Thoriso Dibatana EDUV4841116

ITSDA2 – Software Design

Formative Assessment: Assignment

Notes:

The program I wrote for question 4 is quite long and I have note included the header files for the class because it might be difficult to read on a word document so i have included links to give you password protected access to the one drive folder with all the files for the assignment. I have also included them in my GitHub so you can choose how you would like to access them. OneDrive Password: LwaziMaqoqa GitHub

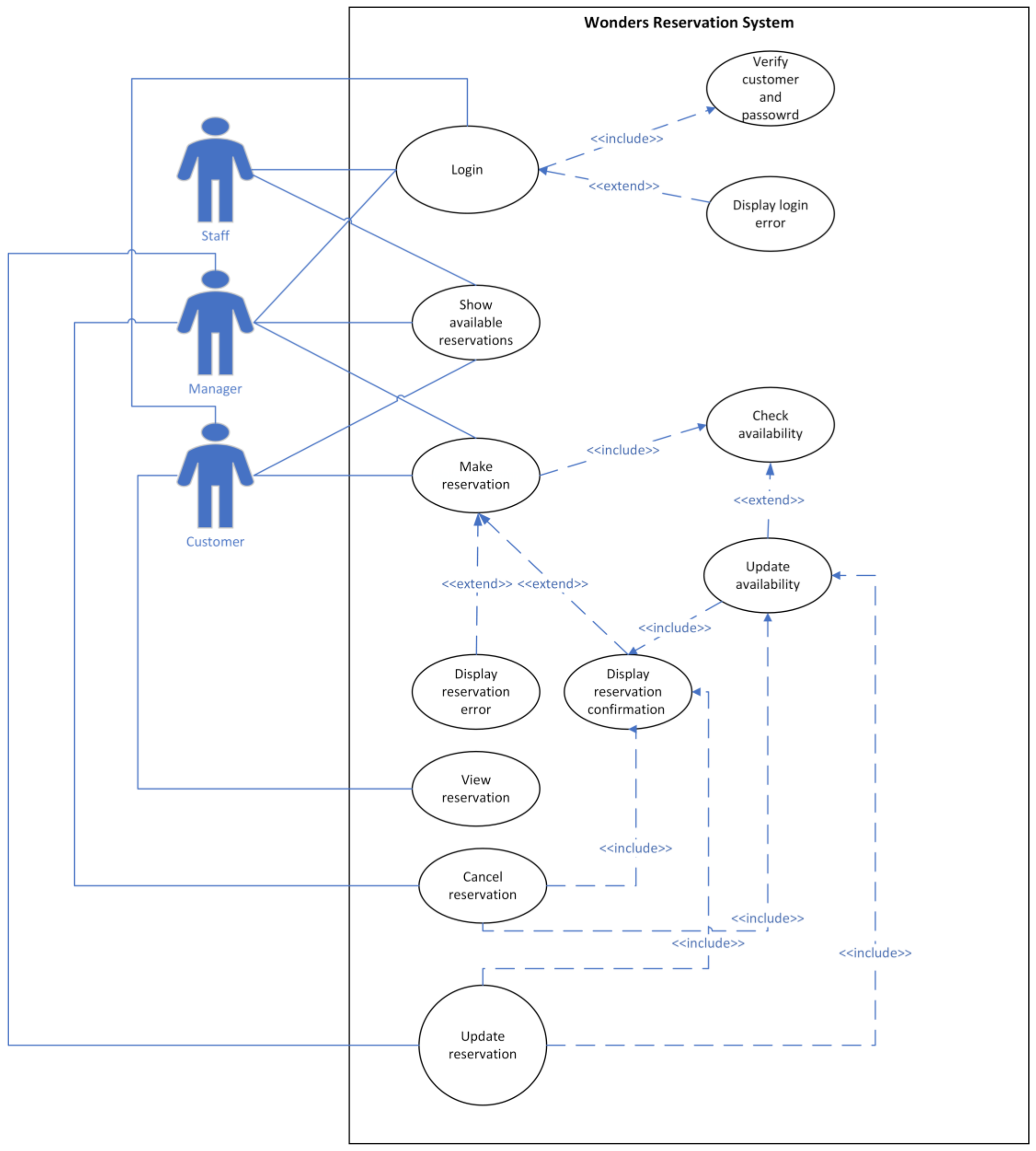
Question 1

a.
   A good design will help with maintaining and scaling the software throughout its lifecycle. Having the design process well documented will help communicate the scope of the software and its operations to parties that are concerned. It will help show the software will be built.
b. The waterfall process is forward-only progression of software development that consists of the following phases: Requirements, design, implementation, testing and maintenance. This model is a documentation-driven process with defined goals, milestones and deliverables for each development phase, like a traditional engineering process.
   The waterfall is true to its name and works just like a waterfall, with each phase flowing into the next. This makes it such that one phase cannot begin until one is completed sufficiently. With requirement gathering being the first phase, it is assumed that the requirements are correct and complete when the design phase begins. This works perfectly for the purpose of the Wonders Reservation System software development process as the requirements and objectives have been completely and clearly laid out for the system. This is important because it allows us to “flow” into the design process. Because software development has the characteristics of a wicked problem, the waterfall process helps to deliver a software product that will meet the system's requirements. The design phase is formally documented with recognized standards

because its defined timespan has little overlap with a timeframe for software construction.

