

## Mini Project

## Phase 2

### Cyber Cafe Management System

### Team Sharda

#### C++ Batch Group-1

#### Team Members

1.Akash Shikhare

2.Diptajit Ghosh

3.Hrushikesh Ahire

4.Kadir Sheikh

5.Pranjal Enchalwar

## Acknowledgement

On the occasion of presenting this project report, we wish to express our deep and profound feelings of gratitude to a number of persons who have contributed to the completion of this project.

First of all we express our deep gratitude to Bhimashankar Gavkare, the supreme guide, for bestowing his blessings upon us in our entire endeavor.

We are grateful to Mallika Mulkey and Jayati Munot for providing the opportunity for the development of our project.

We express our heartfelt gratitude to everyone mentioned for rendering all possible help and support during the development, implementation and presentation of the project.

We are extremely grateful for everyone's valuable guidance and help. We had a great learning experience through the journey of this project development.

## Index

S.no	Topic	Page no.
1	Milestone 1	4
2	Milestone 2	8
3	Milestone 3	20
4	Milestone 4	70
5	Milestone 5	90

# Cyber Cafe Management System

Group-1(Sharda)

Milestone 1

## Problem Statement:

Usually, cybercafé maintain their daily records manually. For further reference, they store all the working hours, usernames, user charges, and addresses daily in the record book. As the manual method is very time-consuming and hectic which sometimes leads to data loss, even processing is not up to the mark which creates inefficiency. Report generation becomes a very tedious and error-prone process.

The purpose of this project is to manage various cybercafé activities which can be maintained where we collect all the records of customers. Based on usage, charges will be calculated. The owner can add the number of computers, create passwords for secure login purposes, manage the list of customers as well as computers. At the end of the day, the owner will be able to calculate total income.

## Objective:

- Create an automated system that will be used to manage and store information about the cyber café's functioning.
- The owner will allocate an idle computer to a customer from the list of computers.
- The owner will be able to keep track of all his customers and he can calculate his income on a daily basis.

- The owner can manage and manipulate all the computers and all the related information.
- Registered customers can log in through the user id and password.
- Customers can view their session charge, profile as well as computer details.
- Customers can edit their profile.

### Functions to be implemented:

#### Owner Functions

- 1) Owner login
- 2) New owner registration
- 3) Allocate idle computer
- 4) Show list of all customers
- 5) Update customer
- 6) Delete customer
- 7) Search customer
- 8) Charges per customer
- 9) Income per day
- 10) Add new computer

11) Show computer list

12) Edit computer

13) Delete computer

14) Search computer

15) Update owner profile

### Customer Functions

1) Customer login

2) Customer registration

3) Charges as per duration

4) Session Time

5) Update profile

6) Customer log out

### Programming skills:

- C++ , Python

### Database :

- Sqlite3

## Software used:

- Visual Studio
- Python IDE
- Windows OS
- PuTTy(VM)

