American University of Sharjah
School of Engineering
Computer Engineering Department
P. O. Box 26666 Sharjah, UAE

Instructor: Dr. Ghassan Z. Qadah
Lab Instructor: Ms. Praveena Kolli
Office: EB2-126
Phone: 971-6-515-2352
e-mail: pkolli@aus.edu
Semester: Spring 2019

CMP305L Data Structures and Algorithms
Lab11

Objectives:

- To understand Vertices and Edges
- To implement Graph

Exercise 1

You are developing a Graph program that allows the user to Add vertices and Edges.

1. Write a class Vertex as given below:

```
class Vertex
{
public:
      Vertex(); //default constructor
      Vertex(string  n); //constructor with argument
      Vertex( Vertex& a); //copy constructor
      Vertex operator=(const Vertex& a)//assignment operator

bool operator == (const Vertex &a)//compares vertices with names only
bool operator != (const Vertex &a); //compares vertices with names
only
bool addEdge(string edgeName, int weight);// Adds an edge.
string getName();//return the name
void print()//print vertex name and it's edges.

private:
      string name;
      UnsortedType edges;
};
```

2. Write a structure Edge with city name and weight as members.

```
struct Edge
{
      string cityName;
      int weight;
      Edge()
      Edge(string  n, int wt)
      Edge(Edge& a)
      Edge operator=(const Edge& a)
      bool operator == (const Edge &a)
      bool operator != (const Edge &a)
      void print(); //prints cityName and weight.

};
```

3. Update given Unsorted list with info part as a struct Edge, `Vertex graph[10];`

4. Write a main with a declaration of an array of Vertex. Display a Menu to choose the options,
    1) Add a Vertex
    2) Add an Edge
    3) Print the Graph
    4) Exit

    A sample output is provided.