Question 2

a.



b. **System**: Wonders Reservation System

Actors: Customers, staff, managers

Use Cases:

1. Login:

   This is meant for the users of the system to log in so that they can access their accounts and further privileges to do what they need to do with the system.

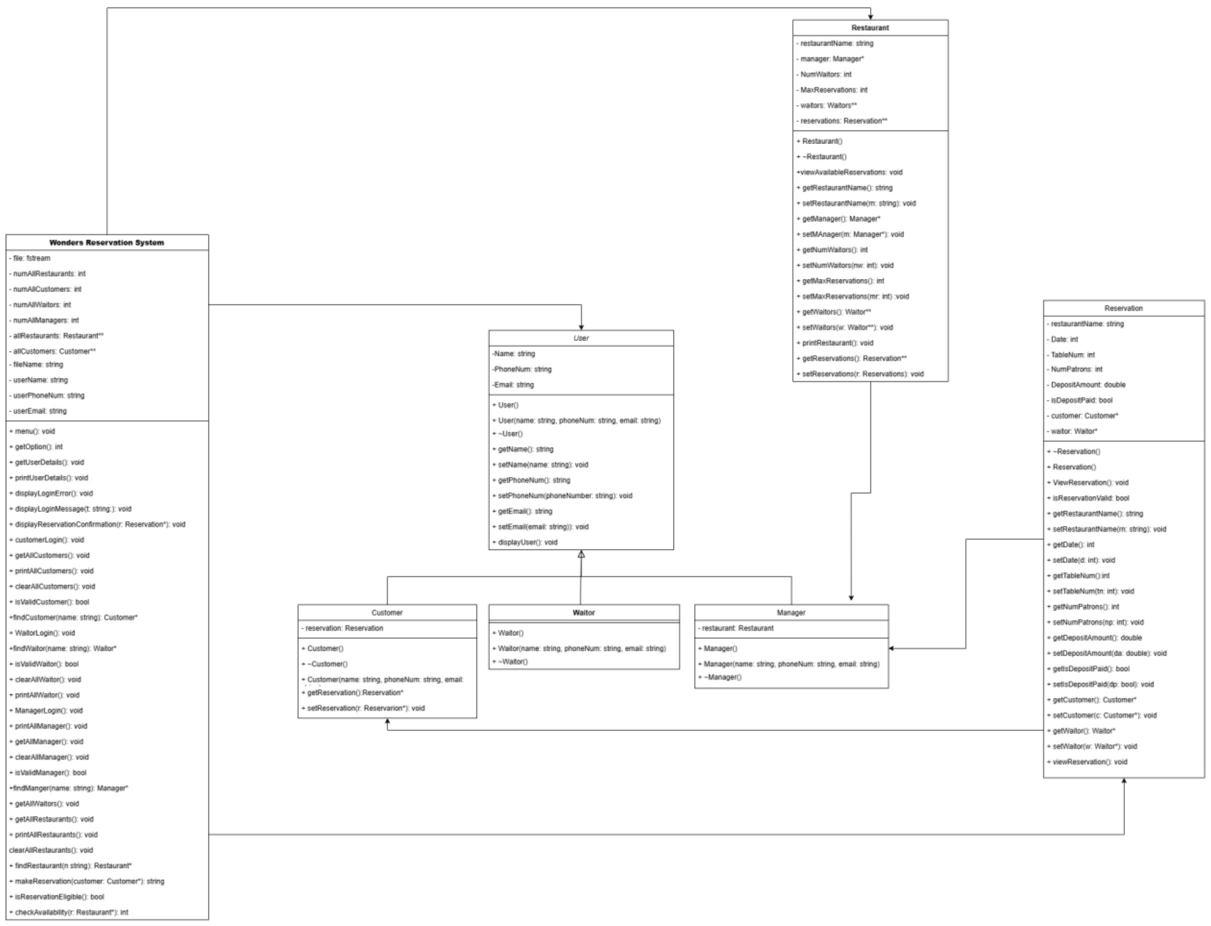
2. Show available reservations:

   This will allow any user of the system, from customers and staff members to managers, to view a roster of the reservations and available slots. This will help the staff members manage tables more efficiently. It will help customers quickly see if there are any available tables for when they want to make a reservation. It will help managers get a brief overview of how busy the restaurant is.

3. Verify customer and password:
   This is initiated by the system whenever a customer or manager enters their account details. This will verify if the password and email address are valid
4. Display login error:
   This will only be displayed by the system to the customer or manager enters invalid account details.
5. Make reservation:
   This can be initiated by either a manager or customer to make a reservation during an open slot.
6. Check availability:
   This will be conducted by the system whenever a customer or manager makes a booking. It will ensure that the selected slot is indeed available for reservation.
7. Update availability:
   An update to the reservation roster will be conducted by the system for the slot selected by the customer or the manager.
8. Display reservation confirmation:
   A confirmation will be sent and displayed to the customer or manager for the reservation they have just made.
9. Display reservation error:
   A reservation error will be displayed if the selected slot is unavailable for a reservation, if there is incomplete information for the reservation or was in issue with the deposit.
10. View reservation:
    This will allow the customer to view all the information pertaining to their reservation. It will allow the manager to get in-depth information on all the reservations made.
11. Cancel reservation:
    This will allow the manager to manage the reservations made on the system by giving the privilege to cancel reservations.
12. Update reservation:
    This will allow the manager to manage the reservations made on the system by giving the privilege to update and modify reservations.

