23F-0599 Naveed Hassan

23F-0541 Mustafa Ahmad

DATA Lab Project (MINI INSTRAGRAM )

BSCS-3E

Code:

Main.cpp

#include<iostream>

#include"insta.h"

using namespace std;

//Added security to send follow request only once

//Added security that the password line will be displayed

int main()

{

int choice;

insta obj;

user\* root = obj.Root();

user\*& Main\_root = obj.Root();

obj.automatic();

do

{

cout << "\n---------------> Social Media App Menu <----------------\n";

cout << "1. Signup\n"; //Done

cout << "2. Login\n"; //Done

cout << "3. Reset Password\n"; //Done

cout << "4. Preorder Traversal of users\n"; //Done

cout << "5. Printing all the vertex \n";//Done

cout << "0. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice)

{

case 1:

{

obj.signup();

break;

}

case 2:

{

root = obj.login();

if (root!=NULL)

{

int subchoice;

do

{

cout << "\n---------------> Login Menu <----------------\n";

cout << "1. View Pending Follow Requests\n"; //Done

cout << "2. Accept/Cancel Follow Requests\n"; //Done

cout << "3. View Timeline (Posts from Followers)\n";//Done

cout << "4. View Notifications\n"; //Done

cout << "5. Send Message\n"; //Done

cout << "6. Search Users\n"; //Done

cout << "7. View Followers List\n"; //Done

cout << "8. View Your Posts (Newsfeed)\n"; //Done

cout << "9. Upload Your Posts\n"; //Done

cout << "10. Display the message inbox .\n"; //Done

cout << "11. View the Following list .\n"; //Done

cout << "12.Send Follow request .\n"; //Done

cout << "13.Print the Graph .\n"; //Done

cout << "0. Logout\n"; //Done

cout << "Enter your choice: ";

cin >> subchoice;

switch (subchoice)

{

case 1:

cout << "\nView Pending Follow Requests selected.\n";

root->pending\_Requests\_printer(root);

break;

case 2:

cout << "\nAccept/Cancel Follow Requests selected.\n";

obj.Decision\_Requests(root);

break;

case 3:

cout << "\nView Timeline(Post From Followers) selected.\n";

obj.feed(root->name);

break;

case 4: //Notification of a login ,request recieving,message recieving,request accepted will come

cout << "\nView Notifications selected.\n";

cout << "Your notificationa are : \n";

root->notifications.print(root->notifications.head);

break;

case 5:

cout << "\nSend Message selected.\n";

obj.message\_sendd(root->name);

break;

case 6:

cout << "\nSearch Users selected.\n";

obj.search();

break;

case 7:

cout << "\nView Followers List selected.\n";

obj.print\_Follower\_list(root->name);

break;

case 8:

cout << "\nView Your Posts selected.\n";

root->mine\_post\_check();

break;

case 9:

{

cout << "\nPost Uploader selected .\n";

//Main root is requireed to post it in their followes list

string text;

cout << "Enter the text that you want to upload : \n";

cin >> text;

string ttime;

time\_t now = time(0); //Getting current time

ttime = ctime(&now); // Converting time\_t to string using ctime

root->post\_uploader(root, Main\_root,text,ttime);

obj.GRAPH.FEED\_UPLOADER(text,ttime,root->name,Main\_root);

}

break;

case 10:

{

cout << "\nMessage inbox selected .\n";

root->message\_inbox(root,obj.current);

break;

}

case 11:

{

cout << "\nFollowing List selected .\n";

obj.print\_Following\_list(root);

break;

}

case 12:

{

cout << "Enter the name of person to send request :\n";

string temp;

cin >> temp;

obj.send\_request(Main\_root,temp,root->name);

//Functionality of sending the follow request

break;

}

case 13:

{

cout << "Print the graph selected .\n";

obj.print\_Graph();

break;

}

case 0:

cout << "Logging out...\n";

root=NULL;

break;

default:

cout << "Invalid choice! Please try again.\n";

break;

}

} while (root);

}

break;

}

case 3:

{

obj.resset();

break;

}

case 4:

{

obj.pre\_order(Main\_root);

break;

}

case 5:

{

obj.vertex\_print();

break;

}

case 0:

{

cout << "Exiting the application.goodbye!\n";

break;

}

default:

{

cout << "Invalid choice! ,Enter the valid choice .\n";

}

}

} while (choice != 0);

return 0;

}

Insta.h

#pragma once

#include"graph.h"

#include <cstdlib>

#include <ctime>

#include<iostream>

#include<fstream>

#include<string>

class insta

{

user\* root;

public:

friend class user;

graph\* GRAPH\_ROOT\_ACESS();

graph GRAPH;

int current;

void feed(string& name);//Print posts of its following

~insta();

void print\_Following\_list(user\* root);

insta();

void print\_Follower\_list(string& name);

void print\_Graph();

void Decision\_Requests(user\*);

void send\_request(user\*&,string&,string&);

void message\_sendd(string& mine);

void vertex\_print();

void resset();

user\* login();

void search();

void signup();//To signup with a new account

void automatic();//that function will create old objects vased on previous use

void pre\_order(user\*);

user\* Root\_finder();

user\*& Root();

};

Insta.cpp

#include"insta.h"

void insta::message\_sendd(string& mine)

{

string to,message;

cout << "Enter the reciever id : ";

cin >> to;

cout << "Enter the message : \n";

cin >> message;

root->messages\_send(root, to, mine,message,current);

}

void insta::print\_Following\_list(user\* root)

{

gnode\* temp = GRAPH.vertex\_finder(root);

GRAPH.print\_following\_list(temp);

}

void insta::print\_Follower\_list(string& name)

{

GRAPH.print\_follower\_list(name);

}

void insta::feed(string& name)

{

root->print\_feed(root,name);

}

graph\* insta::GRAPH\_ROOT\_ACESS()

{

return &GRAPH;

}

void insta::vertex\_print()

{

GRAPH.display();

}

insta::insta()

{

root = NULL;

current = 0;

}

void insta::automatic()

{

string password,user\_name="0",city;

ifstream in("password.txt");

while (in >> user\_name >> password >> city)

{

if (user\_name != "0")

{

++current;

cout << "Account created (based onprevious use) for : " << user\_name << '\n';

GRAPH.vertex\_insertion(user\_name); //creating the vertex in the edge

gnode\* temp = GRAPH.ROOT();

root->insertion(root, user\_name, password, city,temp,current);

}

}

}

void insta::search()

{

string temp;

cout << "Enter the user\_name to search : \n";

cin >> temp;

root->search(root,temp);

}

user\* insta::login()

{

cout << "Enter the name of Account : \n";

string name,password;

cin >> name;

cout << "Enter the password : \n";

cin >> password;

return(root->login\_helper(root, name,password));

}

user\*& insta::Root()

{

return root;

}

void insta::resset()

{

string city,user\_name;

cout << "Enter the city name of lost account : \n";

cin >> city;

cout << "Enter the user name of lost account : \n";

cin >> user\_name;

root->reset(root,user\_name,city);

}

insta::~insta()

{

root->makenull(root);

cout << "ALL - Accounts closed .\n";

}

void insta::pre\_order(user\* root)

{

if (root != NULL)

{

cout << "\n--------------------------------------\n";

cout << "User Name : " << root->name << '\n';

cout << "City : " << root->city << '\n';

cout << "Last login info : " << root->last\_login << '\n';

pre\_order(root->left);

pre\_order(root->right);

}

}

void insta::print\_Graph()

{

GRAPH.display();

}

void insta::Decision\_Requests(user\* Root)