## Version control:

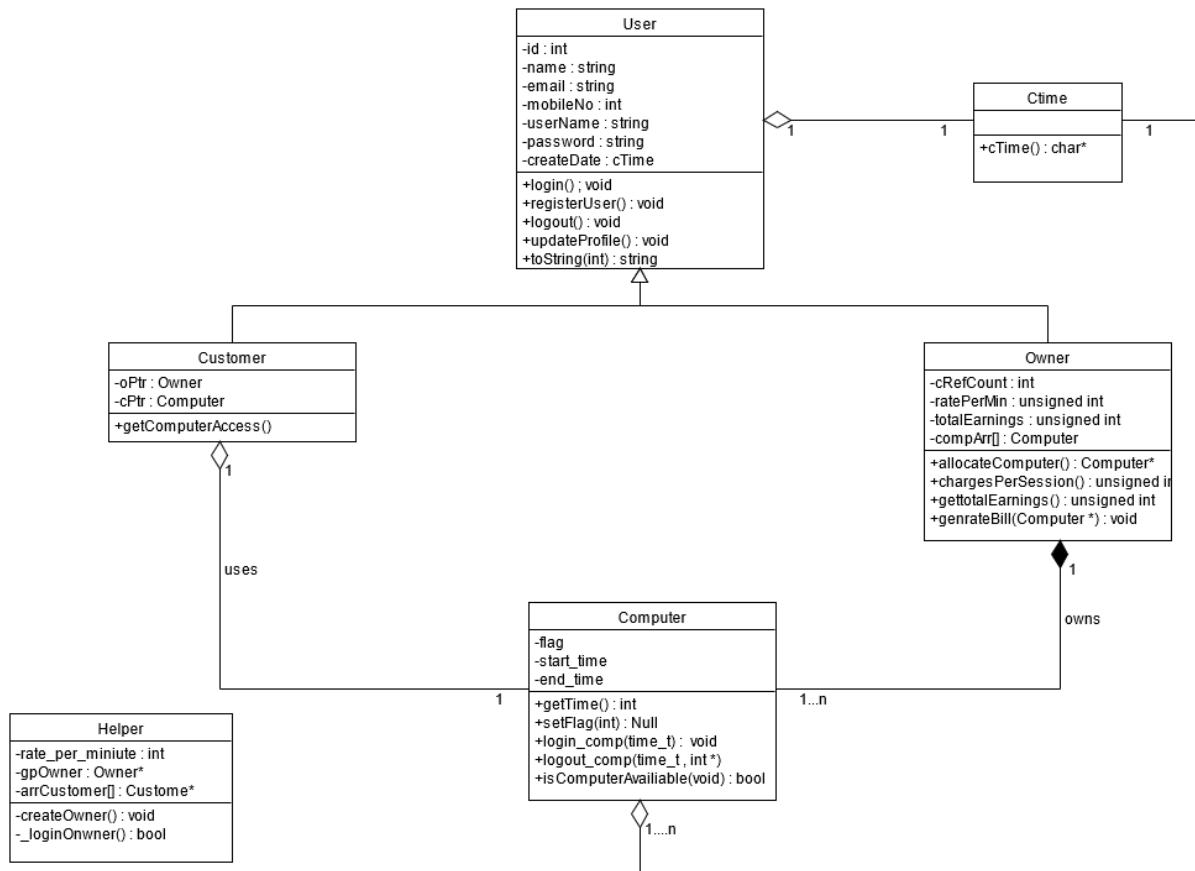
- GitHub

## Team Members:

- 1) Akash Shikare
- 2) Diptajit Ghosh
- 3) Hrushikesh Ahire
- 4) Kadir Sheikh
- 5) Pranjal Enchalwar

## Milestone 2

### Class Diagram



The UML Class diagram is a graphical notation used to model and visualize object oriented systems.

- Classes
- Attributes
- Operations (or methods)
- Relationships among objects.

We have Four classes :

- User
- Customer
- Owner
- Computer

The relationship between the user and Owner ,Customer Class is Inheritance. Its "is a" relationship.

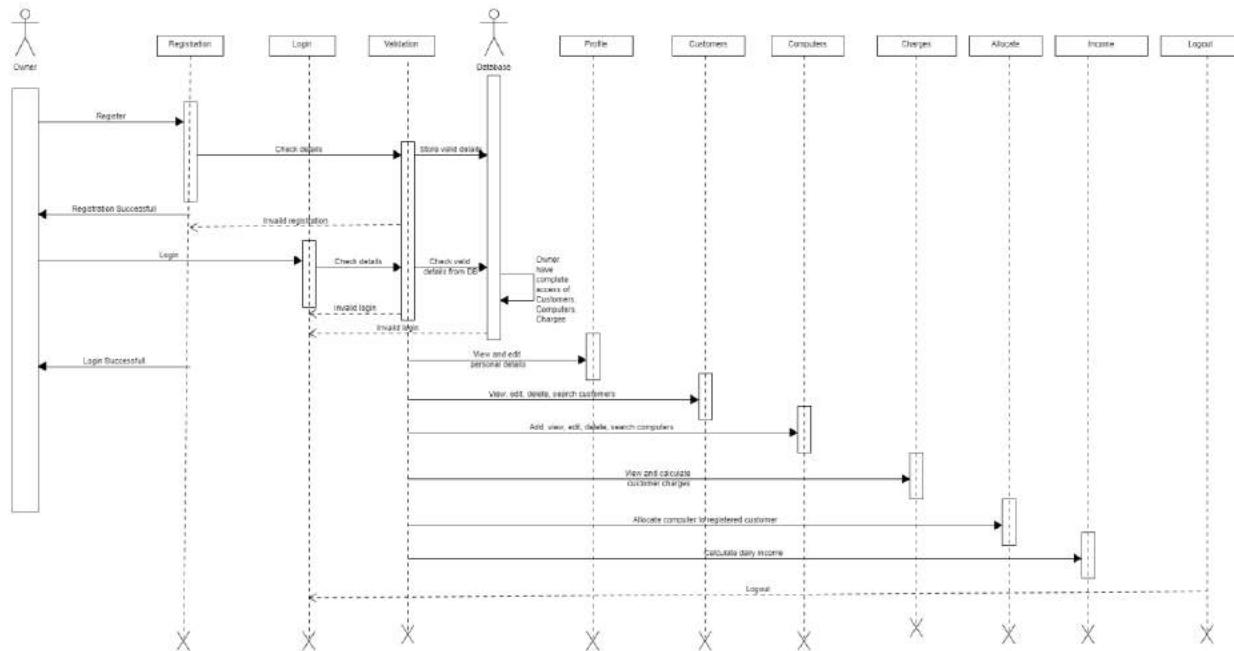
The relationship between Customer and Computer class is Aggregation. It's a "part of" kind of relationship.