Conditions should be checked before the action,

a. On adding a vertex make sure name of the vertex does not exist in the list.
b. On adding the edge make sure that the vertex exist.

```
(base) sheikh@sheikheddy:
1. Add Vertex
2. Add an Egde
3. Print Graph
4. Exit
Choose an option: 1
Enter name:
Dubai
Choose an option: 1
Enter name:
Sharjah
Choose an option: 1
Enter name:
Ajman
Choose an option: 2
Enter Source:
Dubai
Enter Destination:
Sharjah
Enter weight:
25
Choose an option: 2
Enter Source:
Dubai
Enter Destination:
Ajman
Enter weight:
45
Choose an option: 2
Enter Source:
Sharjah
Enter Destination:
Dubai
Enter weight:
25
```

```
Choose an option: 2
Enter Source:
Ajman
Enter Destination:
Dubai
Enter weight:
15
Edge already exists between Ajman and Dubai
Choose an option: 2
Enter Source:
Hawaii
Source does not exist
Choose an option: 2
Enter Source:
Ajman
Enter Destination:
Fujairah
Destination does not exist
Choose an option: 1
Enter name:
Dubai
Vertex already in graph
Choose an option: 4
Goodbye!
```

```
25
Choose an option: 2
Enter Source:
Sharjah
Enter Destination:
Ajman
Enter weight:
10
Choose an option: 2
Enter Source:
Ajman
Enter Destination:
Dubai
Enter weight:
40
Choose an option: 3
Dubai to
        Ajman 45
        Sharjah 25
Sharjah to
        Ajman 10
        Dubai 25
Ajman to
        Dubai 40
```

```
//Vertex.h

#include"UnsortedType.cpp"

class Vertex
{
public:
	Vertex(); //default constructor
	Vertex(string  n); //constructor with argument
	Vertex( Vertex& a); //copy constructor
	Vertex operator=(const Vertex& a);//assignment operator

bool operator == (const Vertex &a);//compares vertices with names only
bool operator != (const Vertex &a); //compares vertices with names only
bool addEdge(string edgeName, int weight);// Adds an edge.
string getName() const;//return the name
void print();//print vertex name and it's edges.
bool hasEdge(string edgeName);

private:
	string name;
	UnsortedType edges;
};

//Vertex.cpp

#include"Vertex.h"
Vertex::Vertex(){ //default constructor
	name = "";
}
	Vertex::Vertex(string  n){ //constructor with argument
	name = n;
}
	Vertex::Vertex( Vertex& a){ //copy constructor
	name = a.getName();
	edges = a.edges;
}
	Vertex Vertex::operator=(const Vertex& a){//assignment operator

	if(this != &a){

	name = a.getName();
	edges = a.edges;

	}
	return *this;
```

```
}
bool Vertex::operator == (const Vertex &a){//compares vertices with names only
	return name == a.getName();
}
bool Vertex::operator != (const Vertex &a){ //compares vertices with names only
	return name != a.getName();
}
bool Vertex::addEdge(string edgeName, int weight){// Adds an edge.
	Edge e(edgeName, weight);
	try{
	edges.PutItem(e);
	return true;
	}catch (std::bad_alloc exception)
	{
		return false;
	}
}
string Vertex::getName() const{//return the name
	return name;
}
void Vertex::print(){//print vertex name and its edges.
	if(name != ""){
	std::cout << name << " to " <<std::endl;

	//I do not call ResetList() because GetNextItem loops around
	for(int i = 0; i < edges.GetLength(); ++i){
		std::cout << "\t";
		edges.GetNextItem().print();
		}
	}
}

bool Vertex::hasEdge(string edgeName){
	bool found;
	Edge e(edgeName, 0);
	edges.GetItem(e, found);
	return found;
}

//Non-class functions

const int SIZE = 10;

void displayOptions();
//reminder: arrays are always passed by reference
void makeDecision(int option, Vertex graph[]);
```

```
//these functions would be nice if we made Vertex[] iterable
void addVertex(Vertex graph[]);
void addEdge(Vertex graph[]);
bool exists(string a, Vertex graph[]);

int main(){

	Vertex graph[SIZE];

	int option;
	displayOptions();
	do{
	std::cout << "Choose an option: ";
	std::cin >> option;
	std::cin.ignore();
	makeDecision(option, graph);
	}while(option != 4);


	return 0;
}

void displayOptions(){
	std::cout << "1. Add Vertex\n"		\
		  << "2. Add an Egde\n"		\
		  << "3. Print Graph\n"		\
		  << "4. Exit"   << std::endl;
}

void makeDecision(int option, Vertex graph[]){
	switch(option){
	case 1: addVertex(graph); break;
	case 2: addEdge(graph); break;
	case 3: for(int i = 0; i < SIZE; ++i){graph[i].print();} break;
	case 4: std::cout << "Goodbye!" << std::endl; break;
	default: std::cout << "Please enter a valid option" << std::endl;break;
	}
}

void addVertex(Vertex graph[]){
	std::cout << "Enter name: " << std::endl;
	string n;
	getline(cin, n);
	Vertex v(n);
	for(int i = 0; i < SIZE; ++i){
```

```
                if(graph[i] == v){std::cout << "Vertex already in graph" << std::endl;
return;}
                if(graph[i].getName() == ""){graph[i] = v; return;}
        }
        std::cout << "Graph is full!" << std::endl;
}
void addEdge(Vertex graph[]){
        std::cout << "Enter Source: " << std::endl;
        string source;
        getline(std::cin, source);
        if(!exists(source, graph)){std::cout << "Source does not exist" << endl; return;}

        std::cout << "Enter Destination: " << std::endl;
        string destination;
        getline(std::cin, destination);
        if(!exists(destination, graph)){std::cout << "Destination does not exist" << endl;
return;}

        std::cout << "Enter weight: " << std::endl;
        int wt;
        std::cin >> wt;

        for(int i = 0; i < SIZE; ++i){
                if(graph[i].getName() == source){
                        if(graph[i].hasEdge(destination)){
                                std::cout << "Edge already exists between " << source << "
and " << destination << std::endl;
                        }else{
                                graph[i].addEdge(destination, wt);
                        }
                        return;
                }
        }

}

bool exists(string str, Vertex graph[]){
        for(int i = 0; i < SIZE; ++i){
                if(graph[i].getName() == str){return true;}
                if(graph[i].getName() == ""){return false;}
        }
}

//UnsortedType.h

#pragma once
```

```
#include <string>
#include<iostream>
#include<limits>
#define ItemType Edge

using std::string;

struct Edge
{
	string cityName;
	int weight;
	Edge(){cityName = ""; weight = std::numeric_limits<int>::max();}
	Edge(string  n, int wt){cityName = n; weight = wt;}
	Edge(Edge& a){cityName = a.cityName; weight = a.weight;}
	Edge operator=(const Edge& a){if(this!=&a){cityName = a.cityName; weight = a.weight;}return*this;}
	bool operator == (const Edge &a){return cityName==a.cityName;}
	bool operator != (const Edge &a){return cityName!=a.cityName;}
	void print(){std::cout << cityName << " " << weight << std::endl;} //prints cityName and weight.

};

struct NodeType{
	ItemType info;
	NodeType* next;
};

class UnsortedType
{

public:
	UnsortedType();

	UnsortedType(const UnsortedType& orginal);
	UnsortedType& operator=(const UnsortedType& orginal);
	~UnsortedType();
	void MakeEmpty();
	bool IsFull() const;
	int GetLength() const;
	ItemType GetItem(ItemType& item, bool& found) ;
	void PutItem(ItemType item);
	void DeleteItem(ItemType item);
	void ResetList();
	ItemType GetNextItem();
private:
```

```
        NodeType* listData;
        int length;
        NodeType* currentPos;
};

//UnsortedType.cpp has not been changed
```