Question 3

a. Link to class diagram



Wonders Reservation System
- file: fstream
- numAllRestaurants: int
- numAllCustomers: int
- numAllWaitors: int
- numAllManagers: int
- allRestaurants: Restaurant**
- allCustomers: Customer**
- fileName: string
- userName: string
- userPhoneNum: string
- userEmail: string
+ menu(): void
+ getOption(): int
+ getUserDetails(): void
+ printUserDetails(): void
+ displayLoginError(): void
+ displayLoginMessage(t: string:): void
+ displayReservationConfirmation(r: Reservation*): void
+ customerLogin(): void
+ getAllCustomers(): void
+ printAllCustomers(): void
+ clearAllCustomers(): void
+ isValidCustomer(): bool
+findCustomer(name: string): Customer*
+ WaitorLogin(): void
+findWaitor(name: string): Waitor*
+ isValidWaitor(): bool
+ clearAllWaitor(): void
+ printAllWaitor(): void
+ ManagerLogin(): void
+ printAllManager(): void
+ getAllManager(): void
+ clearAllManager(): void
+ isValidManager(): bool
+findManger(name: string): Manager*
+ getAllWaitors(): void
+ getAllRestaurants(): void
+ printAllRestaurants(): void
clearAllRestaurants(): void
+ findRestaurant(n string): Restaurant*
+ makeReservation(customer: Customer*): string
+ isReservationEligible(): bool
+ checkAvailability(r: Restaurant*): int
Restaurant
- restaurantName: string
- manager: Manager*
- NumWaitors: int
- MaxReservations: int
- waitors: Waitors**
- reservations: Reservation**
+ Restaurant()
+ ~Restaurant()
+viewAvailableReservations: void
+ getRestaurantName(): string
+ setRestaurantName(rn: string): void
+ getManager(): Manager*
+ setMAnager(m: Manager*): void
+ getNumWaitors(): int
+ setNumWaitors(nw: int): void
+ getMaxReservations(): int
+ setMaxReservations(mr: int) :void
+ getWaitors(): Waitor**
+ setWaitors(w: Waitor**): void
+ printRestaurant(): void
+ getReservations(): Reservation**
+ setReservations(r: Reservations): void
User
-Name: string
-PhoneNum: string
-Email: string
+ User()
+ User(name: string, phoneNum: string, email: string)
+ ~User()
+ getName(): string
+ setName(name: string): void
+ getPhoneNum(): string
+ setPhoneNum(phoneNumber: string): void
+ getEmail(): string
+ setEmail(email: string)): void
+ displayUser(): void
Customer
- reservation: Reservation
+ Customer()
+ ~Customer()
+ Customer(name: string, phoneNum: string, email:
+ getReservation():Reservation*
+ setReservation(r: Reservarion*): void
Waitor
+ Waitor()
+ Waitor(name: string, phoneNum: string, email: string)
+ ~Waitor()
Manager
- restaurant: Restaurant
+ Manager()
+ Manager(name: string, phoneNum: string, email: string)
+ ~Manager()
Reservation
- restaurantName: string
- Date: int
- TableNum: int
- NumPatrons: int
- DepositAmount: double
- isDepositPaid: bool
- customer: Customer*
- waitor: Waitor*
+ ~Reservation()
+ Reservation()
+ ViewReservation(): void
+ isReservationValid: bool
+ getRestaurantName(): string
+ setRestaurantName(rn: string): void
+ getDate(): int
+ setDate(d: int): void
+ getTableNum():int
+ setTableNum(tn: int): void
+ getNumPatrons(): int
+ setNumPatrons(np: int): void
+ getDepositAmount(): double
+ setDepositAmount(da: double): void
+ getIsDepositPaid(): bool
+ setIsDepositPaid(dp: bool): void
+ getCustomer(): Customer*
+ setCustomer(c: Customer*): void
+ getWaitor(): Waitor*
+ setWaitor(w: Waitor*): void
+ viewReservation(): void

b. Coupling is the degree of dependency of a module with other modules. This speaks to a module's interaction with other modules. This is achieved through decomposition and appropriate code separation.
Cohesion is the degree of interaction within a module and how related the tasks are. Cohesion makes maintain methods easier.
These not only improve performance but also allow for code reusability, improve clarity and readability.