The relationship between Owner and the computer is composition (tight Coupled). Its a "has a '' kind of relationship.

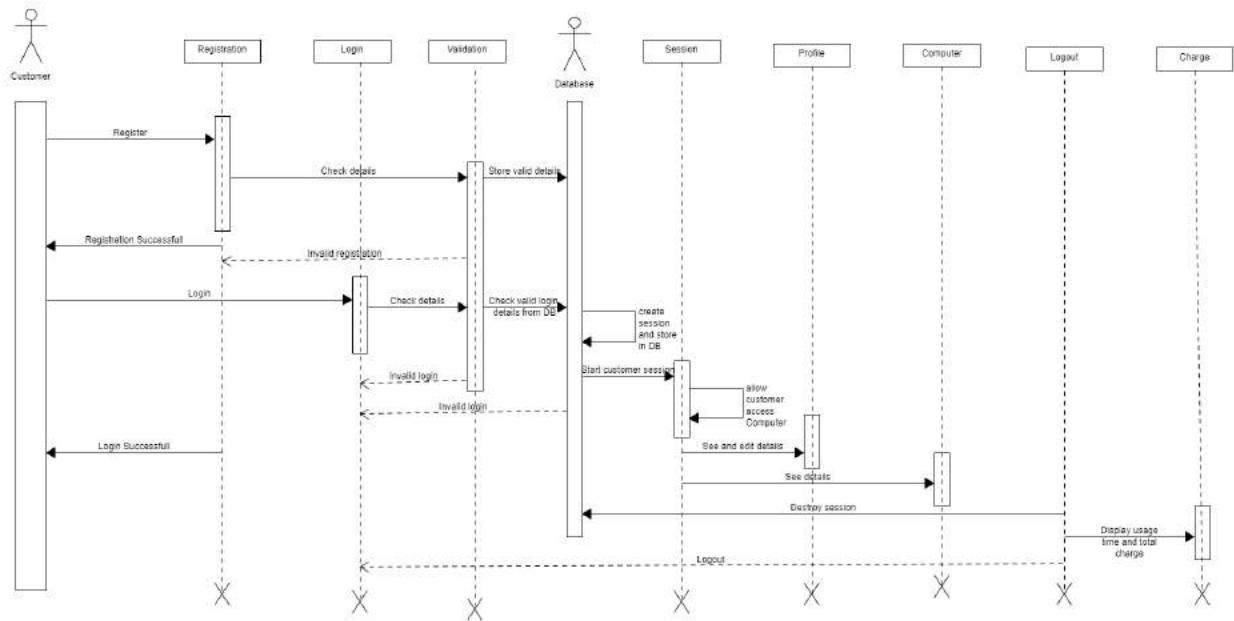
### Sequence Diagrams

Sequence Diagrams illustrate how the different parts of a system interact with each other to carry out a function, and the order in which the interactions occur when a particular use case is executed.