{

Root->pending\_Requests\_printer(Root);

cout << "Enter the 1 To accept Request and 0 to cancel Request .\n";

int choice;

cin >> choice;

string reciever\_temporary\_name = Root->name; //To give a notification to it

if (choice == 0)

{

string target;

cout << "Enter the id to Cancel its request : \n";

cin >> target;

Root->Requests.delete\_from\_center(Root->Requests.head, target);

}

else if(choice == 1)

{

string temp,to;

cout << "Enter the id to Accept its request : \n";

cin >> temp;

to = Root->name;

GRAPH.edge\_insertion\_directed(temp, to, "Friend", "Active");

Root->Requests.delete\_from\_center(Root->Requests.head, temp);

Root = Root->Root\_finder(root,temp);

string t = "Your request is accepted by ";

t += reciever\_temporary\_name;

Root->notifications.enqueue(Root->notifications.head, t);

}

}

void insta::send\_request(user\*& root,string& destination,string& From)

{

root->send\_request(root,destination,From);

}

user\* insta::Root\_finder()

{

cout << "Enter the name of account to help us to confirm its really you trying to sign in .\n";

string name;

cin >> name;

return(root->Root\_finder(root,name));

}

void insta::signup()

{

++current;

string password,user\_name,city;

int temp = 0;

cout << "Enter the user name to create account : ";

cin >> user\_name;

if (user\_name.length() < 5)

{

cout << "\nYour username is not unique, it is being set to a new one.\n";

srand(time(0));

for (int i = 0; i < 3; ++i)

{

int temp = 65 + rand() % 26;

user\_name += char(temp);

}

}

ofstream out("password.txt",ios::app); //Opeining the file in append mode so it will not override above value

out << user\_name<<'\n';

cout << "Enter the password to set : \n";

cin >> password;

while (password.length() < 5)

{

cout << "Enter th password again it is too short : \n";

cin >> password;

}

out << password << '\n';

cout << "Enter the city name : \n";

cin >> city;

out << city<<'\n';

cout << "username : "<<user\_name << '\n';

cout << "Your password is setted to : " << password << '\n';

cout << "your city is : " << city << '\n';

GRAPH.vertex\_insertion(user\_name);

gnode\* t = GRAPH.ROOT();

root->insertion(root,user\_name, password, city,t,current);

}

User.h

#pragma once

#include"Messages\_stack.h"

#include<iostream>

#include<ctime>

#include<string>

#include"stack.h"

#include"queue.h"

#include"graph.h"

using namespace std;

struct gnode; //Forward declaration

class insta;

class user

{

string password;

public:

void message\_inbox(user\* temp,int current);

string last\_login; //To display Account is active or not

Messages\_stack\* messages;

user\* left,\*right;

string name;

string city;

stack post;

stack Feed;

void print\_feed(user\*,string& name);

queue Requests,notifications;

void pending\_Requests\_printer(user\* root);

void send\_request(user\*&,string& ,string&);

void messages\_send(user\*,string,string,string&,int current);

void post\_uploader(user\*,user\*,string&,string&);

void mine\_post\_check();

void search(user\*,string);

void reset(user\*, string,string);

user\* Root\_finder(user\* root,string user\_name);

user(string name, string password, string city);

void insertion(user\*& root, string, string, string,gnode\* temp,int count);

void makenull(user\*&);

user\* login\_helper(user\* root, string& name, string& password);

~user();

};

User.cpp

#include"user.h"

user::~user()

{

delete[]messages;

}

user::user(string name, string password, string city)

{

time\_t now = time(0); //Getting current time

this->last\_login = ctime(&now); // Converting time\_t to string using ctime

this->name = name;

this->password = password;

this->city = city;

left = right = NULL;

messages = new Messages\_stack[10];

}

void user::insertion(user\*& root ,string name, string password, string city,gnode\* temp,int count)

{

if (root == NULL)

{

root = new user(name, password, city);

//Creating the messages stack for the every id created

for (int i = 0; i < count; ++i)

{

root->messages[i].person\_name = temp->vertex;

temp = temp->next;

}

return;

}

if (name > root->name)

{

insertion(root->right, name, password, city,temp,count);

}

else if (name < root->name)

{

insertion(root->left, name, password, city,temp,count);

}

else if (name == root->name)

{

cout << "Duplicate user\_name Found .\n";

}

}

void user::search(user\* root,string name)

{

if (root == NULL)

{

cout << "User Not Found .\n";

return;

}

if (name > root->name)

{

search(root->right,name);

}

else if (name < root->name)

{

search(root->left,name);

}

else

{

cout << "User Found successfully .\n";

cout << "User name is : " << root->name << '\n';

cout << "User city is : " << root->city << '\n';

cout << "User Last login is : " << root->last\_login << '\n';

}

}

void user::makenull(user\*& root)

{

if (root != NULL)

{

makenull(root->left);

makenull(root->right);

delete root;

}

}

user\* user::login\_helper(user\* root, string& name, string& password)

{

if (root == NULL)

{

cout << "User Not Found .\n";

return NULL;

}

if (name > root->name)

{

return(login\_helper(root->right, name,password));

}

else if (name < root->name)

{

return(login\_helper(root->left, name,password));

}

else if(root->name==name&& root->password==password)

{

cout << "Logged in succesfully .\n";

string temp = "You are logged in an new device .\n";

root->notifications.enqueue(root->notifications.head, temp); //A notifications will b erecieved og login

return root;

}

else

{

cout << "PASSWORD OR USER\_NAME IS INCORRECT ! \n";

return NULL;

}

}

void user::reset(user\* root, string user\_name,string city)

{

if (root == NULL)

{

cout << "Accurrent with following details not found .\n";

return;

}

if (name > root->name)

{

reset(root->right,user\_name,city);

}

else if (name < root->name)

{

reset(root->left,user\_name,city);

}

else

{

if (root->name == user\_name && city == root->city)

{

cout << "You have succesfully Found Your Lost password .\n";

cout << "Old password is : " << root->password << '\n';

cout << "Enter the new password : \n";

cin >> root->password;

}

}

}

void user::messages\_send(user\* root,string To,string from, string& message,int current)

{

user\* temp = Root\_finder(root,To);

for (int i = 0; i < current; ++i)

{

if (temp->messages[i].person\_name == from)

{

temp->messages[i].push(temp->messages[i].head,message);

string help;

help = "You have recieved a message from ";

help += from;

temp->notifications.enqueue(temp->notifications.head, help);//Notification of a message

}

}

}

void user::send\_request(user\*& root,string& destination,string& From)

{

user\* To = Root\_finder(root,destination);

if (To == NULL)

{

cout << "User Not Found .\n";

}

else

{

if(To->Requests.front() != NULL && To->Requests.front()->message == From)

{

cout << "You have already sent the request .\n";

return;

}

cout << "Request is SuccessFully sended to " << To->name << '\n';

To->Requests.enqueue(To->Requests.head,From);

string temp;

temp = "You have a new friend request from ";

temp += From;

To->notifications.enqueue(To->notifications.head,temp);//A notification will go to a person of new request

}

}

void user::pending\_Requests\_printer(user\* root)

{

cout << "Request are From : \n";

root->Requests.print(root->Requests.head);

}

user\* user::Root\_finder(user\* root,string name)

{

if (root == NULL)

{

cout << "User Not Found .\n";

return NULL;

}

if (name > root->name)

{

return(Root\_finder(root->right, name));

}

else if (name < root->name)

{

return(Root\_finder(root->left, name));

}

else

{

return root;

}

}

//To check a accurrent its posts

void user::mine\_post\_check()

{

post.print(post.head);

}

//Main root is requireed to post it in their followes list

void user::post\_uploader(user\* root,user\* Main\_root,string& text,string& ttime)

{

root->post.push(post.head,text, ttime);