Question 4

Link to OneDrive folder

Password: LwaziMaqoqa

Link to GitHub

WRS.CPP

```
#include <fstream>
#include "Customer.h"
#include "Waitor.h"
#include "Manager.h"
#include "Restaurant.h"
#include "Reservation.h"

using namespace std;

const double DEPOSIT_PP = 150.00;

// global varaiables
ifstream file;
Customer **allCustomers;
Waitor **allWaitors;
Manager **allManagers;
Restaurant **allRestaurants;
int numAllCustomers = 0, numAllWaitors = 0, numAllManagers = 0, numAllRestaurants
= 0;
string fileName, userName, userPhoneNum, userEmail;

// function prototypes
void menu();
int getOption();

// user
void getUserDetails();
void printUserDetails();
void displayLoginError();
void displayLoginMessage(string t);
void displayReservationConfirmation(Reservation *r);

// customer
void CustomerLogin();
void getAllCustomers();
void printallCustomers();
void clearallCustomers();
bool isValidCustomer();
Customer *findCustomer(string name);

// waitor
void WaitorLogin();
void getAllWaitors();
void printAllWaitors();
void clearAllWaitors();
bool isValidWaitor();
Waitor *findWaitor(string name);

// manager
void ManagerLogin();
void getAllManagers();
void printAllManagers();
void clearAllManagers();
bool isValidManager();
Manager *findManager(string name);

// restaurant
void getAllRestaurants();
void printAllRestaurants();
void clearAllRestaurants();
Restaurant *findRestaurant(string n);

// WRS
string MakeReservation(Customer *customer);
bool isReservationEligible();
int checkAvailability(Restaurant *r);

int main()
{
    int option;

    // read all textfile for a list of users and restaurants
    getAllCustomers();
    getAllWaitors();
    getAllManagers();
    getAllRestaurants();
    printAllRestaurants();

    // printUserDetails();

    menu();
    option = getOption();

    switch (option)
    {
    case 1:
        CustomerLogin();
        break;
    case 2:
        WaitorLogin();
        break;
    case 3:
        ManagerLogin();
        break;
    }

    clearallCustomers();
    clearAllWaitors();
    clearAllManagers();

    return 0;
}

// COMPLETE. WORKS
void menu()
{
    cout << "Wonders Reservation System" << endl;
    cout << "Login in as a: " << endl;
    cout << "1. Customer" << endl;
    cout << "2. Waitor" << endl;
    cout << "3. Manager" << endl;
    cout << "4. Exit" << endl;
}

// COMPLETE. WORKS
void getUserDetails()
{
    /*cout << "Enter the user's name: ";
    cin >> userName;
    cout << "Enter the user's phone number: ";
    cin >> userPhoneNum;

    cout << "Enter the user's email address: ";
    cin >> userEmail;*/
    userName = "Thoriso";
    userPhoneNum = "0636770440";
    userEmail = "thorisodibatana@gmail.com";
}

// COMPLETE. WORKS
void printUserDetails()
{
    cout << "User's name: " << userName << endl;
    cout << "User's phone number: " << userPhoneNum << endl;
    cout << "User's email: " << userEmail << endl;
}

// COMPLETE. WORKS
int getOption()
{
    int opt;

    cout << "Enter option: ";
    cin >> opt;

    return opt;
}

// COMPLETE. WORKS
void CustomerLogin()
{

    // getAllCustomers(fileName);

    // cout << numAllCustomers;

    // printallCustomers();

    if (isValidCustomer() == false)
    {
        displayLoginError();
        return;
    }

    Customer *customer = new Customer;
    customer->setName(userName);
    customer->setEmail(userEmail);
    customer->setPhoneNum(userPhoneNum);

    cout << endl;
    displayLoginMessage("Customer");
    cout << "Would you like to :" << endl;
    cout << "1. Make a reservation" << endl;
    cout << "2. View your reservation" << endl;

    //*/

    // clearallCustomers(customer);
}

// COMPLETE. WORKS
void WaitorLogin()
{

    // getAllWaitors(fileName);

    // printAllWaitors();

    if (isValidWaitor() == false)
    {
        displayLoginError();
        return;
    }

    cout << endl;
    displayLoginMessage("Waitor");
    cout << "Would you like to :" << endl;
    cout << "1. Make a reservation" << endl;
    cout << "2. View your resturant's reservations" << endl;

    // findWaitor("Thoriso")->displayUser();
```

```cpp
    clearAllWaitors();
}

// COMPLETE. WORKS
void ManagerLogin()
{

    // getAllManagers(fileName);

    // printAllManagers();

    if (isValidManager() == false)
    {
        displayLoginError();
        return;
    }

    cout << endl;
    displayLoginMessage("Manager");
    cout << "Would you like to :" << endl;
    cout << "1. Make a reservation" << endl;
    cout << "2. Cancel a reservation" << endl;
    cout << "3. Update a reservation" << endl;
    cout << "4. View your resturant's reservations" << endl;

    // findManager("Thoriso")->displayUser();

    getAllRestaurants(); // seg fault
    printAllRestaurants();

    clearAllManagers();
}

// COMPLETE. WORKS
void getAllCustomers()
{
    string line, name, phoneNum, email;
    int pos, i = 0;
    fileName = "Customers.txt";

    // clearCustomers();
    numAllCustomers = 0;
    file.open(fileName);

    if (file.fail())
    {
        cout << "There has been an error in opening the file as it does not seem to exist." << endl;
        return;
    }

    // READ THROUGH THE TEXTFILE TO GET THE NUMBER OF customers
    while (getline(file, line))
    {
        numAllCustomers++;
    }

    file.close();

    allCustomers = new Customer *[numAllCustomers];

    file.open(fileName);

    // READ THROUGH THE TEXTFILE TO POPULATE THE customers ARRAY
    while (getline(file, line))
    {
        pos = line.find(",", 0);
        name = line.substr(0, pos);
        line.erase(0, pos + 1);

        pos = line.find(",", 0);
        phoneNum = line.substr(0, pos);
        line.erase(0, pos + 1);

        email = line;

