CSCI-2270
Assignment 3
Instructor: Boese

Linked Lists

Communication Between Towers

Objectives

- Create, traverse, add, and delete to/from a linked-list

Background:

Communication between towers

In the Lord of the Rings trilogy, there is a scene where the first beacon is lit in the towers of Minas Tirith. The second beacon then sees the fire, and knows to light its fire to send a signal to the third beacon, and so forth. This was a means of communicating in the days before telegraphs were invented as it was much faster than sending a human rider to deliver a message. Communication towers were equipped with signaling mechanisms, such as mirrors, that could spell out messages using the positions of the mirrors.

Today, there are several examples of communication networks that are conceptually similar, but much more technically advanced, that route messages through multiple hubs between the sender and the receiver. For example, when you type a URL into a web browser, a request is sent through a network of service providers to the destination, and then packets of information are sent back to your machine. If I type www.google.com from my home in Boulder, my request follows this path:

```
1 192.168.2.1 (192.168.2.1)
2 c-24-9-60-1.hsd1.co.comcast.net (24.9.60.1)
3 te-9-7-ur02.boulder.co.denver.comcast.net
4 xe-13-3-1-0-ar01.aurora.co.denver.comcast.net
5 he-3-10-0-0-cr01.denver.co.ibone.comcast.net (68.86.92.25)
te-1-1-0-4-cr01.chicago.il.ibone.comcast.net (68.86.95.205)
6 xe-2-0-0-0-pe01.910fifteenth.co.ibone.comcast.net (68.86.82.2)
7 as15169-1-c.910fifteenth.co.ibone.comcast.net (23.30.206.106)
8 72.14.234.57 (72.14.234.57)
9 209.85.251.111 (209.85.251.111)
10 den03s06-in-f16.1e100.net (74.125.225.208)
```

Each IP address is a hop in the network for my request, which is received at each service provider and then forwarded to the next service provider in the network, depending on the final destination of the message.

(Note: I got this path by typing traceroute www.google.com in a terminal window. From campus, you will see a different path.)

Build your own communications network

In this assignment, you're going to simulate a communications network using a linked list. Each node in your linked list will represent a city and you need to be able to send a message between nodes from one side of the country to the other. Your program also needs to provide the capability to update the network by adding cities and still be able to transmit the message.
(Note: I'll refer to the linked list as the network throughout this document.)

Include the following cities in your network:

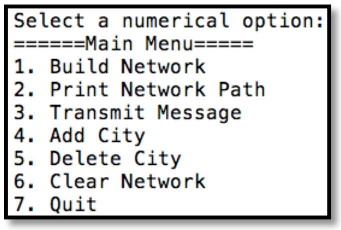
Los Angeles
Phoenix
Denver
Dallas
Atlanta
New York

Implement each city as a `struct` with *(use the starter code)*

- a name,
- a pointer connecting it to the next city in the network, and
- a place to store the message being sent. *(The message is a string.)*
- an integer for the number of messages that have been sent through this city

When you initially build your network, the order of the cities should be the same as the order listed above. After the network is built, you will provide the option of adding additional cities.

First, display a menu
When your program starts, you should display a menu that presents the user with options for how to run your program. The menu needs to look like the one shown here:



```
Select a numerical option:
======Main Menu=====
1. Build Network
2. Print Network Path
3. Transmit Message
4. Add City
5. Delete City
6. Clear Network
7. Quit
```

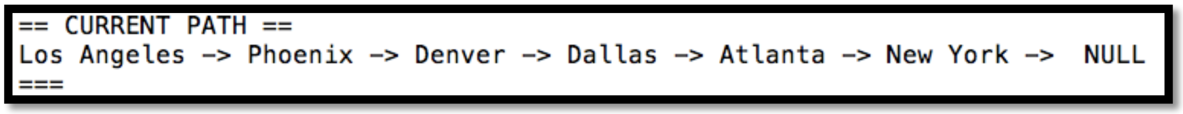
The user will select the number for the menu option and your program should respond accordingly to that number. Your menu options need to have the following functionality.

