# CSCI 2270 – Data Structures

*Instructors: Zagrodzki, Ashraf*

Assignment 3

Due Sunday, September 20 - 2020, 11:59PM

## 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);
```

**<u>INPUT PARAMETERS</u>:**

➔ **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 Buildings

## Background

In this problem you're going to model a communication network among buildings in CU Boulder using a linked list. Each node in the list will represent a building and you need to be able to send a message between nodes from one side of the campus to the other.

## Building your own communications network

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

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

Assignment 3

Due Sunday, September 13 - 2020, 11:59PM

```
struct CUBuilding
{
    string name;          // name of the building
    string message;       // message this building has received
    int numberMessages;   // no. of messages passed through this building
    CUBuilding *next;     // pointer to the next building
    int totalRoom; // number of room in this building
};
```

## Class Specifications

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

**CUBuilding* head;**

- ➔ Points to the first node in the linked list

**CUBuildingNetwork();**

- ➔ Class constructor; set the head pointer to NULL

**void addBuildingInfo(CUBuilding* previous, string buildingName, int totalRoom);** ***// Beware of edge cases***

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

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

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

**void loadDefaultSetup();**

➔ Add the following six buildings along with their total room numbers, in order, with **addBuildingInfo**: "FLMG", "DLC", "ECOT", "CASE", "AERO", "RGNT". Room numbers are 2, 10, 6, 5, 4, 9 respectively.

**CUBuilding* searchForBuilding(string buildingName);**

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

**void transmitRoomInfo(string receiver);**

➔ 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 **"available room at" +<buildingName>+ " is " + <totalNumOfRoom>** and increment the node's **numberMessages** field. If the list is empty, print "Empty list" and exit the function. If the receiver node is not present, print "Building 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;
```

**For a complete example refer to the section Main driver file.**

**void printNetwork();**

➔ 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 ==
FLMG(2) -> DLC(10) -> ECOT(6) -> CASE(5) -> AERO(4) -> RGNT(9) -> 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.

Kindly note main.cpp is provided. **You do not have to code the driver.** This document is here for reference.

### Option 1: Build Network

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

```
adding: FLMG (HEAD)
adding: DLC (prev: FLMG)
adding: ECOT (prev: DLC)
adding: CASE (prev: ECOT)
adding: AERO (prev: CASE)
adding: RGNT (prev: AERO)
== CURRENT PATH ==
FLMG(2) -> DLC(10) -> ECOT(6) -> CASE(5) -> AERO(4) -> RGNT(9) -> NULL
===
```

### Option 2: Print Network Path

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

```
// Output for the default setup
== CURRENT PATH ==
FLMG(2) -> DLC(10) -> ECOT(6) -> CASE(5) -> AERO(4) -> RGNT(9) -> NULL
===

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

### Option 3: Broadcast Room Info

Prompt the user for one input: The name of a building till  the message is broadcasted. Pass the building name to the **transmitRoomInfo** function. *Don't forget to add a newline after the receiver is collected, and before the output is printed. This is done for better readability.* Each node should create its message with its name and totalRoom:

➔ **"available room at " +<buildingName>+ " is " + <totalNumOfRoom>**

Kindly maintain the **numberMessages** for the node accordingly.

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

```
Example 1:
Enter name of the recipient to receive the message:
AERO

FLMG [# messages received: 1] received: available room at FLMG is 2
DLC [# messages received: 1] received: available room at DLC is 10
ECOT [# messages received: 1] received: available room at ECOT is 6
CASE [# messages received: 1] received: available room at CASE is 5
AERO [# messages received: 1] received: available room at AERO is 4
```

If the user then decides to call the function again with "RGNT" as reciever :

```
Example 2:
Enter name of the recipient to receive the message:
RGNT

FLMG [# messages received: 2] received: available room at FLMG is 2
DLC [# messages received: 2] received: available room at DLC is 10
ECOT [# messages received: 2] received: available room at ECOT is 6
CASE [# messages received: 2] received: available room at CASE is 5
AERO [# messages received: 2] received: available room at AERO is 4
RGNT [# messages received: 1] received: available room at RGNT is 9
```

If the user then decides to broadcast room info till “HUMN”, the output when the building is not present will be:

```
Example 3:
Enter name of the recipient to receive the message:
HUMN

Building not found
```

### Option 4: Add Building

Prompt the user for three inputs: the name of a new building to add to the network, **newBuilding**, the number of rooms in the new building and the name of a building already in the network, **previous**, which will precede the new building. Use the member functions **searchForBuilding** and **addBuildingInfo** to insert **newBuilding** into the linked-list right after **previous**.

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

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

- Once a valid previous building name is passed and the new building is added, call the function **printPath** to demonstrate the new linked-list.

- The Names of the buildings 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 ADEN after CASE:

```
Enter a new building name:
ADEN
Enter total room number:
5
Enter the previous building name (or First):
CASE
adding: ADEN (prev: CASE)
== CURRENT PATH ==
```

```
FLMG(2) -> DLC(10) -> ECOT(6) -> CASE(5) -> ADEN(5) -> AERO(4) -> RGNT(9) -> 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. Broadcast Room Info
 4. Add Building
 5. Quit
+-----------------------+
#> 2
== CURRENT PATH ==
nothing in path
===
Select a numerical option:
+=====Main Menu=========+
 1. Build Network
 2. Print Network Path
 3. Broadcast Room Info
 4. Add Building
 5. Quit
```

```
+-----------------------+
#> 3
Enter name of the recipient to receive the message:
ADEN