        allCustomers[i] = new Customer(name, phoneNum, email);
        i++;
    }

    file.close();
}
```

```cpp
// COMPLETE. WORKS
void printallCustomers()
{
    for (int i = 0; i < numAllCustomers; i++)
    {
        allCustomers[i]->displayUser();
        cout << endl;
    }
}

// COMPLETE. WORKS
void clearallCustomers()
{
    for (int i = 0; i < numAllCustomers; i++)
    {
        delete allCustomers[i];
    }
    delete[] allCustomers;
    allCustomers = nullptr;
}

// COMPLETE. WORKS
bool isValidCustomer()
{
    int i = 0;

    while (i < numAllCustomers)
    {
        if (allCustomers[i]->getEmail() != userEmail)
        {
            i++;
        }

        return true;
    }

    return false;
}

Customer *findCustomer(string name)
{
    int i = 0;
    while (i < numAllCustomers)
    {
        if (allCustomers[i]->getName() != n)
        {
            i++;
        }
        else
        {
            return allCustomers[i];
        }
    }

    return nullptr;
}

// COMPLETE. WORKS
void displayLoginError()
{
    cout << "You are not a registered user." << endl;
}

// COMPLETE. WORKS
void displayLoginMessage(string t)
{
    cout << "You are logged in as a " << t << endl;
    cout << "Welcome " << userName << endl;
}

// COMPLETE. WORKS
void getAllWaitors()
{
    string line, name, phoneNum, email;
    int pos, i = 0;
    fileName = "Waitors.txt";

    // clearCustomers();
    numAllWaitors = 0;

    file.open(fileName);

    if (file.fail())
    {
        cout << "There has been an error in opening the file as it does not seem
to exist." << endl;
        return;
    }

    // READ THROUGH THE TEXTFILE TO GET THE NUMBER OF customers
    while (getline(file, line))
    {
        numAllWaitors++;
    }

    file.close();

    allWaitors = new Waitor *[numAllWaitors];

    file.open(fileName);

    // READ THROUGH THE TEXTFILE TO POPULATE THE customers ARRAY
    while (getline(file, line))
    {
        pos = line.find(",", 0);
        name = line.substr(0, pos);
        line.erase(0, pos + 1);

        pos = line.find(",", 0);
        phoneNum = line.substr(0, pos);
        line.erase(0, pos + 1);

        email = line;

        allWaitors[i] = new Waitor(name, phoneNum, email);
        i++;
    }

    file.close();
}

// COMPLETE. WORKS
void printAllWaitors()
{
    for (int i = 0; i < numAllWaitors; i++)
    {
        allWaitors[i]->displayUser();
        cout << endl;
    }
}

// COMPLETE. WORKS
void clearAllWaitors()
{
    for (int i = 0; i < numAllWaitors; i++)
    {
        delete allWaitors[i];
    }
    delete[] allWaitors;
    allWaitors = nullptr;
}

// COMPLETE. WORKS
bool isValidWaitor()
{
    int i = 0;

    while (i < numAllWaitors)
    {
        if (allWaitors[i]->getEmail() != userEmail)
        {
            i++;
        }

        return true;
    }

    return false;
}

// COMPLETE. WORKS
void getAllManagers()
{
    string line, name, phoneNum, email;
    int pos, i = 0;
    fileName = "Managers.txt";

    // clearCustomers();
    numAllManagers = 0;
    file.open(fileName);

    if (file.fail())
    {
        cout << "There has been an error in opening the file as it does not seem
to exist." << endl;
        return;
    }

    // READ THROUGH THE TEXTFILE TO GET THE NUMBER OF customers
    while (getline(file, line))
    {
        numAllManagers++;
    }

    file.close();

    allManagers = new Manager *[numAllManagers];

    file.open(fileName);

    // READ THROUGH THE TEXTFILE TO POPULATE THE customers ARRAY
    while (getline(file, line))
    {
        pos = line.find(",", 0);
        name = line.substr(0, pos);
        line.erase(0, pos + 1);

        pos = line.find(",", 0);
        phoneNum = line.substr(0, pos);
        line.erase(0, pos + 1);

        email = line;

        allManagers[i] = new Manager(name, phoneNum, email);
        i++;
    }

    file.close();
}
```

```cpp
// COMPLETE. WORKS
void printAllManagers()
{
    for (int i = 0; i < numAllManagers; i++)
    {
        allManagers[i]->displayUser();
        cout << endl;
    }
}

// COMPLETE. WORKS
void clearAllManagers()
{
    for (int i = 0; i < numAllManagers; i++)
    {
        delete allManagers[i];
    }
    delete[] allManagers;
    allManagers = nullptr;
}

// COMPLETE. WORKS
bool isValidManager()
{
    int i = 0;

    while (i < numAllManagers)
    {
        if (allManagers[i]->getEmail() != userEmail)
        {
            i++;
        }

        return true;
    }

    return false;
}

// COMPLETE. WORKS
Waitor *findWaitor(string n)
{
    int i = 0;
    while (i < numAllWaitors)
    {
        if (allWaitors[i]->getName() != n)
        {
            i++;
        }
        else
        {
            return allWaitors[i];
        }
    }

    return nullptr;
}

// COMPLETE. WORKS
Manager *findManager(string n)
{
    int i = 0;

    while (i < numAllManagers)
    {
        if (allManagers[i]->getName() != n)
        {
            i++;
        }
        else
        {
            return allManagers[i];
        }
    }

    return nullptr;
}

// WORKS
void getAllRestaurants()
{
    string line, restaurantName, managerName, waitorName;
    int pos, numWaitors, i = 0;
    fileName = "Restaurants.txt";
    Waitor **waitors = nullptr;
    file.open(fileName);

    if (file.fail())
    {
        cout << "There has been an error in opening the file as it does not seem
to exist." << endl;
        return;
    }

    // READ THROUGH THE TEXTFILE TO GET THE NUMBER OF Restaurants
    while (getline(file, line))
    {
        numAllRestaurants++;
    }

    file.close();

    allRestaurants = new Restaurant *[numAllRestaurants];

    file.open(fileName);

    while (i < numAllRestaurants) // getline(file, line))
    {