## 1. Owner Sequence Diagram

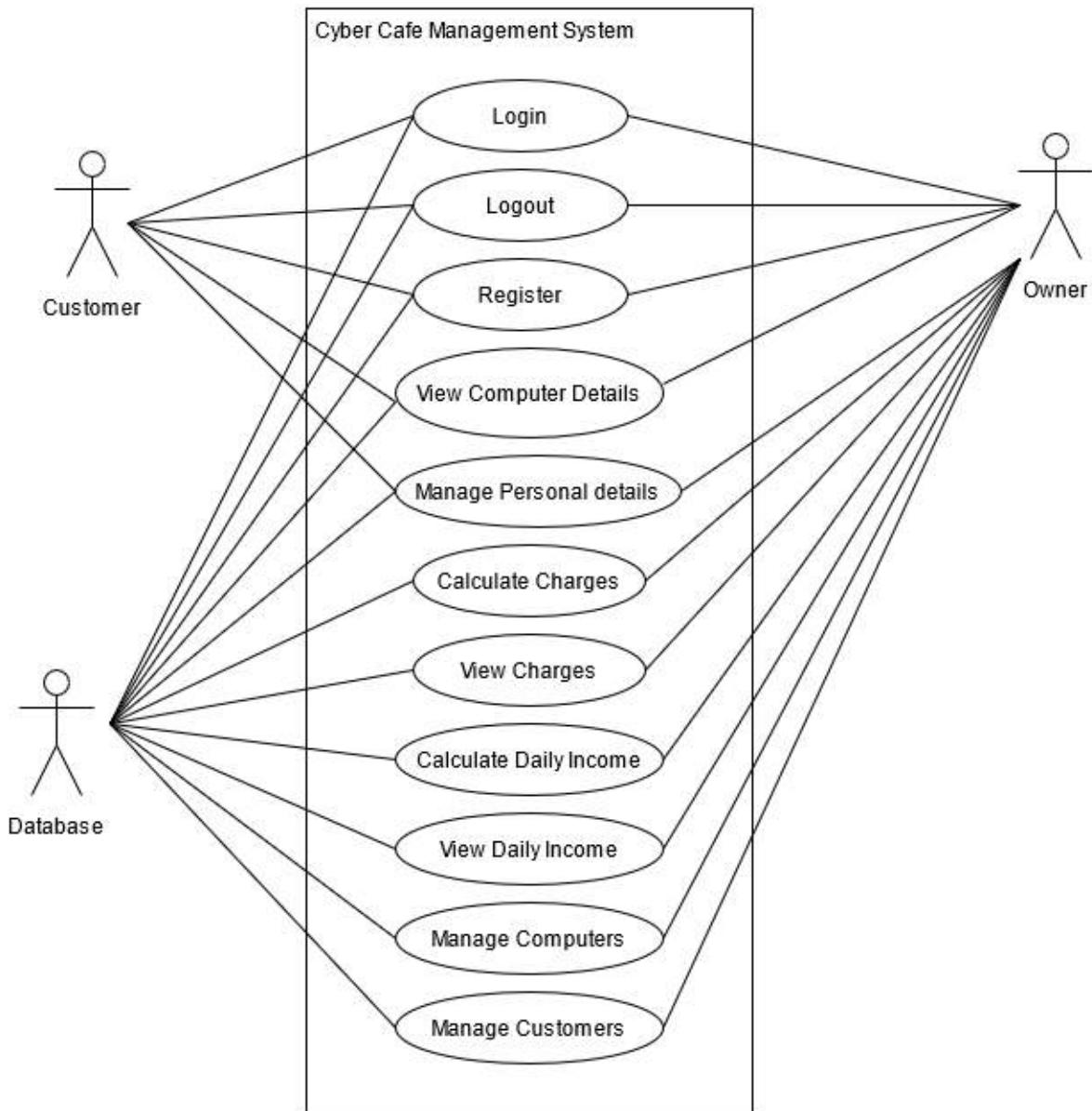


## 2. Customer Sequence Diagram



## Use Case Diagram

Use case diagram specify the expected behavior of the system. It summarizes relationships between use cases, actors, and systems.