//app->GRAPH.FEED\_UPLOADER(text,ttime,root->name,Main\_root);

}

void user::message\_inbox(user\* temp,int current)

{

cout << "Messages inbox : \n";

for (int i = 0; i < current; ++i)

{

cout << temp->messages[i].person\_name << " : \n";

messages[i].print(temp->messages[i].head);

}

}

void user::print\_feed(user\* root,string& name)

{

user\* temp = Root\_finder(root, name);

temp->Feed.print(temp->Feed.head);

}

Stack.h

#pragma once

#include<iostream>

#include<ctime>

#include<time.h>

#include<string>

using namespace std;

struct node

{

string text;

string time;

node\* next;

node(string text,string);

};

class stack

{

public:

node\* head;

void print(node\* head);

stack();

bool isempty();

void push(node\*& data,string,string);

void pop();

node\* top();

~stack();

};

Stack.cpp

#include "stack.h"

node::node(string text,string time)

{

this->text = text;

this->time = time;

this->next = NULL;

}

void stack::print(node\* head)

{

if (head == NULL)

{

cout << "Your post section is empty .\n";

return;

}

node\* temp = head;

while (temp != nullptr)

{

cout << "Text: " << temp->text << ", Time: " << temp->time << endl;

temp = temp->next;

}

}

stack::stack()

{

head = nullptr;

}

bool stack::isempty()

{

return head == nullptr;

}

void stack::push(node\*& head,string text,string time)

{

if (head == NULL)

{

head = new node(text,time);

return;

}

node\* temp = new node(text,time);

temp->next = head;

head = temp;

}

void stack::pop()

{

if (isempty()) return;

node\* temp = head;

head = head->next;

delete temp;

}

node\* stack::top()

{

return head;

}

stack::~stack()

{

node\* temp;

while (head != nullptr)

{

temp = head->next;

delete head;

head = temp;

}

}

Graph.h

#pragma once

#include<iostream>

#include <string>

#include"user.h"

using namespace std;

struct gnode

{

string vertex; //Name of a vertex

string status; //Blocked or not etc

string type; //Friends or Best Friends etc

gnode\* next;

gnode\* edge; //Edge willl be connecting with that

public:

gnode(string vertex);

};

class graph

{

gnode\* head;

public:

friend class user;

void FEED\_UPLOADER(string& text, string& time, string name,user\* Main\_root);

gnode\* ROOT();

graph();

void print\_follower\_list(string temp);

void print\_following\_list(gnode\*);

gnode\* vertex\_finder(user\* name);

void vertex\_insertion(string vertex);

void edge\_insertion\_directed(string from, string to, string type, string status);

~graph();

void display();

};

Graph.cpp

#include"graph.h"

gnode::gnode(string vertex)

{

this->vertex = vertex;

status = "Active";

type = "Friend";

edge = next = NULL;

}

graph::graph()

{

head = NULL;

}

void graph::print\_following\_list(gnode\* root)

{

cout << "Following list : \n";

while (root->edge != NULL)

{

root = root->edge;

cout<<root->vertex<<" ";

}

}

gnode\* graph::ROOT()

{

return head;

}

void graph::vertex\_insertion(string data)

{

if (head == NULL)

{

head = new gnode(data);

}

else

{

gnode\* temp = head;

while (temp->next != NULL)

{

temp = temp->next;

}

temp->next = new gnode(data);

}

}

void graph::edge\_insertion\_directed(string from, string to, string type, string status)

{

gnode\* temp = head;

while (temp->vertex != from)

{

temp = temp->next;

}

gnode\* from\_gnode = temp;

temp = head;

while (temp->vertex != to)

{

temp = temp->next;

}

gnode\* to\_gnode = new gnode(temp->vertex);

gnode\* t = from\_gnode->edge; // t is the previous edge if it is connected through this vertex

from\_gnode->edge = to\_gnode;

to\_gnode->edge = t;

to\_gnode->status=status ;

to\_gnode->type = type ;

}

graph::~graph()

{

gnode\* temp;

while (head != NULL)

{

temp = head->next;

gnode\* t = head->edge;

while (t != NULL)

{

gnode\* temp2 = t->next;

delete t;

t = temp2;

}

delete head;

head = temp;

}

}

gnode\* graph::vertex\_finder(user\* input)

{

gnode\* temp = head;

while (temp != NULL)

{

if (temp->vertex == input->name)

{

return temp;

}

temp = temp->next;

}

return nullptr;

}

void graph::display()

{

gnode\* temp = head;

while (temp != NULL)

{

gnode\* t = temp->edge;

cout << temp->vertex<< " ";

cout << "Following the accounts : ";

if (t == NULL) cout << " Empty\n";

while (t != NULL)

{

cout << "Name : " << t->vertex << " status : "<< t->status<<" type : "<<t->type<<" .";

t = t->edge;

}

cout << endl;

temp = temp->next;

}

}

void graph::print\_follower\_list(string name)

{

gnode\* temp = head;

cout << "Follwers are : ";

while (temp != NULL)

{

gnode\* t = temp->edge;

while (t != NULL)

{

if (t->vertex == name)

{

cout << temp->vertex << "\t";

}

t = t->next;

}

cout << endl;

temp = temp->next;

}

}

void graph::FEED\_UPLOADER(string& text, string& time, string name,user\* Main\_root)

{

gnode\* temp = head;

user\* used = NULL;

cout << "Your post reached till your followers .\n";

while (temp != NULL)

{

gnode\* t = temp->edge;

while (t != NULL)

{

if (t->vertex == name)

{

used = Main\_root->Root\_finder(Main\_root, temp->vertex);

used->Feed.push(used->Feed.head, text, time);

}

t = t->next;

}

cout << endl;

temp = temp->next;

}

}

Message\_stack.h

#pragma once

#include <iostream>

#include <string>

using namespace std;

struct msgnode

{

string message;

msgnode\* next;

msgnode(string message);

};

class Messages\_stack

{

private:

public:

string person\_name;

msgnode\* head;

Messages\_stack();

bool isempty();

void push(msgnode\*& head,string message);

void pop();

msgnode\* top();

void print(msgnode\* head);

~Messages\_stack();

};

Message\_stack.cpp

#include "Messages\_stack.h"

msgnode::msgnode(string message)

{

this->message = message;

this->next = nullptr;

}

Messages\_stack::Messages\_stack()

{

this->person\_name = "Empty";

head = nullptr;

}

bool Messages\_stack::isempty()

{

return head == nullptr;

}

void Messages\_stack::push(msgnode\*& head,string message)

{

msgnode\* new\_node = new msgnode(message);

new\_node->next = head;

head = new\_node;

}

void Messages\_stack::pop()

{

if (isempty()) return;

msgnode\* temp = head;

head = head->next;

delete temp;

}

msgnode\* Messages\_stack::top()

{

return head;

}

void Messages\_stack::print(msgnode\* head)

{

if (isempty())

{

cout << "This section is empty.\n";

return;

}

msgnode\* temp = head;

while (temp != nullptr)

{

cout << "Message: " << temp->message << '\n';

temp = temp->next;

}

}

Messages\_stack::~Messages\_stack()

{

while (head != nullptr)

{

msgnode\* temp = head->next;

delete head;

head = temp;

}

}

Queue.h

#pragma once

#include <iostream>

#include <string>

using namespace std;

struct qnode

{

string message;

qnode\* next;

qnode(string message);

};

class queue

{

private:

public:

void delete\_from\_center(qnode\*&, string&);

qnode\* head;

queue();

bool isempty();

void enqueue(qnode\*& head, string message);

void dequeue();

qnode\* front();

void print(qnode\* head);

~queue();

};

