Name: Phuong Hoang Tran
Id: 010796179
Date: 02/12/2024

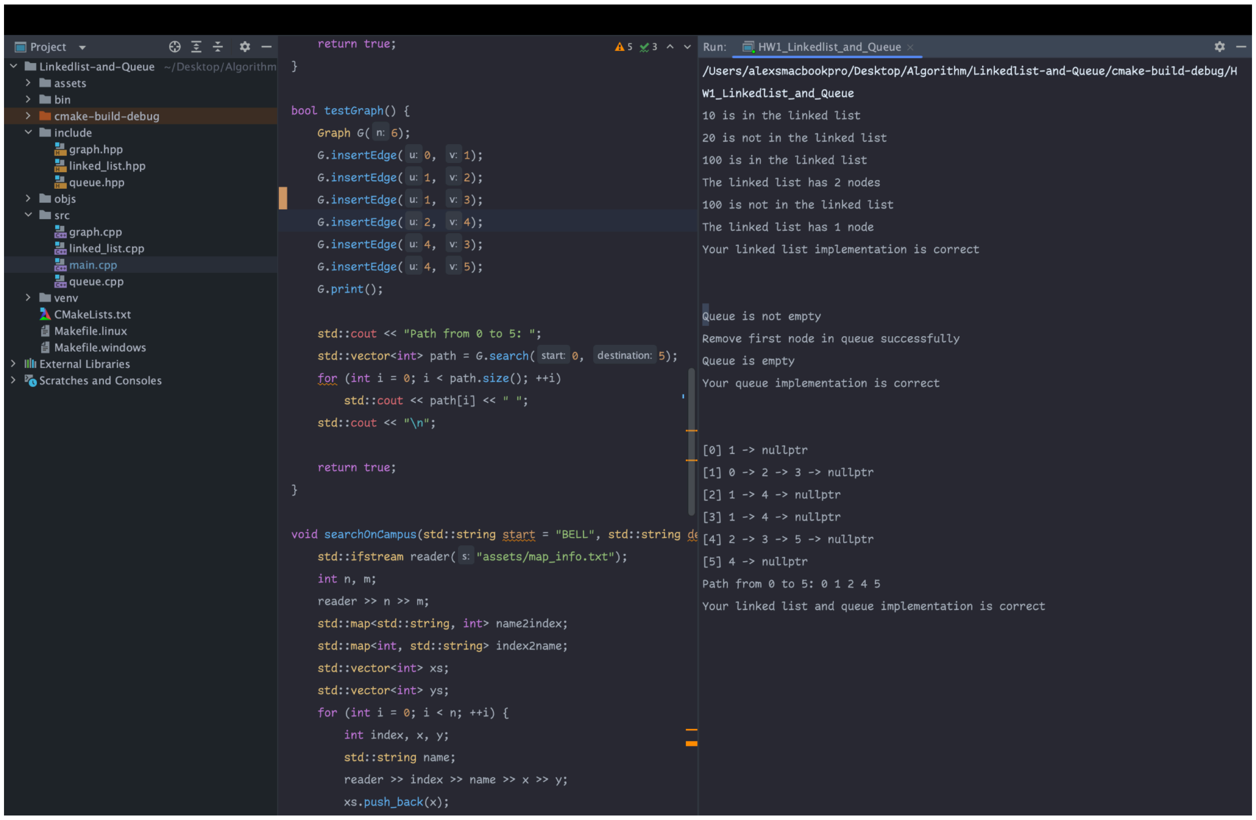
LinkedList and Queue

Summary
In this assignment we learn how to implement a linked list and queue data structure using individual nodes.