1. Build Network: This option builds the linked list using the cities listed above in the order they are listed. Each city needs to have a name, a pointer to the next city, and a message value, which will initially be an empty string. This option should be selected first to build the network, and can be selected anytime the user wants to rebuild the starting network after adding cities. As part of the Build Network functionality, you should print the name of each city in the network once the network is built in the following format by calling the function for #2:

Los Angeles -> Phoenix -> Denver -> Dallas -> Atlanta -> New York -> NULL

2. Print Network Path: This option prints out the linked list in order from the head to the tail by following the next pointer for each city. You should print the name of each city. The function could be very useful to you when debugging your code. The format should be the same as shown above.
There is one space before and after each arrow which is a dash and a >

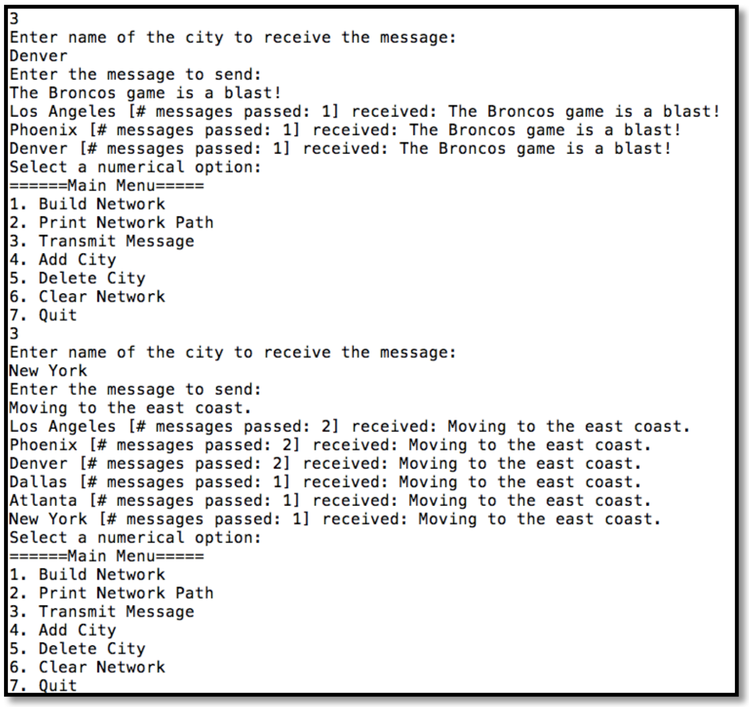
Here is a screenshot showing the format that is expected:



```
== CURRENT PATH ==
Los Angeles -> Phoenix -> Denver -> Dallas -> Atlanta -> New York ->  NULL
===
```

3. Transmit Message: This option reads in from the user the message to transmit, and transmits the message starting at the beginning of the network and ending at the city specified. Each city in which the message passes through updates their numberMessages variable to account for the number of messages passed through that city.

For example:



```
3
Enter name of the city to receive the message:
Denver
Enter the message to send:
The Broncos game is a blast!
Los Angeles [# messages passed: 1] received: The Broncos game is a blast!
Phoenix [# messages passed: 1] received: The Broncos game is a blast!
Denver [# messages passed: 1] received: The Broncos game is a blast!
Select a numerical option:
======Main Menu=====
1. Build Network
2. Print Network Path
3. Transmit Message
4. Add City
5. Delete City
6. Clear Network
7. Quit
3
Enter name of the city to receive the message:
New York
Enter the message to send:
Moving to the east coast.
Los Angeles [# messages passed: 2] received: Moving to the east coast.
Phoenix [# messages passed: 2] received: Moving to the east coast.
Denver [# messages passed: 2] received: Moving to the east coast.
Dallas [# messages passed: 1] received: Moving to the east coast.
Atlanta [# messages passed: 1] received: Moving to the east coast.
New York [# messages passed: 1] received: Moving to the east coast.
Select a numerical option:
======Main Menu=====
1. Build Network
2. Print Network Path
3. Transmit Message
4. Add City
5. Delete City
6. Clear Network
7. Quit
```