Empty list
Select a numerical option:
+=====Main Menu=========+
1. Build Network
 2. Print Network Path
 3. Broadcast Room Info
 4. Add Building
 5. Quit
+-----------------------+
#> 1
adding: FLMG (HEAD)
adding: DLC (prev: FLMG)
adding: ECOT (prev: DLC)
adding: CASE (prev: ECOT)
adding: AERO (prev: CASE)
adding: RGNT (prev: AERO)
== CURRENT PATH ==
FLMG(2) -> DLC(10) -> ECOT(6) -> CASE(5) -> AERO(4) -> RGNT(9) -> NULL
===
Select a numerical option:
+=====Main Menu=========+
 1. Build Network
 2. Print Network Path
 3. Broadcast Room Info
 4. Add Building
 5. Quit
+-----------------------+
#> 2
== CURRENT PATH ==
FLMG(2) -> DLC(10) -> ECOT(6) -> CASE(5) -> AERO(4) -> RGNT(9) -> NULL
===
Select a numerical option:
+=====Main Menu=========+
1. Build Network
 2. Print Network Path
 3. Broadcast Room Info
 4. Add Building
 5. Quit
+-----------------------+
#> 3
Enter name of the recipient to receive the message:
ADEN

Building not found
Select a numerical option:
+=====Main Menu=========+
 Build Network
 2. Print Network Path
 3. Broadcast Room Info
 4. Add Building
 5. Quit
+-----------------------+
#> 4
Enter a new building name:
ADEN
Enter total room number:
6
Enter the previous building name (or First):
FLMG
adding: ADEN (prev: FLMG)
== CURRENT PATH ==
FLMG(2) -> ADEN(6) -> DLC(10) -> ECOT(6) -> CASE(5) -> AERO(4) -> RGNT(9)
-> NULL
===

Select a numerical option:
+=====Main Menu=========+
 1. Build Network
 2. Print Network Path
 3. Broadcast Room Info
 4. Add Building
 5. Quit
+-----------------------+
#> 3
Enter name of the recipient to receive the message:
```

```
RGNT

FLMG [# messages received: 1] received: available room at FLMG is 2
ADEN [# messages received: 1] received: available room at ADEN is 6
DLC [# messages received: 1] received: available room at DLC is 10
ECOT [# messages received: 1] received: available room at ECOT is 6
CASE [# messages received: 1] received: available room at CASE is 5
AERO [# messages received: 1] received: available room at AERO is 4
RGNT [# messages received: 1] received: available room at RGNT is 9
Select a numerical option:
+=====Main Menu=========+
 1. Build Network
 2. Print Network Path
 3. Broadcast Room Info
 4. Add Building
 5. Quit
+-----------------------+
#> 3
Enter name of the recipient to receive the message:
ECOT

FLMG [# messages received: 2] received: available room at FLMG is 2
ADEN [# messages received: 2] received: available room at ADEN is 6
DLC [# messages received: 2] received: available room at DLC is 10
ECOT [# messages received: 2] received: available room at ECOT is 6
Select a numerical option:
+=====Main Menu=========+
 1. Build Network
 2. Print Network Path
 3. Broadcast Room Info
 4. Add Building
 5. Quit
+-----------------------+
#> 4
Enter a new building name:
CHEM
Enter total room number:
10
Enter the previous building name (or First):
First
```

```
adding: CHEM (HEAD)
== CURRENT PATH ==
CHEM(10) -> FLMG(2) -> ADEN(6) -> DLC(10) -> ECOT(6) -> CASE(5) -> AERO(4)
-> RGNT(9) -> NULL
===
Select a numerical option:
+=====Main Menu=========+
 1. Build Network
 2. Print Network Path
 3. Broadcast Room Info
 4. Add Building
 5. Quit
+-----------------------+
#> 4
Enter a new building name:
ECON
Enter total room number:
2
Enter the previous building name (or First):
DUAN
INVALID(previous building name)...Please enter a VALID building name!
ECOT
adding: ECON (prev: ECOT)
== CURRENT PATH ==
CHEM(10) -> FLMG(2) -> ADEN(6) -> DLC(10) -> ECOT(6) -> ECON(2) -> CASE(5)
-> AERO(4) -> RGNT(9) -> NULL
===
Select a numerical option:
+=====Main Menu=========+
 1. Build Network
 2. Print Network Path
 3. Broadcast Room Info
 4. Add Building
 5. Quit
+-----------------------+
#> 5
Quitting...
Goodbye!
```

Assignment 3

Due Sunday, September 13 - 2020, 11:59PM

**Appendix:**

- **BuildingNetwork::addBuildingInfo()**
  - `cout << "adding: " << buildingName << " (HEAD)" << endl;`
  - `cout << "adding: " << buildingName << " (prev: " << previous->name << ")" << endl;`

- **BuildingNetwork::transmitRoomInfo(string receiver)**
  - `cout << "Empty list" << endl;`
  - `cout << "Building not found" << endl;`
  - `cout << sender->name << ‟ [# messages received: " << sender->numberMessages << "] received: " << sender->message << endl;`

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

- **main()**
  - `cout << "Enter name of the building to receive the message: "<< endl;`
  - `cout << endl;`
  - `cout << "Enter a new building name: " << endl;`
  - `cout << "Enter total room number: " << endl;`
  - `cout << "Enter the previous building name (or First): " << endl;`
  - `cout << "INVALID(previous building name)...Please enter a VALID building 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. Broadcast Room Info  " << endl;`
- `cout << " 4. Add Building " << endl;`
- `cout << " 5. Quit " << endl;`
- `cout << "+-----------------------+" << endl;`
- `cout << "#> ";`