        getline(file, line);
        pos = line.find(",", 0);
        restaurantName = line.substr(0, pos);
        line.erase(0, pos + 1);

        pos = line.find(",", 0);
        managerName = line.substr(0, pos);
        line.erase(0, pos + 1);

        pos = line.find(",", 0);
        numWaitors = stoi(line.substr(0, pos)); // convert to string
        line.erase(0, pos + 1);

        waitors = new Waitor *[numWaitors];

        for (int j = 0; j < numWaitors - 1; j++)
        {
            pos = line.find(",", 0);
            waitorName = line.substr(0, pos);
            line.erase(0, pos + 1);

            waitors[j] = new Waitor();
            waitors[j]->setName(findWaitor(waitorName)->getName());
            waitors[j]->setPhoneNum(findWaitor(waitorName)->getPhoneNum());
            waitors[j]->setEmail(findWaitor(waitorName)->getEmail());
        }

        int index = numWaitors - 1;
        if (i == numAllRestaurants - 1)
        {
            waitorName = line;
        }
        else
        {
            waitorName = line.substr(0, line.length() - 1);
        }

        waitors[index] = new Waitor();
        waitors[index]->setName(findWaitor(waitorName)->getName());
        waitors[index]->setPhoneNum(findWaitor(waitorName)->getPhoneNum());
        waitors[index]->setEmail(findWaitor(waitorName)->getEmail());

        allRestaurants[i] = new Restaurant();

        allRestaurants[i]->setRestaurantName(restaurantName);
        allRestaurants[i]->setManager(findManager(managerName));
        allRestaurants[i]->setNumWaitors(numWaitors);
        allRestaurants[i]->setMaxReservations(numWaitors);
        allRestaurants[i]->setWaitors(waitors);
        allRestaurants[i]->setReservations();

        // create an empty array of reservations
        // allRestaurants[i].

        i++;
        for (int j = 0; j < numWaitors; j++)
        {
            delete waitors[j];
        }
        delete[] waitors;
        waitors = nullptr;
```

```
    }

    file.close();
}

// COMPLETE. WORKS
void printAllRestaurants()
{
    for (int i = 0; i < numAllRestaurants; i++)
    {
        allRestaurants[i]->printRestaurant();
        cout << endl;
    }
}

void clearAllRestaurants()
{
    for (int i = 0; i < numAllRestaurants; i++)
    {
        delete allRestaurants[i];
    }
    delete[] allRestaurants;
    allRestaurants = nullptr;
}

// COMPLETE. WORKS
Restaurant *findRestaurant(string n)
{
    int i = 0;

    while (i < numAllRestaurants)
    {
        if (allRestaurants[i]->getRestaurantName() != n)
        {
            i++;
        }
        else
        {
            return allRestaurants[i];
        }
    }

    return nullptr;
```

```
}

bool isReservationEligible(string restaurantChoice, string waitorChoice, int
date, bool isDepositPaid)
{
    if (findRestaurant(restaurantChoice) == nullptr)
    {
        cout << restaurantChoice << "is not a restaurant you can make a
reservation at." << endl;
        return false;
    }

    if (findWaitor(waitorChoice) == nullptr)
    {
        cout << waitorChoice << "is not a waitor that works at the restaurant."
<< endl;
        return false;
    }

    if (date < 0 || date > 31)
    {
        cout << date << "is invalid." << endl;
        return false;
    }

    if (!isDepositPaid)
    {
        cout << "You have not paid the deposit." << endl;
        return false;
    }

    return true;
}

void displayReservationConfirmation(Reservation *r)
{
    cout << "You have successfuly made a reservation." << endl;
    r->viewReservation();
}

string customerMakeReservation(Customer *customer)
{
    string restaurantChoice, waitorChoice, result;
    int date, tableNum, numPatrons;
    double depositAmount;
    bool isDepositPaid, isReservationEligible;

    cout << "Available restaurants: " << endl;
    for (int i = 0; i < numAllRestaurants; i++)
    {
        cout << "\t" << i + 1 << allRestaurants[i]->getRestaurantName() << endl;
    }
    cout << "Enter the name of the restaurant: ";
    cin >> restaurantChoice;

    Restaurant *ptrRestaurant = findRestaurant(restaurantChoice);
    Waitor **ptrWaitors = ptrRestaurant->getWaitors();

    cout << "Available waitors: " << endl;
    for (int i = 0; i < ptrRestaurant->getNumWaitors(); i++)
    {
        if (ptrRestaurant->getReservations()[i] == nullptr)
        {
            cout << "\t" << i + 1 << ptrWaitors[i]->getName() << endl;
        }
    }
    cout << "Enter the name of the waitor you would preffer: ";
    cin >> waitorChoice;

    cout << "Enter date: ";
    cin >> date;

    cout << "Enter number of people that will be present: ";
    cin >> numPatrons;

    depositAmount = tableNum * DEPOSIT_PP;

    cout << "The deposit amount is " << depositAmount << "\nHave you paid the
deposit.\nEnter true or false: ";
    cin >> isDepositPaid;

    if (!isReservationEligible(restaurantChoice, waitorChoice, date,
isDepositPaid))
    {
        return "The information entered is not eligible for an appropriate
reservation";
```

```
    }

    for (int i = 0; i < ptrRestaurant->getNumWaitors(); i++)
    {
        if (waitorChoice == findWaitor(waitorChoice).getName())
        {
            tableNum = i;
        }
    }

    Reservation *reservation = new Reservation;
    reservation->setRestaurantName(restaurantChoice);
    reservation->setDate(date);
    reservation->setTableNum(tableNum);
    reservation->setNumPatrons(numPatrons);
    reservation->setDepositAmount(depositAmount);
    reservation->setIsDepositPaid(isDepositPaid);
    reservation->setWaitor(findWaitor(waitorChoice));
    reservation->setCustomer(customer);

    displayReservationConfirmation();
    // isRese
    // assign table num to waitor index
}
```