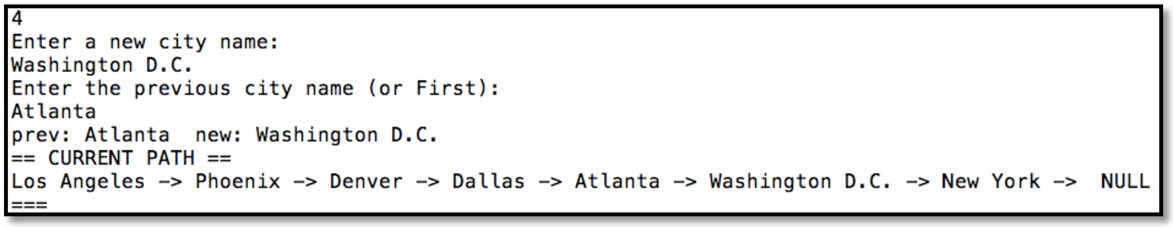
4. <u>Add City</u>: This option allows the user to add a new city to the network. If the user selects this option, then they should be prompted for the name of the city and the city that the new city should follow in the network. For example, if the user wants to add Tucson after Phoenix in the network, then the first four cities in the network would be:

Los Angeles -> Phoenix -> Tucson -> Denver…

If the user wants to add a new city to the head of the network, e.g. replace Los Angeles as the starting city, then they should type First when prompted for the previous city and your code should handle this special case.

Modify your `loadDefaultSetup()` function to use `addCity()` internally.

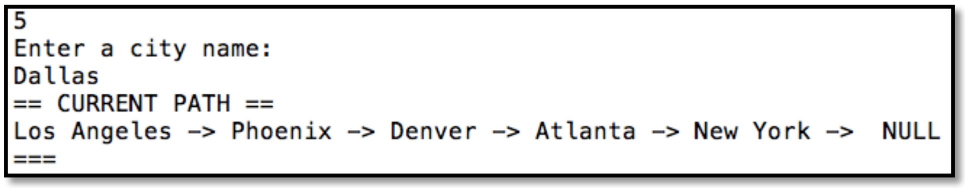
Here is a screenshot showing the expected output for the add city functionality when the user selects Add City from the menu.



```
4
Enter a new city name:
Washington D.C.
Enter the previous city name (or First):
Atlanta
prev: Atlanta  new: Washington D.C.
== CURRENT PATH ==
Los Angeles -> Phoenix -> Denver -> Dallas -> Atlanta -> Washington D.C. -> New York ->  NULL
===
```

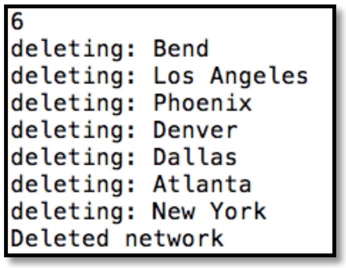
Figure 1: Example output (if the item is inserted into the first position in the list, do not print the line beginning with `prev`*.*

5. Delete City: Delete one of the cities in the network. If the user selects this option, then they should be prompted for the name of the city to delete.



```
5
Enter a city name:
Dallas
== CURRENT PATH ==
Los Angeles -> Phoenix -> Denver -> Atlanta -> New York ->  NULL
===
```

6. Clear Network: Deletes all cities in the network. Inside the `deleteEntireNetwork` function, as each node is deleted, print the name of the city being deleted.



```
6
deleting: Bend
deleting: Los Angeles
deleting: Phoenix
deleting: Denver
deleting: Dallas
deleting: Atlanta
deleting: New York
Deleted network
```

7. Quit: This option allows the user to exit the program.

For each of the options presented (except Quit), after the user makes their choice and your code runs for that option, you should re-display the menu to allow the user to select another option.