Queue.cpp

#include "queue.h"

qnode::qnode(string message)

{

this->message = message;

this->next = nullptr;

}

void queue::delete\_from\_center(qnode\*& head, string& target)

{

if (isempty()) return;

if (head->message == target)

{

qnode\* temp = head;

head = head->next;

delete temp;

return;

}

qnode\* current = head;

qnode\* prev = nullptr;

while (current != nullptr && current->message != target)

{

prev = current;

current = current->next;

}

if (current == nullptr) return; // Target not found

prev->next = current->next;

delete current;

}

queue::queue()

{

head = nullptr;

}

bool queue::isempty()

{

return head == nullptr;

}

void queue::enqueue(qnode\*& head, string message)

{

qnode\* new\_node = new qnode(message);

if (head == nullptr)

{

head = new\_node;

return;

}

qnode\* temp = head;

while (temp->next != nullptr)

{

temp = temp->next;

}

temp->next = new\_node;

}

void queue::dequeue()

{

if (isempty()) return;

qnode\* temp = head;

head = head->next;

delete temp;

}

qnode\* queue::front()

{

return head;

}

void queue::print(qnode\* head)

{

if (isempty())

{

cout << "This section is empty.\n";

return;

}

qnode\* temp = head;

while (temp != nullptr)

{

cout <<temp->message << '\n';

temp = temp->next;

}

}

queue::~queue()

{

while (head != nullptr)

{

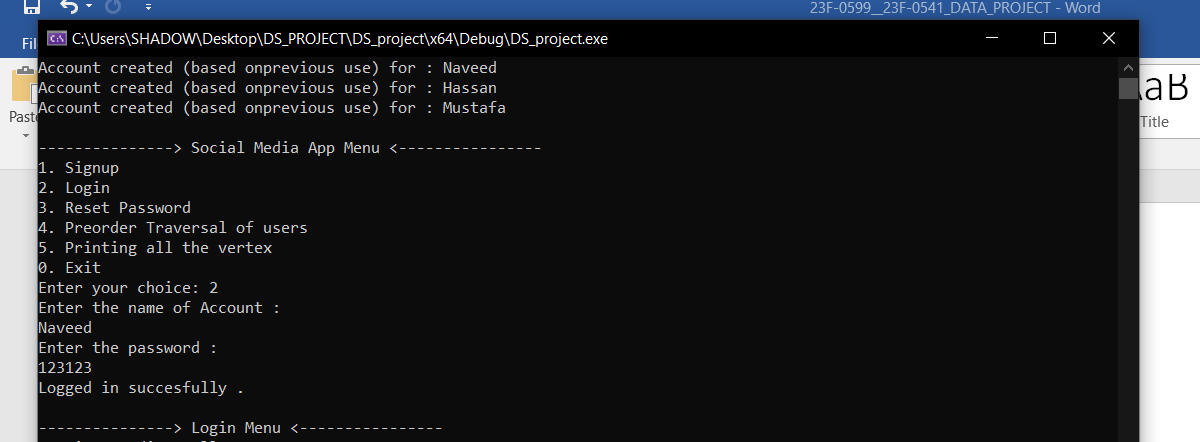
qnode\* temp = head->next;

delete head;

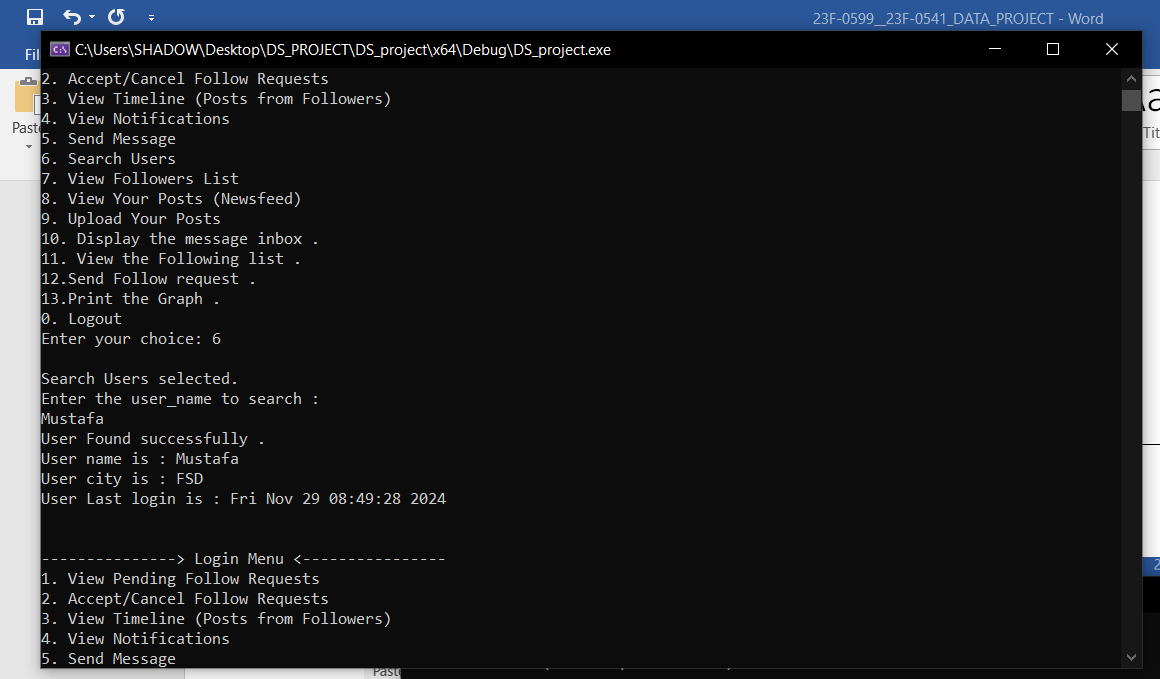
head = temp;

}

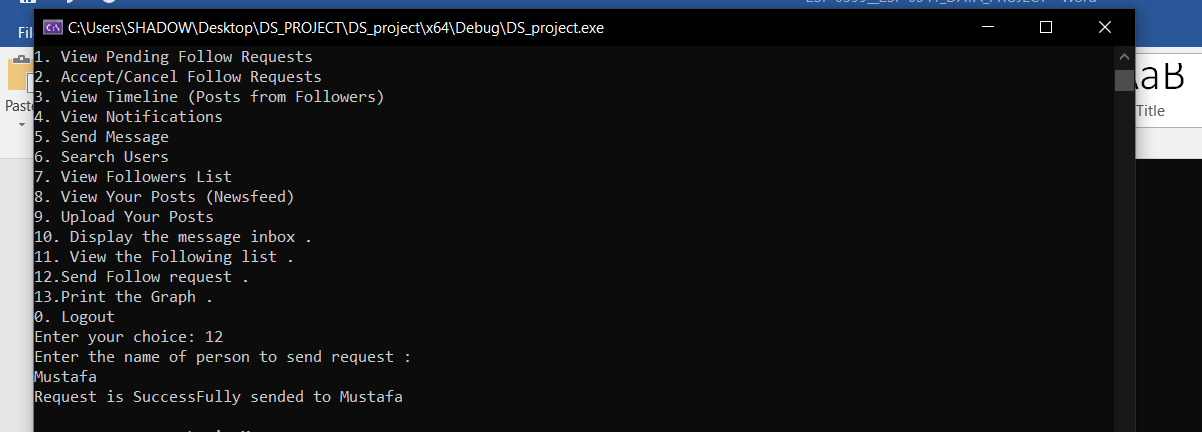
}

OUTPUTS:  


