Hacettepe University
Computer Science Department
BBM 203 Programming Assignment
Name and Surname: Mert Çökelek
ID: 21727082
Subject: Data Structures, Abstract Data Types (Queues, Stacks), Recursion, Dynamic Memory Allocation, File i/o
Data Due: 25.11.2018
Advisor: Selim Yılmaz
Email: b21727082@cs.hacettepe.edu.tr - 100mert101@gmail.com

PROGRAMMING ASSIGNMENT 2 REPORT

1. ***What is the problem and what needs to be understood?***

   The main goal of this assignment is to implement ADT such as stacks and queues for a very useful way - transfer of a message froma client to another in a client community. -in daily life.

2. ***My approach to this problem is:***
   a. Determine the inputs and find a efficient way to use them.
      i. The inputs are:
         1. Clients.dat: Holds information about the clients, such as ID, IP, MAC addresses.
         2. Routing.dat: Holds information about the links between two clients.
         3. Commands.dat: The commands of the program which are going to be processed are obtained from this file.
         4. 4th argument: The Maximum Size of the message which can be sent in once.
         5. 5th and 6th arguments: Are the sender's port and receiver's port numbers, respectively.
      ii. The output is printed on the screen.
   b. ***Data Structure:***
      i. I created 2 Header files for the modularity of the code and 1 main.c file where the main function is processed.
         1. The Header Files are:
            a. Stack.h: In this header a basic Stack structure is defined.
               i. The "layer struct" has been implemented as a doubly linked list element. It holds informations about a frame's sender and receiver either IDs, IPs, MACs, port numbers and message chunk, depending on the layer type. And since it has been

implemented as a linked list, the struct has a prev and a next layer pointer.

ii. The functions in this header are:

1. Void Push(Layer* layer, char* senderInfo, char* receiverInfo, char* chunk): This function takes 4 parameters, 1 for the stack pointer, the others for the infos to be pushed. The stack is being initialized with this function. Used in Message and Send commands.
2. Void Pop(Layer* layer): This function pops from the top of is its input stack. Used for changing the MAC address in SEND command.
3. Void PrintStack(Layer* layer): Prints the parameter stack from top to down, Used in Message command.

b. Queue.h: In this header a basic Queue Structure is defined.

i. The "frame struct" is implemented as a single linked list. It holds a layer pointer (which corresponds to a message frame) and a next frame pointer.

ii. The functions are:

1. Void Enque(Frame* , Layer*): It adds the second parameter(Stack) into the first parameter(queue). Used in Message and Send command to create or change the outgoing-incoming queues.
2. Layer* Deque(Frame*): This function removes the first enqued element in the queue. Used in Send command.

c. The main File # Main.c # : This is the main source file of the program.

i. Standart libraries like math.h, stdio.h, stdlib.h, string.h, and also the user-defined headers like stack.h and queue.h are included.

ii. I defined a struct for the clients, the fields are:

1. ID
2. IP
3. MAC
4. Port number
5. Routes Array
6. Incoming and Outgoing Queues
7. Succession flags for the send operation.

iii. The functions I defined in this file are:

1. Split: This function takes 3 parameters, 1- The string to be splitted, 2- The token, 3- the length of the returning array.
2. SplitIntoChunks: This function splits the given string (the Message in this assignment) into smaller chunks according to the max capacity of the frames.

3. GetClients: This function reads the clients.dat file and returns an array holding the client struct. The fields of the client struct are filled here.
4. GetRoutes: Reads the routing.dat file and initializes the routes of the clients.
5. MESSAGE: This function is called when the commands.dat file is read and the current line is MESSAGE. Finds the correct sender and receiver from the client array, creates frames from layers, and fills the client's outgoingQueue with the frames.
6. FindNextDestination: This function is called while send command is processed. From the sender's routes, finds the next destination of the frame to be sent.
7. FindReceiver: This function is also called for send command. From the sender's outgoingQueue's ingredients, finds the receiver for making the transfer.
8. MakeForward: This is also called for send command. If the current destination is not the real receiver, changes the mac addresses of the top of the stacks, and makes forward of the message.
9. SEND: This function is called when the send command is active while reading the commands.dat. This is recursive function and makes the transfer of the message from the sender to the receiver. The number of hops, client logs, client's queues are dynamically changed here. Finally, if there is a link between sender and the receiver, returns '1' value for success, and for '0', for unsuccessful send operation.
10. ReadCommands: In this function, the commands.dat file is read and the lines are processed with order. The Message, Show Queue and Frame info, Send, Print log commands are processed here.
11. Finally, the main function. The Files are opened, Clients are created here, routes are initialized, arguments are taken and the readcommands function is called.
12. And lastly, in main function, freed the allocated memories and closed the files.