Program Specifications

- Use the starter code in the following section, making sure not to modify sections that are marked "DO NOT MODIFY"
- Do not add any extra '#include' statements. You may only use the include statements provided in the starter-code
- Your code needs to be readable, efficient, and accomplish the task provided.
- Make sure your code is commented enough to describe what it is doing. Include a comment block at the top of the .cpp file with your name, assignment number, and course instructor, and anyone you worked with.
- You must fill in the functions as specified. You do not need any additional functions.
- Your functions must use the exact output format specified above. Where appropriate, example output code has been provided in the functions.

Starter Code

```
// ==========================================
// File: Assignment 3 starter code
// Author: Matt Bubernak
// Date: 1/29/2015
// Modified: Fall 2016 E.S.Boese
//           Fall 2017 W.Temple
// Description: Linked List Fun
// ==========================================

#include <iostream>
#include <fstream>
#include <cstdlib>
#include <string>

using namespace std;

// DO NOT MODIFY THIS STRUCT
struct city
{
    string  name;           // name of the city
    city    *next;          // pointer to the next city
    int     numberMessages; // how many messages passed through this city
    string  message;        // message we are sending accross
};

/* Function prototypes */

city* addCity(city *head, city *previous, string cityName );
city* deleteCity(city *head, string cityName);
city* deleteEntireNetwork(city *head);
city* searchNetwork(city *head, string cityName);
city* loadDefaultSetup(city *head);
void transmitMsg(city *head, string receiver, string filename);
void printPath(city *head);
void displayMenu();
city* handleUserInput(city *head);

/* Do NOT modify main function */
int main(int argc, char* argv[])
{
    // pointer to the head of our network of cities
    city *head = NULL;
    head = handleUserInput(head);
    printPath(head);
    head = deleteEntireNetwork(head);
    if (head == NULL)
        cout << "Path cleaned" << endl;
    else
        printPath(head);

    cout << "Goodbye!" << endl;
    return 0;
}

/*
 * Purpose: handle the interaction with the user
 * @param head the start of the linked list
 * @return the start of the linked list
 *
 * DO NOT MODIFY THIS FUNCTION
 */
city* handleUserInput(city *head)
{
    bool quit = false;
    string s_input;
    int input;

    // loop until the user quits
    while (!quit)
    {
        displayMenu();

        // read in input, assuming a number comes in
        getline(cin, s_input);
        input = stoi(s_input);

        switch (input)
        {
            // print all nodes
            case 1:     //rebuild network
                head = loadDefaultSetup(head);
                printPath(head);
                break;

            case 2:     // print path
                printPath(head);
                break;

            case 3: //message is read in from file
              {
```

```
        string cityReceiver;
        cout << "Enter name of the city to receive the message: "
            << endl;
        getline(cin, cityReceiver);

        cout << "Enter the message to send: " << endl;
        string message;
        getline(cin, message);

        transmitMsg(head, cityReceiver, message);
      }
        break;

    case 4:
      {
        string newCityName;
        string prevCityName;
        cout << "Enter a new city name: " << endl;
        getline(cin, newCityName);

        cout << "Enter the previous city name (or First): " << endl;
        getline(cin, prevCityName);

        // find the node containing prevCity
        city *tmp = NULL;
        if(prevCityName !="First")
            tmp = searchNetwork(head, prevCityName);
        // add the new node
        head = addCity(head, tmp, newCityName);
        printPath(head);
      }
        break;

    case 5:     // delete city
      {
        string city;
        cout << "Enter a city name: " << endl;
        getline(cin, city);
        head = deleteCity(head, city);
        printPath(head);
      }
        break;

    case 6:     // delete network
        head = deleteEntireNetwork(head);
        break;

    case 7:     // quit
