1. This is a continuation of the previous lab. Download the following files and create a workspace using them:

<https://markbowman.org/231/Lab26.zip>

The node declaration has been updated with a new function, and a private Boolean variable, which is initialized in the constructor. The data files are copied from the previous lab, and Program 07.

2. Add your code from Lab 25 to insert, connect, and display the nodes in the graph. A function find() has been added to make this simpler.

3. Compile and run the program, using Lab26a.txt (the data file from the previous lab) to verify that it works. You should see this displayed:

Sample Run

Enter file name: ***Lab26a.txt***

Sunday

Monday

Tuesday

Monday

Thursday

Tuesday

Friday

Thursday

Sunday

Wednesday

Saturday

Friday

Sunday

Wednesday

Saturday

Wednesday

Monday

Tuesday

Saturday

Text

Description automatically generated

4. Refer to the class notes on depth-first traversal. If you were to start at Sunday, what would the depth-first output be?

**Graph – Lab26a.txt:**

Diagram

Description automatically generated

**Graph – Lab26b.txt:**

Diagram, schematic

Description automatically generated

* Start at Sunday
* Mark Sunday as visited and output
* Move to Monday
* Mark Monday as visited and output
* Move to Thursday since Thursday is the next neighbor
  + Mark Thursday as visited and output
  + Move from Thursday to Sunday – already has been visited and marked
  + Move to Wednesday from Thursday
    - Mark Wednesday as visited and output
    - Move to Monday – already visited and marked
    - Move to Tuesday
      * Mark Tuesday as visited and output
      * Move to Friday
        + Mark Friday as visited and output
        + Move to Sunday – already visited (start point)
        + Move to Wednesday (already visited)
        + Move to Saturday
        + Mark Saturday as visited and output

Sunday has no more neighbors and Saturday has no neighbors to connect to

**Output should look like (similar):**

Sunday -> Monday -> Thursday -> Wednesday -> Tuesday -> Friday -> Saturday

5. Uncomment the section in main() for step 5. Implement the node::depth() function using the algorithm from class:

* *If visited == true, return*
* *Set visited = true*
* *Output this node’s contents*
* *Loop through each edge in list*
  + *Call edge[i]->depth()*

6. Test your function with data file Lab26b.txt and the start node A. Your output from the depth-first search should correspond to the chapter notes:

Sample Run

Enter file name: ***Lab26b.txt***

:

Enter start node: A

A B D C F E

7. Run your program twice for each data file, choosing a different start point for each, and save the outputs.

**Sample Outputs-Lab26a.txt:**

**Sunday as starting node:**

**Text

Description automatically generated**

**Wednesday as the starting node:**

Text

Description automatically generated

**Friday as the starting node:**

Text

Description automatically generated

**Sample Outputs-Lab26b.txt:**

**‘A’ as starting node:**

Text

Description automatically generated

**‘D’ as starting node:**

Text

Description automatically generated

**‘F’ as starting node:**

Text

Description automatically generated

**Code:**

**Node.h:**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Node.h

\* Written by Mark M Bowman

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Node Declarations

#define ERR -1

#define NODE\_MAX 20

#define EDGE\_MAX 4

// Node class

class node

{ public:

node(); // Constructor

void set\_value(string); // Set string value

string get\_value(); // Return string value

void connect(node \*); // Connect this node to another

void put(ostream &); // Output node and neighbors

void depth(ostream &); // Depth-first search;

private:

string value; // Node value

node \*edge[EDGE\_MAX]; // Edges array

bool visit;

};

**Node.cpp:**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Lab26.cpp

\* Written by Twymun K. Safford

\* Last Updated: 12/1/2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <string>

using namespace std;

#include "Node.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Null constructor

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

node::node()

{ int i;

value = "";

for(i=0;i<EDGE\_MAX;i++)

edge[i] = NULL;

visit = false;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* set\_value()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void node::set\_value(string arg)

{ value = arg;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* get\_value()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

string node::get\_value()

{ return value;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* connect()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void node::connect(node\* otherNodes)

//function to connect one node to another

{

//number of edges, existing index, and existing node

int numberOfEdges, existingIndex, existingNode;

numberOfEdges = 0;

existingIndex = 0;

existingNode = 0;

