CSCI 2270 – Data Structures

Instructors: Zagrodzki, Ashraf, Trivedi

Assignment 3

Due Sunday, February 9 2020, 11:59 PM

Array Doubling and Linked Lists

OBJECTIVES

1. Append to an array (Array doubling)
2. Create, traverse, add nodes to a linked list
3. Get practice implementing classes

1. Array Doubling

Append to an array

In this problem, you will emulate the **push_back** function of the C++ **vector** class. We will call this function **append**. This function will insert an element to an array at the smallest vacant index. In case the array is full, the function should perform array doubling and then insert the element to the new array.

Unlike the **resize** function (Recitation 3 exercise), here you will not be returning a pointer after doubling the array. Instead, a reference-to-array pointer will be passed to your function **append**, so that you can modify the pointer to the array itself. The function should return `true` if array doubling was performed, otherwise return `false`.

Use the function prototype provided below:

```
bool append(string* &str_arr, string s, int &numEntries, int &arraySize);
```

INPUT PARAMETERS:

- ➔ **str_arr** is an array of type string in which you insert elements. A reference to this array pointer is passed to your function.
- ➔ **s** is a new string that you want to insert in your string array

- ➔ **numEntries** keeps track of the number of elements that have been inserted in your array so far
- ➔ **arraySize** variable stores the current size of your array

OUTPUT PARAMETERS:

- ➔ **doubled** is just a boolean value true or false
    - ◆ You return true if array has been doubled else return false

You are also required to update the variable **numEntries** and **arraySize** *within* the function:

- ➔ Update **numEntries** when you add a new element to the array.
- ➔ Update **arraySize** whenever you perform array doubling.

---

2. Linked Lists

Communication Between Towers

Background

In the Lord of the Rings trilogy, there is a scene where a warning beacon is lit in the towers of Minas Tirith, which is seen by a second beacon, prompting them to light their own fire which a third beacon sees, and so forth. This was a rapid means of communication in the days before telegraphs were invented.

In this problem, you're going to model and simulate a similar communications network using a linked list. Each node in the list will represent a country and you need to be able to send a message between nodes from one side of the world to the other.

Building your own communications network

You will be implementing a class to simulate a linear communication network between countries. There are three files on Moodle containing a code skeleton to get you started. You will have to complete the TODOs in both the class implementation in ***CountryNetwork.cpp*** and the driver file ***main.cpp***. ***Do not modify the header file or your code won't work in Moodle!***

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

```
struct Country
{
    string name;          // name of the country
    string message;       // message this country has received
    int numberMessages;   // no. of messages passed through this country
    Country *next;        // pointer to the next country
};
```

Class Specifications

The **CountryNetwork** class definition is provided in the file *CountryNetwork.hpp* in Moodle. ***Do not modify this file or your code won't work on Moodle!*** Fill in the file *CountryNetwork.cpp* according to the following specifications.

Country* head;

- ➔ Points to the first node in the linked list

CountryNetwork();

- ➔ Class constructor; set the head pointer to NULL

void insertCountry(Country* previous, string countryName); ***// Beware of edge cases***

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

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

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

void loadDefaultSetup();

- ➔ Add the following six countries, in order, to the network with **insertCountry**: "United States", "Canada", "Brazil", "India", "China", "Australia"

Country* searchNetwork(string countryName);

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

void transmitMsg(string receiver, string msg);

- ➔ Traverse the linked list from the head to the node with name **receiver**. For each node in this path (including the head), set the node's **message** to **msg** and increment the node's **numberMessages** field. If the list is empty, print "Empty list" and exit the function. If the node is not present, print "Country not found".
- ➔ As you traverse the list, at each node report the message received and the number of messages received using the following cout: (See the end of this document for example output)

```
cout << node->name << " [# messages received: " <<
node->numberMessages << "] received: " << node->message << endl;
```

void printPath();

- ➔ 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 ==
United States -> Canada -> Brazil -> India -> China -> Australia -> NULL
===
```

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

Main driver file

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 **printPath** function. You should get the following output:

```
adding: United States (HEAD)
adding: Canada (prev: United States)
adding: Brazil (prev: Canada)
adding: India (prev: Brazil)
adding: China (prev: India)
adding: Australia (prev: China)
== CURRENT PATH ==
United States -> Canada -> Brazil -> India -> China -> Australia -> NULL
===
```

Option 2: Print Network Path

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