```

```
                quit = true;
                cout << "Quitting... cleaning up path: " << endl;
                break;

            default:    // invalid input
                cout << "Invalid Input" << endl;
                break;
        }
    }
    return head;
}

/*
 * Purpose: Add a new city to the network
 *   between the city *previous and the city that follows it in the network.
 * Prints: `prev: <city name>  new: <city name>` when a city is added,
 *   prints _nothing_ if the city is being added to the _first_ position in
 *   the list.
 * @param head pointer to start of the list
 * @param previous name of the city that comes before the new city
 * @param cityName name of the new city
 * @return pointer to first node in list
 */
city* addCity(city *head, city *previous, string cityName )
{

        cout << "prev: " << previous->name << "  new: " << cityName << endl;

    return head;
}


/*
 * Purpose: Search the network for the specified city and return a pointer
 * to that node
 * @param ptr head of the list
 * @param cityName name of the city to look for in network
 * @return pointer to node of cityName, or NULL if not found
 * @see addCity, deleteCity
 */
city *searchNetwork(city *ptr, string cityName)
{

    return ptr;
}

/*
 * Purpose: deletes all cities in the network starting at the head city.
 * @param ptr head of list
 * @return NULL as the list is empty
 */
city* deleteEntireNetwork(city *ptr)
{


        cout << "deleting: " << ptr->name << endl;



    cout << "Deleted network" << endl;
    // return head, which should be NULL
    return ptr;
}

/*
 * Purpose: transmit a message from city to city
 * @param head pointer to head of the list
 * @param receiver the name of the City to receive the message
 * @param message the message to transmit*/
void transmitMsg(city *head, string receiver, string message)
{
    if(head == NULL)
    {
        cout << "Empty list" << endl;
        return;
    }




        cout << sender->name << " [# messages passed: " <<
             sender->numberMessages << "] received: " <<
             sender->message << endl;
}


/*
 * Purpose: delete the city in the network with the specified name.
 * @param head first node in list
 * @param cityName name of the city to delete in the network
 * @return head node of list
 */
city* deleteCity(city *head, string cityName)
{
    // if the city dosn't exist, nothing we can do.
    //      use this output statement
        cout << "City does not exist." << endl;



    return head;
}


/*
 * Purpose: prints the current list nicely
 * @param ptr head of list
 */
void printPath(city *ptr)
{
    cout << "== CURRENT PATH ==" << endl;

    // If the head is NULL
    if (ptr == NULL)
        cout << "nothing in path" << endl;

    // Add code here to print the network path.




    cout << "===" << endl;
}

/*
 * Purpose: populates the network with the predetermined cities
 * @param head start of list
 * @return head of list
 */
city* loadDefaultSetup(city *head)
{
    head = deleteEntireNetwork(head);
    head = addCity(head,NULL,"Los Angeles");

    // Add code here to populate the LinkedList with the default values

    return head;
}

/* Purpose: displays a menu with options
 * DO NOT MODIFY THIS FUNCTION
 */
```

```
void displayMenu()
{
    cout << "Select a numerical option:" << endl;
    cout << "======Main Menu=====" << endl;
    cout << "1. Build Network" << endl;
    cout << "2. Print Network Path" << endl;
    cout << "3. Transmit Message" << endl;
    cout << "4. Add City" << endl;
    cout << "5. Delete City" << endl;
    cout << "6. Clear Network" << endl;
    cout << "7. Quit" << endl;
}
```

Submitting Your Code:

Log into Moodle and go to the Homework 3 link. It is set up in the quiz format. Follow the instructions on each question to submit all or parts of each assignment question. You can check your solution to each question by clicking on the “Check” button. Note that you can only submit your homework once.

Note: there is no late period on assignments! If you miss the deadline or do not do well, you can sign up for an optional grading interview to get up to half the points missed back. There is also an optional extra credit assignment at the end of the semester you can use to replace one of your homework scores.