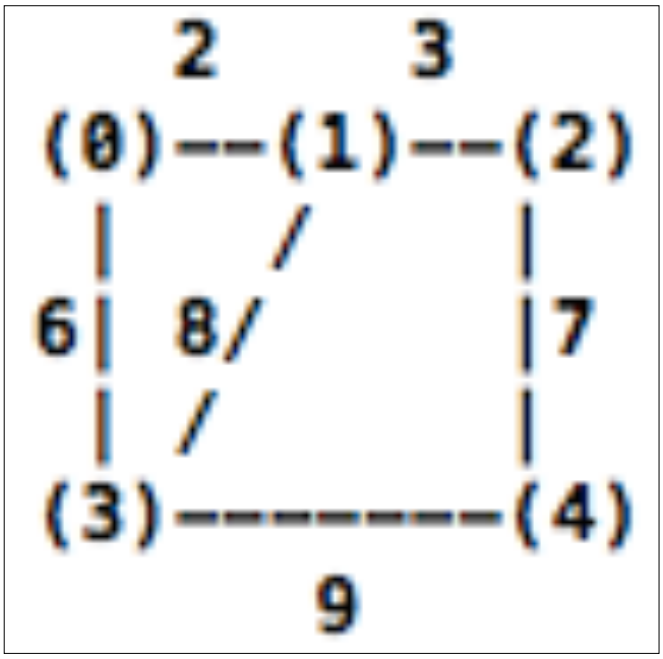


Midterm Project – Final Project

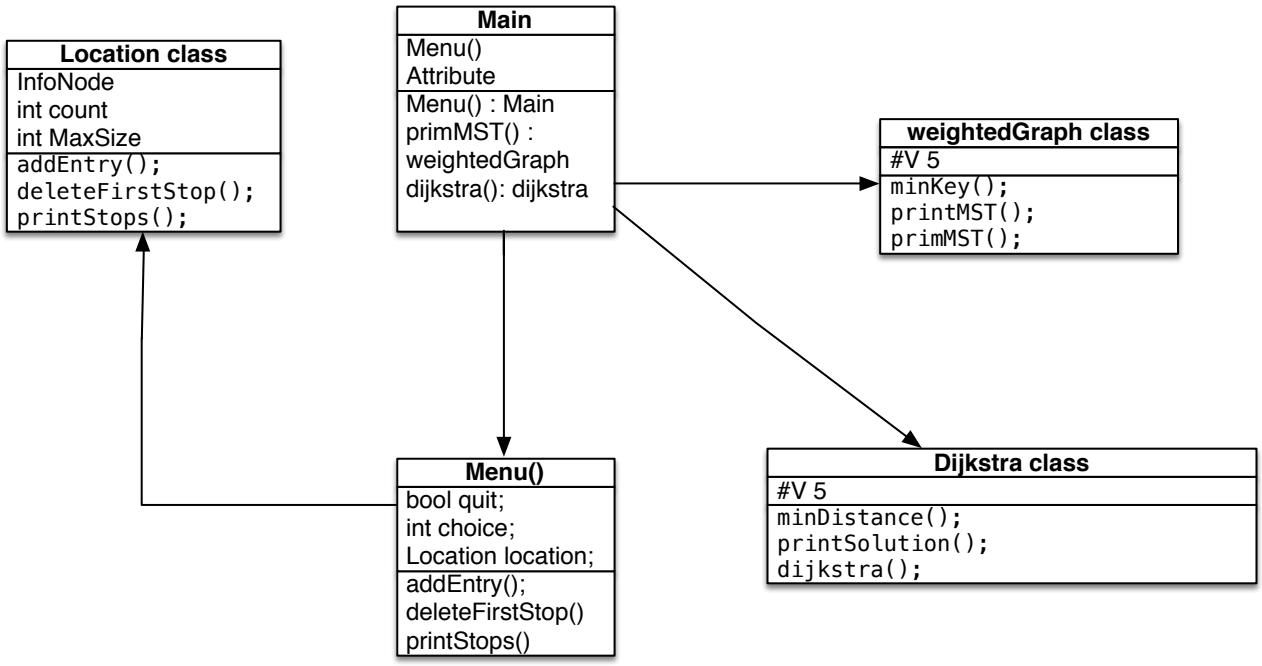
Prepared for: Dr. Mark Lehr
Prepared by: Joel Bateman
December 13, 2014

Objective

My objective in making this piece of software was to create a piece of reusable code that could be used as a plugin for some version of location software. This plugin will be able to store a list of selected locations within a data structure. It will also be able to take those selected locations and create a minimum spanning tree from them as well as a short distance from one point to all connecting points upon the graph.

.

Project Outline



This UML shows the major logic of the pluggin application where execution begins in the main. From the main the user is sent to a switch controlled Menu that contains a Location class object. In the Location class the “Stop Stack” [List of selected locations] is filled and managed.