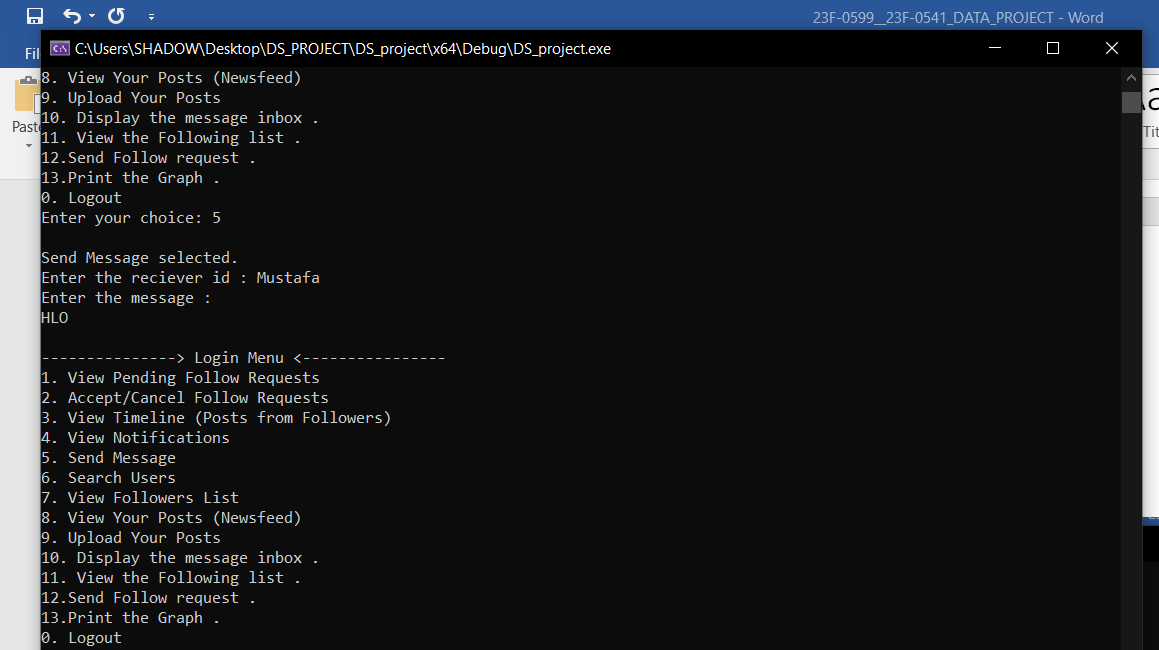
To check last login and to search a person