```
// Output for the default setup
== CURRENT PATH ==
United States -> Canada -> Brazil -> India -> China -> Australia -> NULL
===

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

Option 3: Transmit Message

Prompt the user for two inputs: a message to send, and the name of a country to receive the message *(Hint: use **getline** in case there are spaces in the user input)*. Pass the message and country name to the **transmitMsg** function. *Don't forget to add a newline after the message is collected, and before the output is printed. This is done for better readability.*

For example, the following should be the output if the linked-list contains the default setup from option (1) and the message “bom dia” is sent to “Brazil”:

```
Example 1:
Enter name of the country to receive the message:
Brazil
Enter the message to send:
bom dia

United States [# messages received: 1] received: bom dia
Canada [# messages received: 1] received: bom dia
Brazil [# messages received: 1] received: bom dia
```

If the user then decides to transmit the message “ni hao” to “China”, the output will be:

```
Example 2:
Enter name of the country to receive the message:
China
Enter the message to send:
ni hao

United States [# messages received: 2] received: ni hao
Canada [# messages received: 2] received: ni hao
Brazil [# messages received: 2] received: ni hao
India [# messages received: 1] received: ni hao
China [# messages received: 1] received: ni hao
```

If the user then decides to transmit the message “Sushi” to “Japan”, the output when the country is not present will be:

```
Example 3:
Enter name of the country to receive the message:
Japan
Enter the message to send:
Sushi

Country not found
```

Option 4: Add Country

Prompt the user for two inputs: the name of a new country to add to the network, **newCountry**, and the name of a country already in the network, **previous**, which will precede the new country. Use the member functions **searchNetwork** and **insertCountry** to insert **newCountry** into the linked-list right after **previous**.

- If the user wants to add the new country to the head of the network then they should enter "First" instead of a previous country name.
- If the user enters an invalid previous country (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 country name or "First":

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

- Once a valid previous country name is passed and the new country is added, call the function **printPath** to demonstrate the new linked-list.
- First letter of the country to be added should be **Uppercase** (sentence case) e.g. If you want to add "mexico, it should be stored as "**Mexico**" in 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 add Colombia after Brazil:

```
Enter a new country name:
Colombia
Enter the previous country name (or First):
Brazil
adding: Colombia (prev: Brazil)
== CURRENT PATH ==
United States -> Canada -> Brazil -> Colombia -> India -> China -> Australia ->
NULL
===
```

Option 5: Quit

Print the following message:

```
cout << "Quitting..." << endl;
```

Finally, print the following before exiting the program:

```
cout << "Goodbye!" << endl;
```