Details
To implement a linked list or a queue, we first start by creating a node class. The Node class contains 2 pointers and a value as member variable. The node class constructor takes in the value of that node, a next pointer which points to the next node and previous pointers which point to the previous node in the list or queue. The Node class is the building block of our linked list and queue because each individual element in a LinkedList or Queue is a Node. The LinkedList class contains 2 member variables; a root pointer which points to the beginning of the list and a tail pointer which points to the tail of the list. The LinkedList class methods are insert, find which locate and return the pointer of a node of value, remove which delete a node from memory and return the root pointer of a linked list. The LinkedList class also contains some helper methods for diagnosis which includes; print which displays the entire list and printBackward which print the entire list backward which help us ensure that our tail pointer for each node is correctly set. The Queue class, similar to LinkedList class; however differs in the way we remove value from the collection. The Queue class removes the value based on First in First out or FIFO so our remove method is different from our LinkedList implementation.

Design Decision
Let's start with the linked list implementation. The addition of a tail pointer helps us insert value in constant time, because without a tail pointer, to insert a value to the end of a list we must iterate through every time in the list before getting to the end of a list. The size method in the LinkedList implementation iterates through the entire list to get the size. Although the size method is not optimal, this helps us reduce complexity by not needing to keep track of a variable and update it accordingly. If efficiency becomes an issue, I will refactor and optimize this portion of the code. The remove method in my LinkedList implementation always returns the root pointer, this helps us determine if our linked list is empty without needing to call another function. Initially I considered returning a boolean value which validates that a value has been successfully removed, however after thinking about it.

Conclusion
The code works as expected and all test cases pass. The Queue test cases are not as extensive as I like, the test failed to validate if the Queue will work properly if more than one value gets inserted into the list.



```
Project
Linkedlist-and-Queue ~/Desktop/Algorithm
  assets
  bin
  cmake-build-debug
  include
    graph.hpp
    linked_list.hpp
    queue.hpp
  objs
  src
    graph.cpp
    linked_list.cpp
    main.cpp
    queue.cpp
  venv
  CMakeLists.txt
  Makefile.linux
  Makefile.windows
External Libraries
Scratches and Consoles
```

```cpp
    return true;
}

bool testGraph() {
    Graph G( n: 6);
    G.insertEdge( u: 0,  v: 1);
    G.insertEdge( u: 1,  v: 2);
    G.insertEdge( u: 1,  v: 3);
    G.insertEdge( u: 2,  v: 4);
    G.insertEdge( u: 4,  v: 3);
    G.insertEdge( u: 4,  v: 5);
    G.print();

    std::cout << "Path from 0 to 5: ";
    std::vector<int> path = G.search( start: 0,  destination: 5);
    for (int i = 0; i < path.size(); ++i)
        std::cout << path[i] << " ";
    std::cout << "\n";

    return true;
}

void searchOnCampus(std::string start = "BELL", std::string de
    std::ifstream reader( s: "assets/map_info.txt");
    int n, m;
    reader >> n >> m;
    std::map<std::string, int> name2index;
    std::map<int, std::string> index2name;
    std::vector<int> xs;
    std::vector<int> ys;
    for (int i = 0; i < n; ++i) {
        int index, x, y;
        std::string name;
        reader >> index >> name >> x >> y;
        xs.push_back(x);
```

Run: HW1_Linkedlist_and_Queue

```
/Users/alexsmacbookpro/Desktop/Algorithm/Linkedlist-and-Queue/cmake-build-debug/H
W1_Linkedlist_and_Queue
10 is in the linked list
20 is not in the linked list
100 is in the linked list
The linked list has 2 nodes
100 is not in the linked list
The linked list has 1 node
Your linked list implementation is correct


Queue is not empty
Remove first node in queue successfully
Queue is empty
Your queue implementation is correct


[0] 1 -> nullptr
[1] 0 -> 2 -> 3 -> nullptr
[2] 1 -> 4 -> nullptr
[3] 1 -> 4 -> nullptr
[4] 2 -> 3 -> 5 -> nullptr
[5] 4 -> nullptr
Path from 0 to 5: 0 1 2 4 5
Your linked list and queue implementation is correct
```