Lab 02

Resources

- Discussion slides (./lab02slides.pdf) - Copy of the presentation given by the TA
- More Unix (http://www.ucs.cam.ac.uk/docs/leaflets/u5) - A list of 30 useful uni commands

Idea

The goal of this lab is to do some preparation for your first project. We will go over s design and implementation and I/O in the context of graphs. Later on we will expan to explore more areas of C++.

In-lab assignment

Description

You are to write a program using the given template struct. You will implement a "G data type that supports operations for adding vertices, edges, and printing out the st the graph. We will use structs and arrays to do this. You will also get experience usin menu and prompting the user for commands to enter.

Start with the following code and add in the required functionality:

```
#include <iostream>
using namespace std;


const int MAX_VERTICES = 100;
```

```
const int MAX_EDGES = 4950;

// A graph structure implemented with arrays
struct Graph{
    string vertices[MAX_VERTICES];
    int edges[MAX_EDGES][2];
    int numVertices;
    int numEdges;

    // Constructor, called when a graph is created
    Graph(){

    }

    // Print the entire graph according to the format in the
    // lab description
    void printGraph(){

    }

    // Print the edges of a given vertex
    void printEdgesOfVertex(string vert){

    }

    // Add a vertex to the graph
    void addVertex(string vert){

    }

    // Add an edge to the graph
    void addEdge(string vert1, string vert2){

    }
};

// Should provide the functionality described in the lab
```

```
int main(int argc, char ** argv){


    return 0;
}
```

The struct contains an array of strings which represent the vertices of the graph (by
storing their names). The variable `numVertices` keeps track of how many vertices th
graph currently has. You should increment or decrement this as necessary. The `edge`
array is a 2D array. The first index denotes the edge number. The second index has 2
possibilities which denote the *indices* of the vertices the edge is between. So if
`vertices[1]` is "abc" and `vertices[2]` is "def" and there is an edge between verte
and 2 ("abc" and "def") with an index of `i`, `edges[i][0]` should be 1 and `edges[i]`
should be 2 (or vice versa). The variable `numEdges` keeps track of the number of edg
currently in your graph.

Requirements/Deliverables

- Your program should create an empty graph and then display a prompt to the u
  with the following options:
  0. Exit
  1. Add a vertex
  2. Add an edge
  3. Print the graph
  4. Print the edges of a vertex

  >
- The program should keep displaying the prompt each time the user enters a
  command until the user hits exit.
- Exit exits the program.
- Add a vertex prompts the user for the name of a vertex and then creates a verte
  that name
- Add an edge prompts the user for 2 vertices (names) to connect with an edge an
  adds an edge between them
- Print the graph should print the graph in the following format

```
Vertices: [vertex_1], [vertex_2], [vertex_3], ...
Edges: (edge_1_vert_1, edge_1_vert_2), (edge_2_vert_1,
edge_2_vert_2), ...
```

- Print the edges of a vertex should prompt the user for the name of a vertex and p the edges of that vertex in the following format:

  ```
  [vertex_name]: (edge_1_vert_1, edge_1_vert_2), (edge_2_vert_1,
  edge_2_vert_2), ...
  ```

  Note that in this case, only the edges containing the given vertex should be print the given vertex does not exist in the graph, you should print:

  ```
  Error: [vertex_name] does not exist
  ```

Hints

- Please see the example output (./exampleoutput.txt) for a sample run of a worki program
- You may want to create additional functions or methods to do logical things (ie: display prompt method or print edge method)
- Make sure you test each thing you do throuroughly before moving on to other pa
- You can test your menu before you even think about writing code for Graph, just print statement in each Graph function with the function name and check to ma sure you are calling them correctly. These are called function *stubs*
- Likewise, you can test your graph functions by hard-coding function calls in mai without having a menu.
- strings can be compared using the `==` operator, just like a primitive.

Grading Distribution

- 2 points for correct implementation of the menu and user prompts
- 2 points for add vertex method
- 2 points for add edge method
- 2 points for print graph method
- 2 points for print edges of vertex method
  - 2 points for handling vertex does not exist case

- 3 points for good style

Optional Enhancements

The following are completely optional additions you can add to your code once you a
done. Completion or incompletion will not aid/hurt your grade.

- Add in functionality to remove vertices or edges. Note that removing a vertex re
removing any edges that vertex was a part of
- Work in a way to save/load graphs to a simple text file. You could the ifstream c
for this. You would need to come up with a consistent format to use ie:

```
num_vertices
vert_1
vert_2
...
num_edges
index_of_edge_1_vert_1 index_of_edge_1_vert_2
index_of_edge_2_vert_1 index_of_edge_2_vert_2
...
```

- Check for error cases, such as adding in a vertex that already exists or an edge t
already there.
- Write a function to clear the edges of a graph or clear the vertices of a graph.
- Write a function that creates a *complete* graph on the current vertices. A compl
graph has an edge between each pair of vertices (how many edges will you need?
- Keep the vertices in the edges ordered in alphabetical order. So an edge (def, ab
be stored as (abc, def).
- Anything else fancy you want to add