Example run

```
Select a numerical option:
+=====Main Menu=========+
 1. Build Network
 2. Print Network Path
 3. Transmit Message
 4. Add Country
 5. Quit
+-----------------------+
#> 2
== CURRENT PATH ==
nothing in path
===
Select a numerical option:
+=====Main Menu=========+
 1. Build Network
 2. Print Network Path
 3. Transmit Message
 4. Add Country
 5. Quit
+-----------------------+
#> 3
Enter name of the country to receive the message:
Japan
Enter the message to send:
Cherry Blossom

Empty list
Select a numerical option:
+=====Main Menu=========+
 1. Build Network
 2. Print Network Path
 3. Transmit Message
 4. Add Country
 5. Quit
+-----------------------+
#> 1
adding: prev: [HEAD]
adding: United States (HEAD)
adding: Canada (prev: United States)
adding: Brazil (prev: Canada)
adding: India (prev: Brazil)
adding: China (prev: India)
adding: Australia (prev: China)
== CURRENT PATH ==
United States -> Canada -> Brazil -> India -> China -> Australia -> NULL
===
Select a numerical option:
+=====Main Menu=========+
 1. Build Network
 2. Print Network Path
 3. Transmit Message
 4. Add Country
 5. Quit
+-----------------------+
#> 2
== CURRENT PATH ==
United States -> Canada -> Brazil -> India -> China -> Australia -> NULL
===
Select a numerical option:
+=====Main Menu=========+
 1. Build Network
 2. Print Network Path
 3. Transmit Message
 4. Add Country
 5. Quit
+-----------------------+
#> 3
Enter name of the country to receive the message:
Japan
Enter the message to send:
```

```
Sushi

Country not found
Select a numerical option:
+=====Main Menu=========+
 1. Build Network
 2. Print Network Path
 3. Transmit Message
 4. Add Country
 5. Quit
+-----------------------+
#> 4
Enter a new country name:
Japan
Enter the previous country name (or First):
United States
adding: Japan (prev: United States)
== CURRENT PATH ==
United States -> Japan -> Canada -> Brazil -> India -> China -> Australia
-> NULL
===
Select a numerical option:
+=====Main Menu=========+
 1. Build Network
 2. Print Network Path
 3. Transmit Message
 4. Add Country
 5. Quit
+-----------------------+
#> 3
Enter name of the country to receive the message:
Australia
Enter the message to send:
Kangaroo

United States [# messages received: 1] received: Kangaroo
Japan [# messages received: 1] received: Kangaroo
Canada [# messages received: 1] received: Kangaroo
Brazil [# messages received: 1] received: Kangaroo
India [# messages received: 1] received: Kangaroo
```

```
China [# messages received: 1] received: Kangaroo
Australia [# messages received: 1] received: Kangaroo
Select a numerical option:
+=====Main Menu=========+
 1. Build Network
 2. Print Network Path
 3. Transmit Message
 4. Add Country
 5. Quit
+-----------------------+
#> 3
Enter name of the country to receive the message:
Brazil
Enter the message to send:
Football

United States [# messages received: 2] received: Football
Japan [# messages received: 2] received: Football
Canada [# messages received: 2] received: Football
Brazil [# messages received: 2] received: Football
Select a numerical option:
+=====Main Menu=========+
 1. Build Network
 2. Print Network Path
 3. Transmit Message
 4. Add Country
 5. Quit
+-----------------------+
#> 4
Enter a new country name:
Singapore
Enter the previous country name (or First):
First
adding: Singapore (HEAD)
== CURRENT PATH ==
Singapore -> United States -> Japan -> Canada -> Brazil -> India -> China
-> Australia -> NULL
===
Select a numerical option:
+=====Main Menu=========+
 1. Build Network
 2. Print Network Path
 3. Transmit Message
 4. Add Country
 5. Quit
+-----------------------+
#> 4
Enter a new country name:
Australia
Enter the previous country name (or First):
Spain
INVALID(previous country name)...Please enter a VALID previous country
name!
Canada
adding: Australia (prev: Canada)
== CURRENT PATH ==
Singapore -> United States -> Japan -> Canada -> Australia -> Brazil ->
India -> China -> Australia -> NULL
===
Select a numerical option:
+=====Main Menu=========+
 1. Build Network
 2. Print Network Path
 3. Transmit Message
 4. Add Country
 5. Quit
+-----------------------+
#> 5
Quitting... cleaning up path:
== CURRENT PATH ==
Singapore -> United States -> Japan -> Canada -> Australia -> Brazil ->
India -> China -> Australia -> NULL
===
Goodbye!
```

Appendix:

- **CountryNetwork::insertCountry()**
  - `cout << "adding: " << countryName << " (HEAD)" << endl;`
  - `cout << "adding: " << countryName << " (prev: " << previous->name << ")" << endl;`

- **CountryNetwork::transmitMsg()**
  - `cout << "Empty list" << endl;`
  - `cout << "Country not found" << endl;`
  - `cout << sender->name << " [# messages received: " << sender->numberMessages << "] received: " << sender->message << endl;`

- **CountryNetwork::printPath()**
  - `cout << "== CURRENT PATH ==" << endl;`
  - `cout << "nothing in path" << endl;`
  - `cout << ptr->name << " -> ";`
  - `cout << ptr->name << " -> ";`
  - `cout << "NULL" << endl;`
  - `cout << "===" << endl;`

- **main()**
  - `cout << "Enter name of the country to receive the message: "<< endl;`
  - `cout << "Enter the message to send: " << endl;`
  - `cout << endl;`
  - `cout << "Enter a new country name: " << endl;`
  - `cout << "Enter the previous country name (or First): " << endl;`
  - `cout << "INVALID(previous country name)...Please enter a VALID previous country name!" << endl;`
  - `cout << "Quitting..." << endl;`
  - `cout << "Invalid Input" << endl;`
  - `cout << "Goodbye!" << endl;`

- **displayMenu()**
  - `cout << endl;`
  - `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 Country " << endl;`
- `cout << " 5. Quit " << endl;`
- `cout << "+-----------------------+" << endl;`
- `cout << "#> ";`