Notes on Assignment 3

CSCI 2270: Data Structures
Section: 202
TA: Sanskar Katiyar

Problem 1: Append to an array

In this problem, you will be look at another way of performing array doubling. Previously, we accomplished array doubling in main() [*Assignment 2*] as well as by returning a pointer [*Recitation 3*].

Before we discuss anything else, let's look at the function signature:

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

The writeup describes the role of each of these parameters and how you should handle them. However, the thing I want to draw your attention to is the how the array is being passed.

We are passing a reference-to-the-array pointer to this function. ***But why is that?***

Assume we didn't pass a reference but just the array pointer (`string* str_arr`). Recall that `str_arr` pointer points to the address of the first element of the array. By using the dereference operator (*), we can modify the elements of the array.

However, we cannot modify the address that this array pointer points to. A pointer is a variable and the value it stores is a memory address. When you pass just the pointer variable (`string* str_arr`), think of it as passing-by-value. The function will make a copy of the address that the pointer points to, but it will not allow you to change this. So, when you want to point str_arr to the new doubled array, this will result in a segmentation fault.

When you are passing the reference to the pointer, you are passing the value held by the pointer by-reference (in a way). This means that you have the authority to change this address within the function. And this is exactly what we do when we perform array doubling within this function.

Common Mistakes

1. **return**: Remember to return true when you perform array doubling and false when you don't perform array doubling.

2. **Update arraySize, numEntries**: Notice that arraySize and numEntries are being passed-by-reference. So, make sure you update these variables accordingly before you return.

3. **Don't forget to insert the element**: After performing array doubling, do not return from the function prematurely. Insert the element before you return.

Problem 2: Basic Operations on Linked List

In this problem, we will modularize our work and solve each of the following functions before we write the main program:

`CountryNetwork();`
`void printPath();`
`Country* searchNetwork(string countryName);`
`void insertCountry(Country* previous, string countryName);`
`void loadDefaultSetup();`
`void transmitMsg(string receiver, string msg);`

CountryNetwork()

This is the constructor of our class. Look at the CountryNetwork.hpp file to grasp what data members each Country node (implemented using a struct) contains.

The constructor is a special function which is automatically called whenever a new object of the class is created. As per the writeup, you are just required to set the head pointer to NULL, since we will always start with an empty list.

printPath()

Look for `traversal.cpp` in code and slides [*Recitation 4*] and understand how we implemented it.

You will also need to check if the list is empty. How? (Use the head pointer)

searchNetwork(string countryName)

We worked (or will work) on a search feature in Recitation 4 by modifying the `traversal.cpp` file. Go through the `search.cpp` file to understand how we find the exact node.

Remember: Once found, you should return the pointer to the node, else if there are no more nodes left, you should return NULL.

insertCountry(Country previous, string countryName)*

In order to implement this, you will need to utilize the previous pointer which is given to you as an argument.

As discussed in the writeup, there are two cases for insertion here:

1. **previous == NULL**
   This implies that the list is currently empty and thus the node you will be inserting will be the head of this list.

   Refer to Recitation 4 Slides: 24-27 to understand step-wise instructions.

2. **previous != NULL**
   This is the case of any other valid node.

   Refer to Recitation 4 Slides: 29-34 to understand step-wise instructions

You should also refer to the insertion.cpp file for case-by-case implementation. You may also use the function provided to you in the Recitation 4 exercise and modify this to suit the problem.

Remember to return the pointer of the node that you inserted and initialize the other data members accordingly.

loadDefaultSetup()

Use `insertCountry`, `searchNetwork` functions together to implement this function. You already know the exact order and the countries that you need to add.

`insertCountry` can use `searchNetwork` for getting the previous pointer. This will allow you to maintain the order of the list. Think of the order of insertions.

Another way to solve this is to insert the country in reverse order by changing the HEAD. You will be setting the previous as NULL, in this case.

transmitMsg(string receiver, string msg)

How will you find the receiver node pointer? (Use `searchNetwork`)

You should modify the code for the traversal to implement this function successfully.

Restrict your traversal to `tmp` in [HEAD, receiver node] portion of the list. Within each iteration of this traversal loop, modify the `tmp.message` and increment `tmp.numberMessages` data members.

main()

Now that you have all the pieces together, you just need to put it all together.

Follow the instructions in the writeup, you will most likely need to use a `switch` statement and call the helper functions.

Use only `getline` for inputs and print the exact statements as mentioned in the writeups. Coderunner may truncate spaces/tabs during the run so make sure there aren't any unnecessary spaces/tabs.