MIDDLE EAST TECHNICAL UNIVERSITY, NORTHERN CYPRUS CAMPUS
CNG315 Algorithms

Assignment 3: Communication Network Graph

This assignment aims to help you practice **graph data structure and basic graph operations**. Your main task in this assignment is to develop a graph of emails using **C programming language**. Please note that you need to implement the graph as an **adjacency list** in this assignment.

Overview:
It is time to put the emails you have worked on and processed in the previous assignments in graph format. In other words, making a network of users and linking them together based on emails they exchange will be your task in this assignment.

To be more specific, you will need to develop a program with the following functionalities:

- Read a list of emails from files without making assumptions about the length of the content of each email
- Make sure to take out each user and record how many emails they sent and received.
- Create a directed weighted graph where a vertex is created for each person who sent/received an email, and an edge is created between two vertices representing an email between users. For example, if A sent B an email, an edge should be drawn from A directed towards B. The weight of the edge should be equal the number of words in the content of that email. If A sent multiple emails to B, then the weight of the edge should be the sum of the number of words in the emails.
- Display the graph by showing the vertices and edges by printing the adjacency list
- Display which people have the maximum number of emails sent and which people have the maximum number of words received.
- Check if there is a path between two given people

Input:
The program will have data files as inputs:

- Data files with each one storing an email.

  Email_ID
  From: Sender
  To: Recipient
  Date: Day_of_the_month
  Content

Information	Data Type
Email_ID	int
Sender	char[51]
Recipient	char[51]
Day_of_the_month	int
Content	Char*

Internal Processing:
The internal processing will be performed using the following functions. The main function calls the **readEmails, printGraph** and **checkPath** functions. The **readEmails** function calls **createVertex** and **createEdge** functions.

- **readEmails**: This function takes the path to the directory which contains the data files and the number of data files. It then reads the files that are named according to a number. For example, if there are 50 data files, then the files will be named "1.txt", "2.txt", all the way to "50.txt". **Be sure to add a few error checks to this function, such as when the user enters a path that does not exist or a number of emails that is more than the one found in the entered directory.**

  This function will first create a graph. When this function reads an email, it will use the **createVertex** function to create a vertex for the sender and the receipt if they are not already added to the graph. The function will then use the **createEdge** function to create an edge between the sender and receiver if it is not already created. Once the function reads all the emails, it will return the graph.

- **createVertex:** This function takes the graph and the person's name (the sender or the receipt). It then checks if the person is already added. If not, it will create a new vertex and add the vertex to the graph. Each vertex should keep the person's name, the number of emails s/he sent, and the number of emails s/he received.

- **createEdge:** This function takes the graph, the sender's name, the receipt's name, and the number of words in the email. It then creates an edge between the sender and the receipt, and the weight of the edge will equal the number of words in the email. If an edge between the sender and receiver is already created, then the weight of the edge is updated by adding the number of words to the current weight. Please note that this function creates a directed edge. If an email is sent from A to B, the function should add the edge from A to B. Since the graph should keep track of the number of sent and received emails for each person, this function should also update the related fields in the vertex.

- **printGraph:** Once the graph is created, the **printGraph** function should be executed and this function will print:
  - The adjacency list of the resulting graph along with the vertices.
  - The people who sent the maximum number of emails (which can be more than one person)
  - The people who received the maximum number of words (which can be more than one person)

  The format would look like:

```
The resulting graph's adjacency list:
Jimmy -> Ash | 3 -> Malena | 7 -> Molly | 4
Ash -> Molly | 5 -> Malena | 8
Molly -> Ash | 6 -> Jimmy | 10
Aissen -> Ash | 5
Malena ->

People with the maximum number of emails sent:
Jimmy sent 3 emails

People with the maximum number of words received:
Melena received 15 words
```

- **checkPath:** After calling the **printGraph** function, the main function will ask for two people names and then call **checkPath** function with the graph and two people names. This function

will check if there is a path in the graph from the first person to the second one. If there is part, the function will return 1. Otherwise, it will return 0. The main function will then print whether there is path between two people or not.

Incremental and Modular Development

You are expected to follow an incremental development approach. Each time you complete a function or module, you are expected to test it to make sure that it works properly. You are also expected to follow modular programming which means that you are expected to divide your program into a set of meaningful functions.

Professionalism and Ethics

You are expected to complete the assignments on your own. Sharing your work with others, uploading the assignment to online websites to seek solutions, and/or presenting someone else's work as your own work will be considered as cheating.

Rules

- You need to write your program by using C programming.
- You need to name your file with your student id, e.g., 1234567.c, and submit it to ODTUCLASS.
- Code quality, modularity, efficiency, maintainability, and appropriate comments will be part of the grading.
- You should not use any code from other resources.
- **You need strictly follow the specifications given in this document.**
- **It is suggested to use helper functions to support modular development.**

Grading Scheme

Grading Item	**Mark (out of 100)**
Graph Structure and Correct Use of Graph Structure	10
Main function	5
readEmails	20
createVertex	10
createEdge	15
printGraph	20
checkPath	20