//counter variable

int i = 0;

cout << "Now connecting " << otherNodes->get\_value() << " to " << get\_value() << endl;

//search for the position which is empty in edge array of the node - search all elements of array edge

bool found;

found = false;

//search through all while the edge is not null

while (edge[i] != NULL)

{

if (edge[i] == otherNodes)

{

//if existing node is found, set index to i

found = true;

existingIndex = i;

}

//incrmeent by i

i++;

}

//set number of edges equal to i

numberOfEdges = i;

if (!found)

{

//if not found, the number of edges will be equal to the edges shared by the other node

edge[numberOfEdges] = otherNodes;

}

else

{

//if found, set the value of the edge for that index in array

//equal to the value of the edge for that index in the existing index

edge[numberOfEdges] = edge[existingIndex];

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* put()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void node::put(ostream&)

//output function-displays the node's value followed by values of node's immediate neighbors

{

//counter integer

int i = 0;

//print statement

cout << get\_value() << endl;

//need to put in while loop to print all edges of the node until NULL

while (edge[i] != NULL)

{

cout << "---" << edge[i]->get\_value() << endl;

i++;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* depth()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void node::depth(ostream&)

//function to print the traversal depth of a graph as we visit from node to node

//with connecting edges

{

// If visited == true, return - we have visited each and every node

if (visit == true)

{

return;

}

// Set visited = true once all nodes have been visited

visit = true;

// Output this node's contents.

cout << get\_value() << " ";

// Loop through each edge in list and Call edge[i]->depth().

int i = 0; //counter variable

while ((i < EDGE\_MAX) && (edge[i] != NULL))//as long as we haven't visited the maximum number of edges

//or we haven't incured an issue of an edge that is null

{

edge[i]->depth(cout);

i++;

}

}

**Main.cpp:**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Lab26.cpp

\* Written by Twymun K. Safford

\* Last Updated: 12/1/2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <iomanip>

#include <fstream>

#include <string>

using namespace std;

#include "Node.h"

int find(string name,node map[],int n);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* main()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void main()

{ int i,n;

int source\_index,target\_index,distance;

string fname,source\_name,target\_name;

fstream in;

bool isTargetPresent, isSourcePresent;

node map[NODE\_MAX];

node\* sourceNode;

node\* targetNode;

// Initialize

n = 0;

i = 0;

cout << left;

// Get file name

cout << "Enter file name: ";

cin >> fname;

// Open file

in.open(fname,ios::in);

// Loop through file

while(!in.eof())

{ in >> source\_name >> target\_name >> distance;

// Add to array

if(in.good())

{

isSourcePresent = false;

isTargetPresent = false;

//rest i back to zero

i = 0;

while (i < n)

{

if (map[i].get\_value() == source\_name)

{

//if source name is present

isSourcePresent = true;

//index of source i

source\_index = i;

}

else if (map[i].get\_value() == target\_name)

{

// If target is present

isTargetPresent = true;

// Index of target node

target\_index = i;

}

i++;

}

// Find source

// Add source if needed

if (!isSourcePresent)

{

//set source index to number of nodes

source\_index = n;

map[n++].set\_value(source\_name);

}

// Find target

// Add target if needed

if (!isTargetPresent)

{

target\_index = n;

map[n++].set\_value(target\_name);

}

// Add to array

//if (in.good())

//{

// cout << setw(12) << source\_name;

// cout << setw(12) << target\_name;

// cout << setw(4) << distance;

// cout << endl;

//};

sourceNode = &(map[source\_index]);

targetNode = &(map[target\_index]);

// Connect

sourceNode->connect(targetNode);

};

};

// Close file

in.close();

// Display array

for (i = 0; i < n; i++)

{

cout << "\n";

map[i].put(cout);

cout << "----------------" << endl;

}

//Step 5

// Get start

cout << "Enter start node: ";

cin >> source\_name;

source\_index = find(source\_name,map,n);

// Do depth first search

map[source\_index].depth(cout);

cout << endl;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* find()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int find(string name,node map[],int n)

{ int i;

// Loop through array

for (i = 0; i < n; i++)

{

if (name == map[i].get\_value())

{

return i;

}

}

// Return failure

return ERR;

}