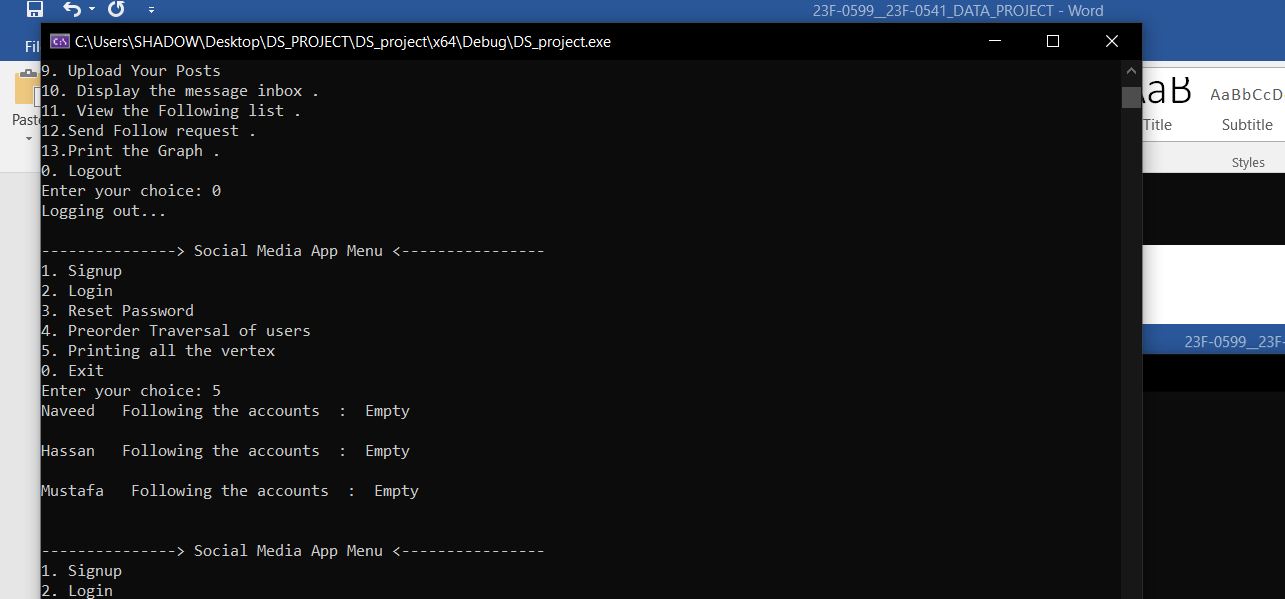
Sending Request



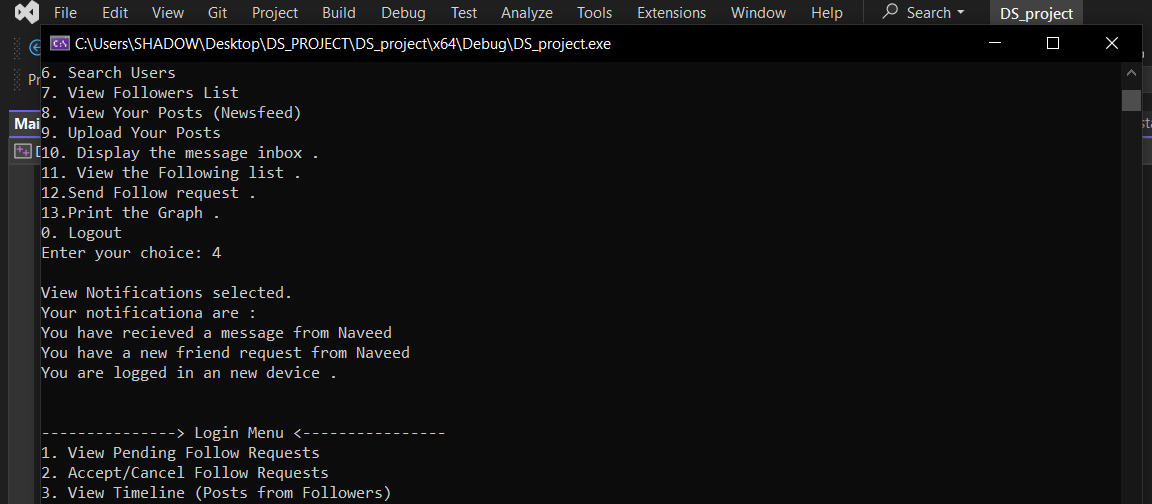
Sending message



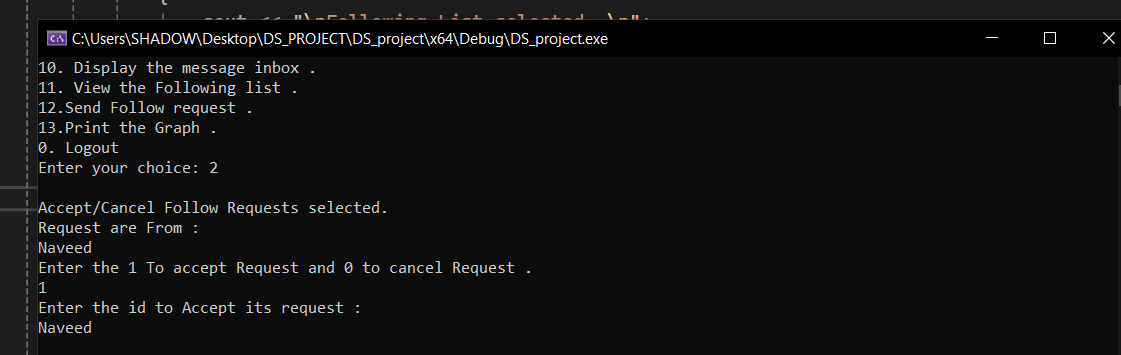
Printed the Graph



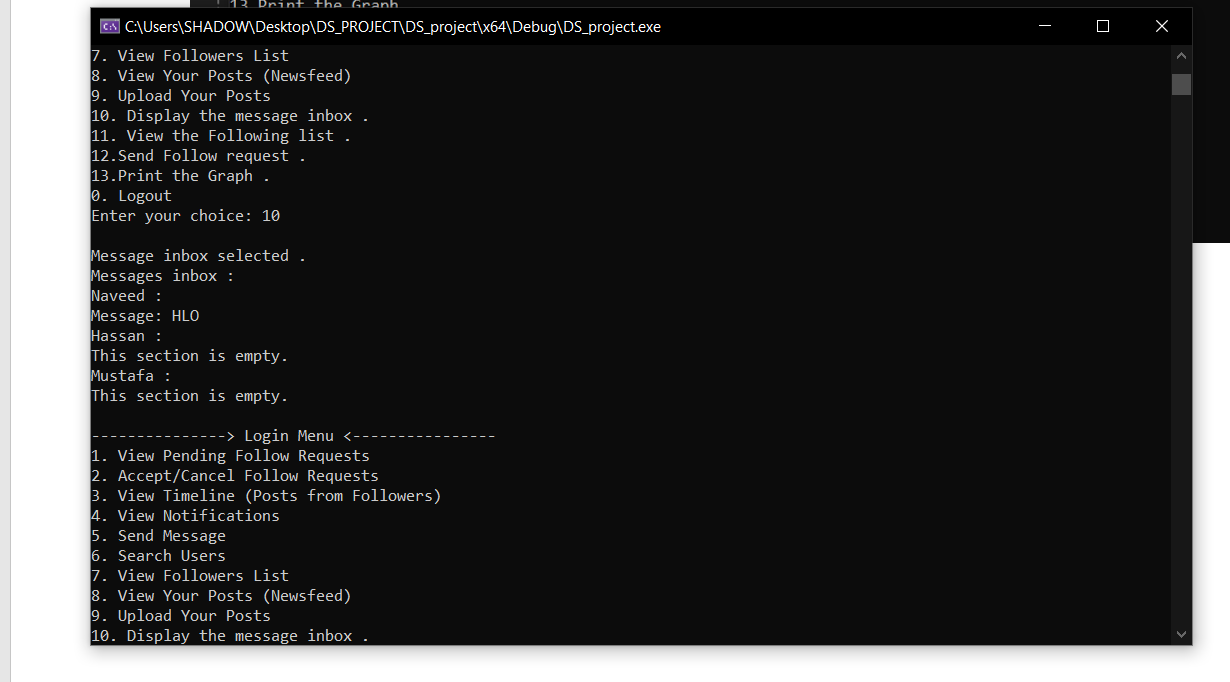
View Notification



