Part I

1. Download the following files and create a workspace using them:

<https://markbowman.org/231/Lab25.zip>

The data file contains node information. Each line contains the name of the source node, target node, and distance. The links are unidirectional (directed).

2. Compile and run the program. You should see this displayed:

Sample Run

Enter file name: ***Lab25.txt***

Sunday Monday 10

Sunday Tuesday 20

Monday Thursday 30

Tuesday Friday 30

Wednesday Monday 20

Wednesday Tuesday 10

Thursday Sunday 50

Friday Sunday 60

Thursday Wednesday 10

Friday Wednesday 20

Thursday Saturday 20

Friday Saturday 10

The program reads from each line a source node, target node, and distance.

Text

Description automatically generated

Part II

1. Look at the data file. Draw a picture of how you think the nodes should be linked together. Arrange the nodes so that none of the pointer lines cross each other.

Part III

1. Add a loop inside your input loop to find the array location of each source node. Hint: Use get\_value() to compare a node’s value to source\_name. If the node does not exist, add it to the array using the set\_value() function and incrementing n. Save the index.
2. Comment out the output lines in the input loop. Test your program to verify that your array has been populated:

Sample Run

Enter file name: ***Lab25.txt***

Sunday

Monday

Tuesday

Wednesday

Thursday

Friday

Note that Saturday is not shown.

Text

Description automatically generated

Part IV

1. Add a second inner loop to find the array location of each target node. If the node does not exist, add it to the array by using the set\_value() function and then incrementing n. Save the index.
2. Test your program to verify that your array has been populated:

Sample Run

Enter file name: ***Lab25.txt***

Sunday

Monday

Tuesday

Thursday

Friday

Wednesday

Saturday

Note that the order of nodes is different now.

Text

Description automatically generated

Part V

1. Using the index of the source, and address of the target, call the connect() function.

Hint: &(map[index]) is a pointer to a node.

1. Implement the connect() function to add an edge to a node. It will use a loop to locate and update the first available entry in the edge[] array.
2. Implement the put() function so that it displays the node’s value, followed by the values of the node’s immediate neighbors. Example:

Sunday

Monday

Tuesday

1. Update your code in **main()** so that it calls the **put()** function to output the nodes, rather than using the **get\_value()** function.
2. Compile and run your program. Save your output.

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

private:

string value; // Node value

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

};

Node.cpp:

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

\* Lab25.cpp

\* Written by Twymun Safford

\* Date Updated: 11/22/2021

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

#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;

}

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

\* 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++;

}

}

Main.cpp:

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

\* Lab25.cpp

\* Written by Twymun Safford

\* Date Updated: 11/29/2021

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

#include <iostream>

#include <iomanip>

#include <fstream>

#include <string>

using namespace std;

#include "Node.h"

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

\* main()

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

void main()

{

int i, n; //n - number of nodes, i - node index

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;

if (in.good())

{

//first loop - inner loop

// add loop here

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++;

}

// Add to array

if (!isSourcePresent)

{

//set source index to number of nodes

source\_index = n;

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

}

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 to next node

sourceNode->connect(targetNode);

}

};

// Close file

in.close();

// Display array

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

{

//cout << map[i].get\_value() << endl;

cout << "\n";

map[i].put(cout);

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

}

}