User.cpp

```
#include "User.h"
using namespace std;

User::User()
{
    this->name = "";
    this->phoneNum = "";
    this->email = "";
}

User::User(string n, string pn, string em)
{
```

```cpp
    this->name = n;
    this->phoneNum = pn;
    this->email = em;
}

User::~User()
{
    // cout << "User " << this->name << " has been deleted." << endl;
}

string User::getName()
{
    return this->name;
}

void User::setName(string n)
{
    this->name = n;
}

string User::getPhoneNum()
{
    return this->phoneNum;
}

void User::setPhoneNum(string pn)
{
    this->phoneNum = pn;
}

string User::getEmail()
{
    return this->email;
}

void User::setEmail(string em)
{
    this->email = em;
}

// helper function // COMPLETE. WORKS
void User::displayUser()
{
    cout << "Name: " << getName() << endl;
    cout << "Phone Number: " << getPhoneNum() << endl;
    cout << "Email: " << getEmail() << endl;
}
```

Customer.cpp

```
#include "Customer.h"

using namespace std;

Customer::Customer()
{
    this->name = "";
    this->phoneNum = "";
    this->email = "";
    this->reservation = nullptr;
}

Customer::Customer(string n, string pn, string em)
{
    this->name = n;
    this->phoneNum = pn;
    this->email = em;
    this->reservation = nullptr;
}

// COMPLETE. WORKS
Customer::~Customer() // DESTRUCTOR
{
    delete reservation;
    reservation = nullptr;

    // cout << "Customer " << this->name << " has been deleted." << endl;
}

Reservation *Customer::getReservation()
{
    return reservation;
}

void Customer::setReservation(Reservation *r)
```

```
{
    reservation = new Reservation();
    reservation->setRestaurantName(r->getRestaurantName());
    reservation->setDate(r->getDate());
    reservation->setTableNum(r->getTableNum());
    reservation->setNumPatrons(r->getNumPatrons());
    reservation->setDepositAmount(r->getDepositAmount());
    reservation->setIsDepositPaid(r->getIsDepositPaid());
    reservation->setWaitor(r->getWaitor());
    reservation->setCustomer(this);
}
```

Waitor.cpp

```
#include "Waitor.h"

using namespace std;

Waitor::Waitor()
{
    this->name = "";
    this->phoneNum = "";
    this->email = "";
}

Waitor::Waitor(string n, string pn, string em)
{
    this->name = n;
    this->phoneNum = pn;
    this->email = em;
}

Waitor::~Waitor() // DESTRUCTOR
{
    // cout << "Waitor " << this->name << " has been deleted." << endl;
}
```

Manager.cpp

```
#include "Manager.h"
```

```cpp

using namespace std;

Manager::Manager()
{
    this->name = "";
    this->phoneNum = "";
    this->email = "";
    // this->restaurantName = nullptr;
}

Manager::Manager(string n, string pn, string em)
{
    this->name = n;
    this->phoneNum = pn;
    this->email = em;
}

Manager::~Manager() // DESTRUCTOR
{
    // cout << "Manager " << this->name << " has been deleted." << endl;
}


