup
== CURRENT PATH ==
Marshall(10) -> Lily(8) -> Ted(6) -> Robin(5) -> Barney(4) -> Ranjit(15) ->
NULL
===

// Output when the linked list is empty
== CURRENT PATH ==
nothing in path
===
```

Option 3: Add Profile

Prompt the user for four inputs: the name of a new profile to add to the network, **newProfile**, the number of friends the new profile has, the social media status of the new profile, and the name of a profile already in the network, **previous**, which will precede the new profile. Use the member functions **searchForProfile** and **addProfileInfo** to insert **newProfile** into the linked-list right after **previous**.

- If the user wants to add the new profile to the head of the network then they should enter “First” instead of a previous profile name.
- If the user enters an invalid previous profile (not present in the linked list), then you need to prompt the user with the following error message and collect input again until they enter a valid previous profile name or “First”:

```
cout << "INVALID(previous profile name)...Please enter a VALID profile
name!" << endl;
```

- Once a valid previous profile name is passed and the new profile is added, call the function **printPath** to demonstrate the new linked-list.
- The Names of the profiles are case-sensitive.

For example, the following should be the output if the linked-list contains the default setup from option (1) and the user wants to add Kevin after Robin

```
Enter a new profile name:
Kevin
Enter number of friends:
8
Enter the social media status:
I love soccer!
Enter the previous profile name (or First):
Robin
adding: Kevin (prev: Robin)
== CURRENT PATH ==
Marshall(10) -> Lily(8) -> Ted(6) -> Robin(5) -> Kevin(8) -> Barney(4) ->
Ranjit(15) -> NULL
===
```

Option 4: Delete Profile

Prompt the user for a profile name, then pass that name to the **deleteSocialMediaProfile** function and call **printNetwork** to demonstrate the new linked-list.

For example, the following should be the output if the linked-list contains the default setup from option (1) and the user wants to delete Ted:

```
Enter a profile name:
Ted
== CURRENT PATH ==
Marshall(10) -> Lily(8) -> Robin(5) -> Barney(4) -> Ranjit(15) -> NULL
===
```

Option 5: Create and Detect loop in network

Call the **createLoop** and **detectLoop** functions.

User will be prompted to enter the name of the profile to loop back. **createLoop** function will be called to create the loop accordingly. After that **detectLoop** will be called. Depending on the status of loop creation **detectLoop** will return either true (if the loop is created) or false (if the

loop could not be created). After calling **createLoop** function, If there is a loop it will be broken by the driver (refer to the starter code for more detail). So, in this operation, a loop is created in the linked list (if appropriate input is given) and it is removed immediately.

```
#> 5
Enter the profile name to loop back:
Robin
Network contains a loop
Breaking the loop
```

Option 6: Re-adjust Network

Call the **readjustNetwork** function, then the **printNetwork** function. User should be prompted to input the start index and end index.

For example, the following should be the output if the linked-list contains the default setup from option (1):

```
#> 6
Enter the start index:
1
Enter the end index:
2
== CURRENT PATH ==
Marshall(10) -> Robin(5) -> Barney(4) -> Ranjit(15) -> Lily(8) -> Ted(6) ->
NULL
===
```

Option 7: Clear network

Call the **deleteEntireNetwork** function. For example, deleting the default network should print:

```
Network before deletion
== CURRENT PATH ==
Marshall(10) -> Lily(8) -> Ted(6) -> Robin(5) -> Barney(4) -> Ranjit(15) ->
NULL
===
deleting: Marshall
deleting: Lily
deleting: Ted
deleting: Robin
```

```
deleting: Barney
deleting: Ranjit
Deleted network
Network after deletion
== CURRENT PATH ==
nothing in path
===
```

Option 8: Quit

Print the following message:

```
cout << "Quitting... cleaning up path: " << endl;
```

Then proceed to exit the program. Prior to the exit, the destructor will be invoked automatically, and print the profiles' names as they are deleted, same as **deleteEntireNetwork**.

```
Quitting... cleaning up path:
== CURRENT PATH ==
Marshall(10) -> Lily(8) -> Ted(6) -> Robin(5) -> Barney(4) -> Ranjit(15) ->
NULL
===
deleting: Marshall
deleting: Lily
deleting: Ted
deleting: Robin
deleting: Barney
deleting: Ranjit
Deleted network
```