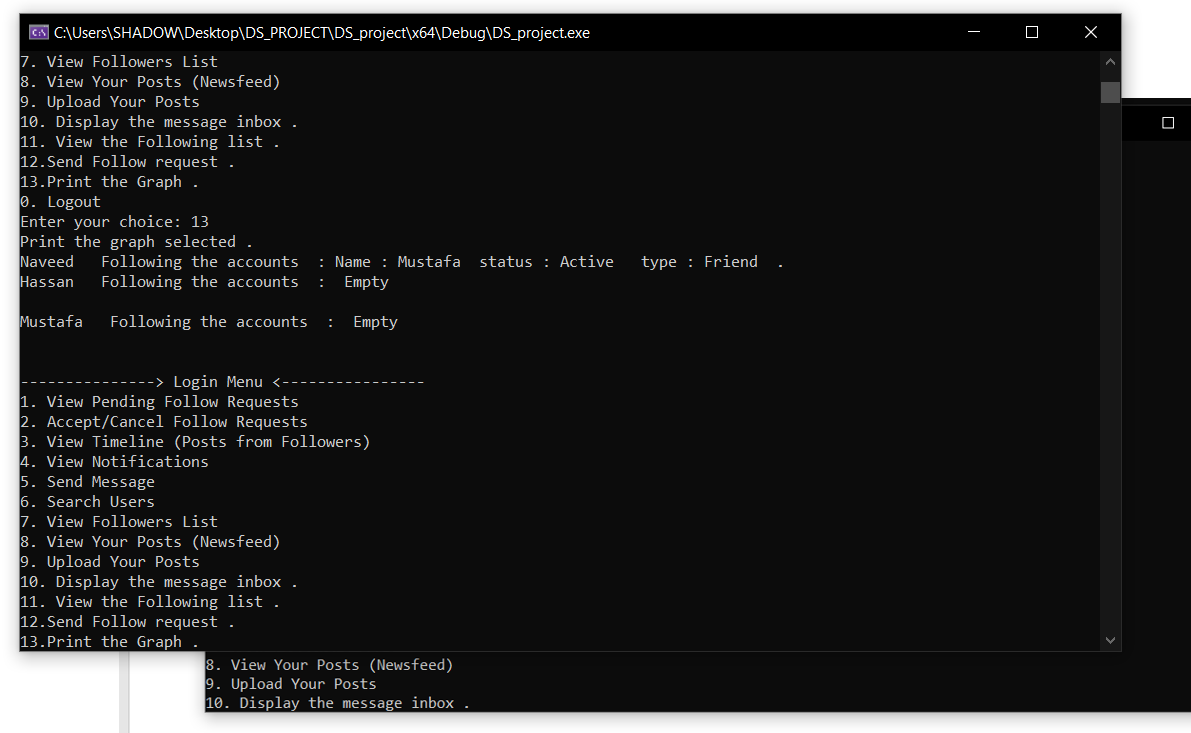
Accepting Request



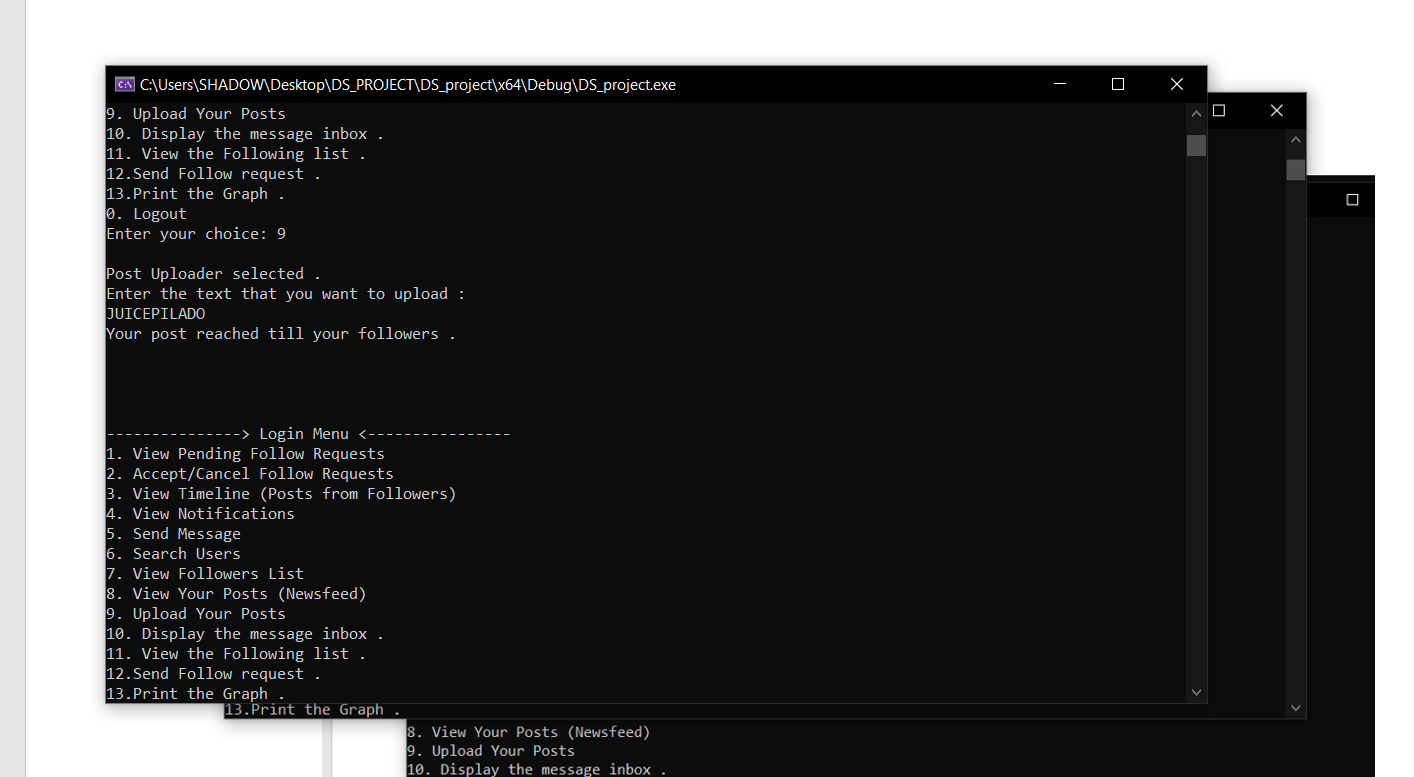
Message inbox For every Account



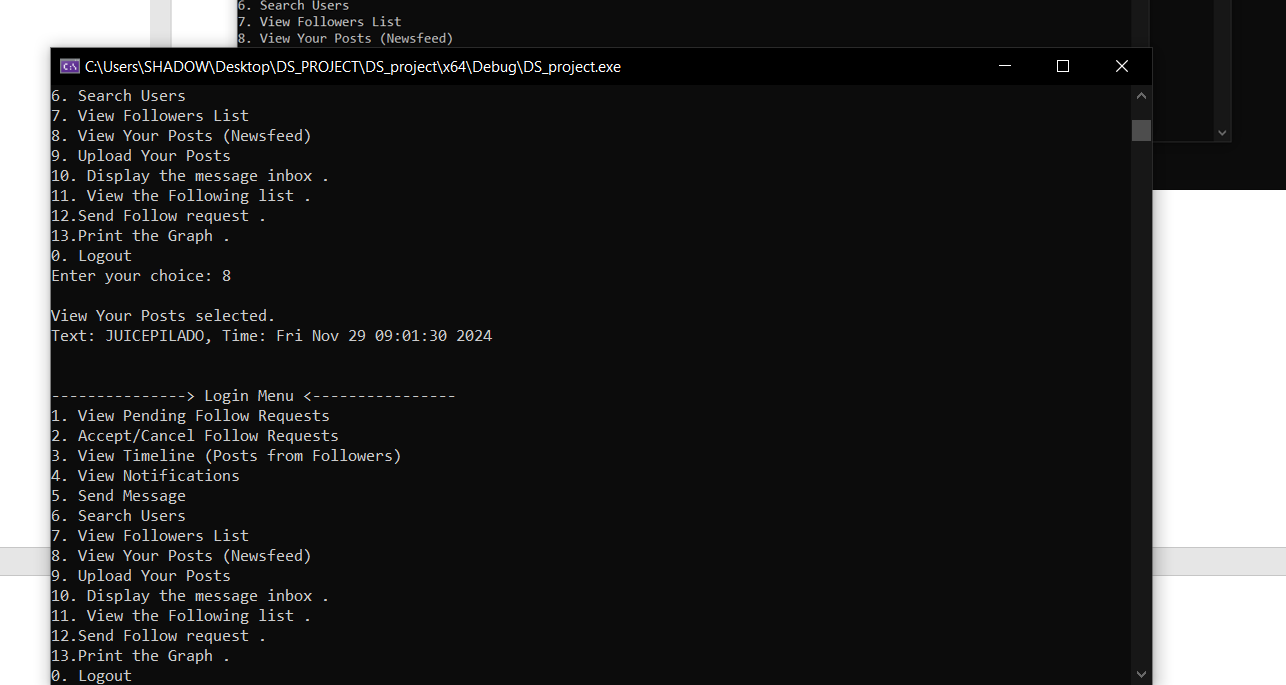
Graph After Naveed Followed Mustafa



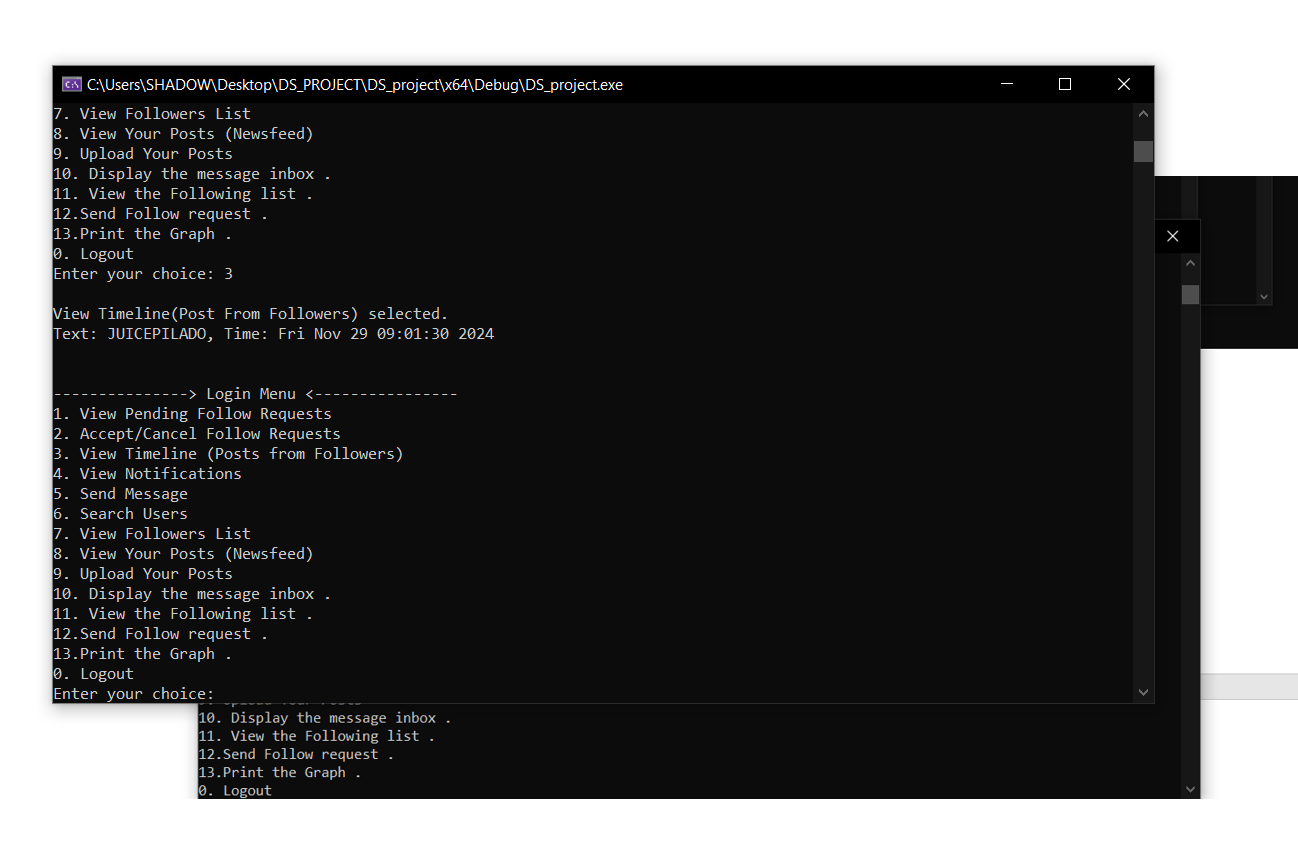