```

Restaurant.cpp

```cpp
#include "Restaurant.h"

using namespace std;

Restaurant::Restaurant()
{
    restaurantName = "";
    manager = nullptr;
    numWaitors = 0;
    MaxReservations = 0;
    waitors = nullptr;
    reservations = nullptr;
}

Restaurant::~Restaurant()
{
    delete manager;
    manager = nullptr;

    for (int i = 0; i < numWaitors; i++)
    {
        delete waitors[i];
    }
    delete[] waitors;
    waitors = nullptr;

    for (int i = 0; i < MaxReservations; i++)
    {
        delete reservations[i];
    }
    delete[] reservations;
    reservations = nullptr;
}

string Restaurant::getRestaurantName()
{
    return restaurantName;
}

void Restaurant::setRestaurantName(string rn)
{
    restaurantName = rn;
}

Manager *Restaurant::getManager()
{
    return manager;
}

void Restaurant::setManager(Manager *m)
{
    manager = new Manager();
    this->manager->setName(m->getName());
    this->manager->setEmail(m->getEmail());
    this->manager->setPhoneNum(m->getPhoneNum());
}

int Restaurant::getNumWaitors()
{
    return numWaitors;
```

```cpp
}

void Restaurant::setNumWaitors(int nw)
{
    this->numWaitors = nw;
}

int Restaurant::getMaxReservations()
{
    return MaxReservations;
}

void Restaurant::setMaxReservations(int mr)
{
    this->MaxReservations = mr;
}

Waitor **Restaurant::getWaitors()
{
    return waitors;
}

void Restaurant::setWaitors(Waitor **w) // create deep copy
{
    waitors = new Waitor *[numWaitors];

    for (int i = 0; i < numWaitors; i++)
    {
        waitors[i] = new Waitor();
        waitors[i]->setName(w[i]->getName());
        waitors[i]->setPhoneNum(w[i]->getPhoneNum());
        waitors[i]->setEmail(w[i]->getEmail());
    }
}

void Restaurant::printRestaurant()
{
    cout << "Restaurant's information:" << endl;
    cout << "Restaurant Name: " << getRestaurantName() << endl;
    cout << "Manager's information:" << endl;
    getManager()->displayUser();
    cout << "Number of waitors: " << getNumWaitors() << endl;
    cout << "Max number of reservations: " << getMaxReservations() << endl;
    cout << "Watoirs' information:" << endl;
    for (int i = 0; i < getNumWaitors(); i++)
    {
        waitors[i]->displayUser();
    }
    cout << "Reservations information:" << endl;
    for (int i = 0; i < getMaxReservations(); i++)
    {
        cout << i + 1 << ".";
        if (reservations[i] == nullptr)
            cout << "Available" << endl;
        else
            reservations[i]->viewReservation();
    }
}

Reservation **Restaurant::getReservations()
{
    return reservations;
}

void Restaurant::setReservations()
{
    reservations = new Reservation *[MaxReservations];

    for (int i = 0; i < MaxReservations; i++)
    {
        reservations[i] = nullptr;
    }
}
```

Reservation.cpp

```
#include "Reservation.h"

using namespace std;

Reservation::Reservation()
{
    restaurantName = "";
    date = 0;
    tableNum = 0;
    numPatrons = 0;
    depositAmount = 0.00;
    isDepositPaid = false;
    customer = nullptr;
    waitor = nullptr;
}

Reservation::~Reservation()
{
    delete customer;
    delete waitor;
    customer = nullptr;
    waitor = nullptr;
}

string Reservation::getRestaurantName()
{
    return restaurantName;
}

void Reservation::setRestaurantName(string rn)
{
    restaurantName = rn;
}

int Reservation::getDate()
{
    return date;
}

void Reservation::setDate(int d)
{
    date = d;
}

int Reservation::getTableNum()
{
    return tableNum;
}

void Reservation::setTableNum(int tn)
{
    tableNum = tn;
}

int Reservation::getNumPatrons()
{
    return numPatrons;
}

void Reservation::setNumPatrons(int p)
{
    numPatrons = p;
}

double Reservation::getDepositAmount()
{
    return depositAmount;
}

void Reservation::setDepositAmount(double da)
{
    depositAmount = da;
}

bool Reservation::getIsDepositPaid()
{
    return isDepositPaid;
}

void Reservation::setIsDepositPaid(bool dp)
{
    isDepositPaid = dp;
}

Customer *Reservation::getCustomer()
{
    return customer;
}

void Reservation::setCustomer(Customer *c)
{
    customer = new Customer();
    customer->setName(c->getName());
    customer->setPhoneNum(c->getPhoneNum());
```

```cpp
    customer->setEmail(c->getEmail());
}

Waitor *Reservation::getWaitor()
{
    return waitor;
}

void Reservation::setWaitor(Waitor *w)
{
    waitor = new Waitor();
    waitor->setName(w->getName());
    waitor->setPhoneNum(w->getPhoneNum());
    waitor->setEmail(w->getEmail());
}

void Reservation::viewReservation()
{
    cout << "Reservation information" << endl;
    cout << "\tRestaurant name: " << getRestaurantName() << endl;
    cout << "\tDate: " << getDate() << endl;
    cout << "\tTable number: " << getTableNum() << endl;
    cout << "\tNumber of patrons: " << getNumPatrons() << endl;
    cout << "\tDeposit amount: " << getDepositAmount() << endl;
    cout << "\t:Is deposit paid " << getIsDepositPaid() << endl;
    cout << "Customer information" << endl;
    cout << "\tName: " << customer->getName() << endl;
    cout << "\tPhone number: " << customer->getPhoneNum() << endl;
    cout << "\tEmail: " << customer->getEmail() << endl;
    cout << "Waitor information" << endl;
    cout << "\tName: " << waitor->getName() << endl;
}
```

Makefile

```
WRS:            User.o Customer.o Waitor.o Manager.o Restaurant.o Reservation.o
WRS.cpp
                g++ -g -static User.o Customer.o Waitor.o Manager.o Restaurant.o
Reservation.o WRS.cpp -o WRS

User.o:         User.cpp
                g++ -static -c User.cpp
```

```
Customer.o:         Customer.cpp
                g++ -static -c Customer.cpp

Waitor.o:       Waitor.cpp
                g++ -static -c Waitor.cpp

Manager.o:      Manager.cpp
                g++ -static -c Manager.cpp

Restaurant.o:   Restaurant.cpp
                g++ -static -c Restaurant.cpp

Reservation.o:  Reservation.cpp
                g++ -static -c Reservation.cpp

run:
                ./WRS

clean:          rm -f *.o WRS

tar:            tar -cvz *.h *.cpp *.txt makefile WRS.tar.gz
```