- Execution begins in the main class
- The Location class contains both a custom stack to maintain the integrity of when the locations were entered in. This class also contains a list to hold the selected locations to give the program more to do with the data for example; iterate, print particular node without manipulating the lis

SUMMARY

```
================================
          MENU
================================
1) Add Stop
2) Delete Stop
3) View Stops
4) MST & Short Distance
choice:
```

Execution begins in the Main, which sends the user to a switch controlled Menu within the Menu function. The Menu function contains 4 options 1) -> Add a stop the “Stop Stack” 2) -> Delete Latest entry from “Stop Stack” 3) -> View the entries that have been added in to the “Stop Stack” | Stop Stack has a limit of 5 entries. This is because I implemented a Minimum Spanning Tree and Shortest Distance algorithm. To uses these algorithms I had to make sure the data was within an array of a set size, in this instance 5 was the chosen value. I did not find the program very easy to program but it was not terribly difficult either. I struggled most with design and what I actually wanted to build.

```
================================
           MENU
================================
1) Add Stop
2) Delete Stop
3) View Stops
4) MST & Short Distance
choice: 1
Select Location
   2    3
(0)--(1)--(2)
 |    /    |
6| 8/     |7
 | /      |
(3)-------(4)
0 -> Home
1 -> Chevron
2 -> Vons
3 -> Wingstop
4 -> Winco
0
Stop Added
```

At this point the user is out of the Main and into the Menu. The Menu function contains indicators of the locations 1) -> “Home” 2) -> ”Chevron” 3) -> ”Vons” 4) -> “Wingstop” 5) -> “Winco” | Stop Stack has a limit of 5 entries. Once the stack is full the user is sent to the MST and short Distance portion of the program.

Project size: 700 lines

Variable Amount: 30

Method Amount: 15

I built this program to save selected locations from the user into a Stack. I chose a Stack as the data structure to maintain the integrity that the user enters in their selected location for possible reuse later. I also have the data simultaneously being added into a List to quickly print the contents of both the Stack and List using an iterator. This portion of the application fulfills the requirements of STL library as well as list, stack and iterator.

```
else if(temp!=NULL){
    while(temp!=NULL){
        string selectedStops = "";
        switch(temp->ID)
        {
            case 0:selectedStops = "Home";
                preBuiltLocations.push_back(selectedStops);
                break;
            case 1: selectedStops = "Chevron";
                preBuiltLocations.push_back(selectedStops);
                break;
            case 2: selectedStops = "Vons";
                preBuiltLocations.push_back(selectedStops);
                break;
            case 3: selectedStops = "Wingstop";
                preBuiltLocations.push_back(selectedStops);
                break;
            case 4: selectedStops = "Winco";
                preBuiltLocations.push_back(selectedStops);
                break;
            default: cout<<"Not in favorites location spots"<<endl;
                break;
        }
        temp=temp->next;
    }
    for(list<string>::iterator it=preBuiltLocations.begin();it!=preBuiltLocatior
        ();it++){
        cout<<count++<<") "<<*it<<endl;
    }
}
cout<<"==============="<<endl;
cout<<"Stops Printed"<<endl;
preBuiltLocations.clear();
```

```
================================
     Minimum Spanning Tree
================================
Start       End            Distance
Home(0)     Chevron(1)     2
Chevron(1)  Vons(2)        3
Home(0)     Wingstop(3)    6
Chevron(1)  Winco(4)       5
================================
       SHORTEST DISTANCE
================================
Enter X (Starting Location)
ID   Location
0 -> Home
1 -> Chevron
2 -> Vons
3 -> Wingstop
4 -> Winco
Location ID: 0
   2    3
(X)--(1)--(2)
 |   /    |
6| 8/     |7
 | /      |
(3)-------(4)
      9
Vertex         Distance from Source
Home           0
Chevron        2
Vons           5
Wingstop       6
Winco          7
Would you like to exit Distance Calculator?(y/n)
```

This is the part of the application that calls on the MST and Shortest Distance algorithms. The program by default displays the minimum spanning tree. The user is then prompted for a starting point to trace the shortest distance. The selected starting point is marked with an (X) to give reference on the graph and then the user is prompted to repeat the program or exit.

Added Functionality

```
================================
             MENU
================================
1) Add Stop
2) Delete Stop
3) View Stops
4) MST & Short Distance
5) Show Current Location
choice: 5
================================
       CURRENT LOCATION
================================
   2    3
(X)--(1)--(2)
 |    /    |
6| 8/     |7
 | /      |
(3)-------(4)
      9
```

I added in a Show Current Locations case into the Main Menu switch. At the start of the application the User is prompted to select their Current Location(This part could be changed in the future to read in current location from a GPS) this user input is purely for demonstration purposes. For the entire runtime of the application instance, the entered current location is stored and can be viewed by selecting 5)→Show Current Location from the Menu. Current Location also prints a picture of the graph with an (X) to mark the current location.

```
HashTable<InfoNode *>* Location::getLocations() const{
    return locations;
}

bool Location::addNode(int ID, InfoNode *infoNode){
    locations->put(ID, infoNode);
    return true;
}

bool Location::deleteNode(int ID){
    locations->remove(ID);
    return true;
}

InfoNode* Location::findLocationID(int ID){
    InfoNode* infoNode=locations->get(ID);
    if(infoNode!=NULL){
        return infoNode;
    }
    else{
        cout<<"Not Found";
        return NULL;
    }
}
```

I added in methods to add, delete and find the ID using Hashing. Inside the Location.h, I added a private member hash table of infoNodes called locations. This hash table is store the ID member of the InfoNode class using hashing. This could be used for security if this plugin were to be used with another portion of the code. My goal was to make that private member of locations to hold the data of “Stop Stack” and well as “Current Location” and that way no outside classes would have access to the data outside of the plugin.

I also tried my best to fix any bugs and memory leaks that took place in the application as a part of this Final update.