Posting the post

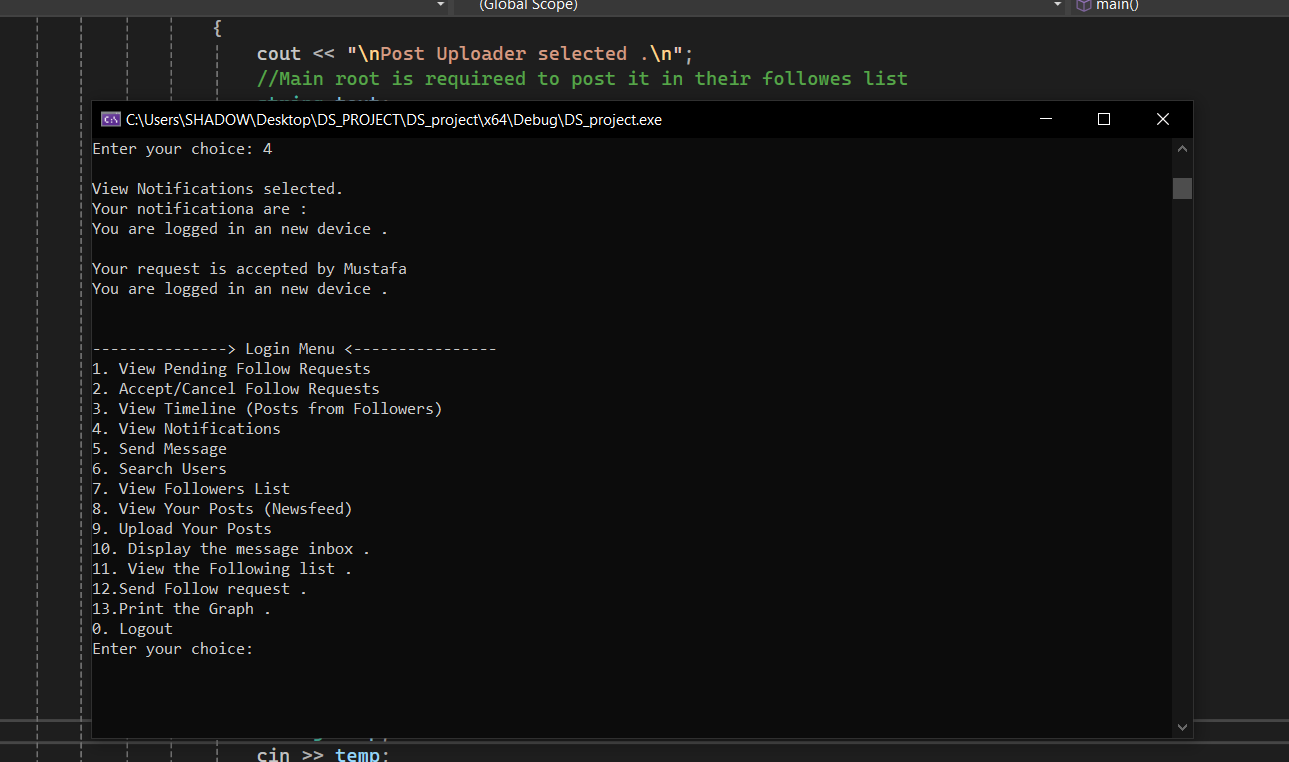


Viewing Your Own POSTS

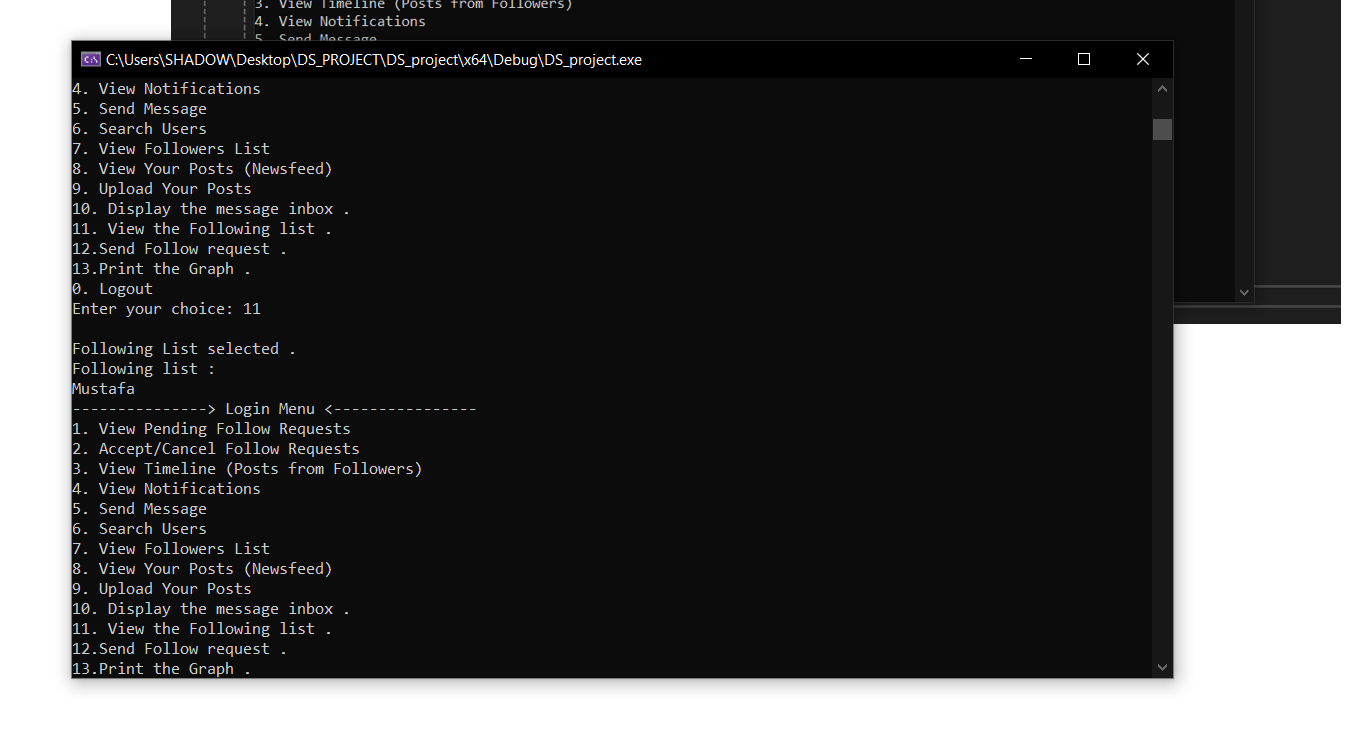


One post reached till its following

Accepted Friend Request Notification



Viwing Your Following List



PreOrder Traversal of Users