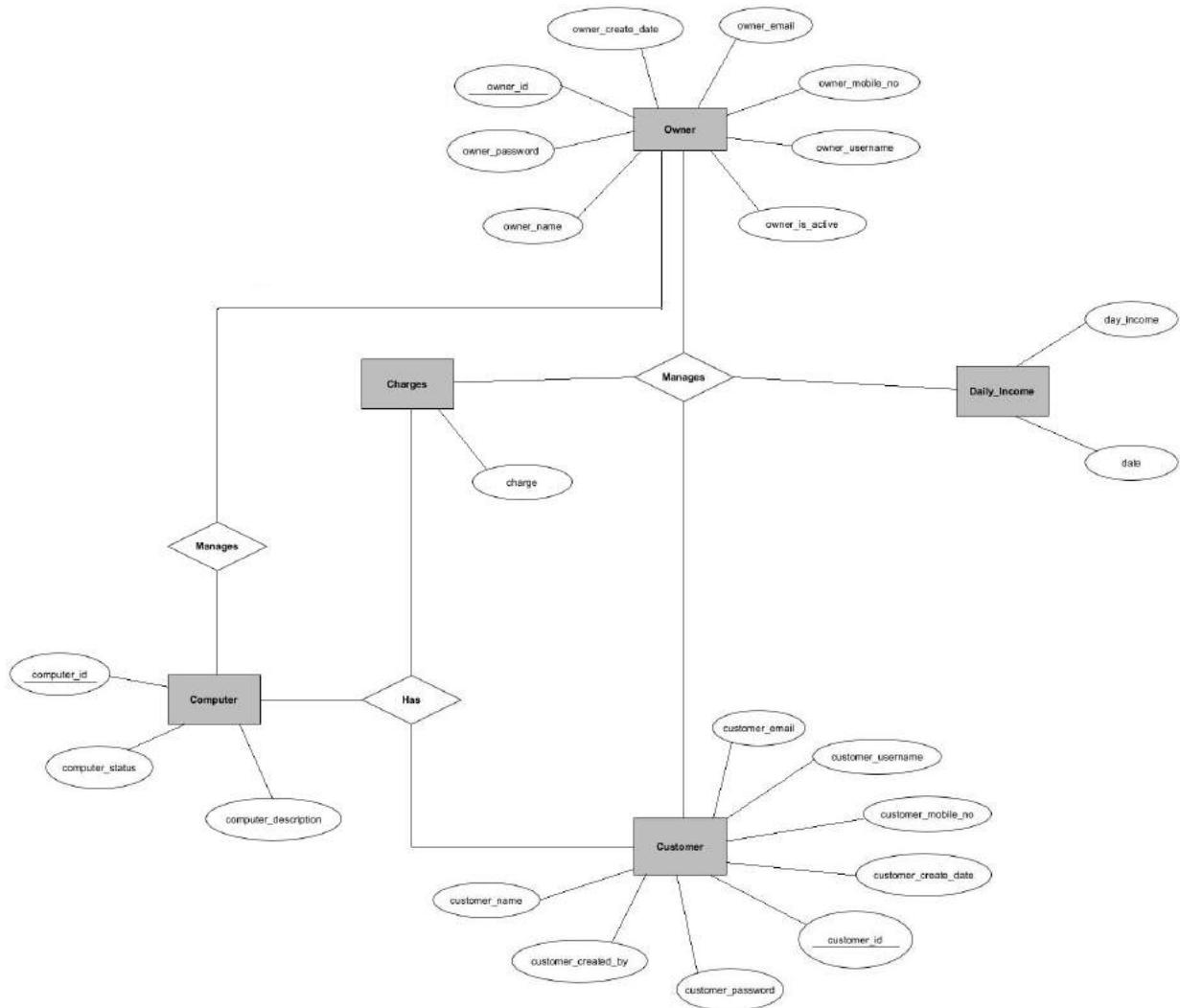
SR NO	Use Case	Description
1	Login	This Use case allows the Customer as well as   the owner to login into the system.
2	Logout	Logs Out the Customer as well as the Owner. Ends the session
3	Register	Registers the user by taking personal information
4	View computer Details	Allows the logged in user to check its computer details
5	Manage Personal details	Allows the user to update any personal <u>details</u> from <u>from</u> the update profile. <u>User</u> can update name, email id and contact number.
6	Calculate charges	Calculates the charges for the particular session for customers.
7	View Charges	Can View the Charges for the customer.
8	Calculate Daily income	Adds up all the charges of the customers and gives a daily income.
9	View daily income	Can view daily income
10	Manage Computers	Operations like adding, allocating and updating computer information is done.
11	Manage customers	Operations like adding, allocating, deleting, searching and updating customers information is done.

## ER Diagram

ER Diagram stands for Entity Relationship Diagram, is a diagram that displays the relationship of entity sets stored in a database.

It helps to explain the logical structure of databases.

ER diagrams are created based on three basic concepts: entities, attributes and relationships.



The details of Owner, Customer and Computers are stored in their respective tables. Each entity (Owner, Customer, Computer, Charge, Daily Income) contains primary keys.

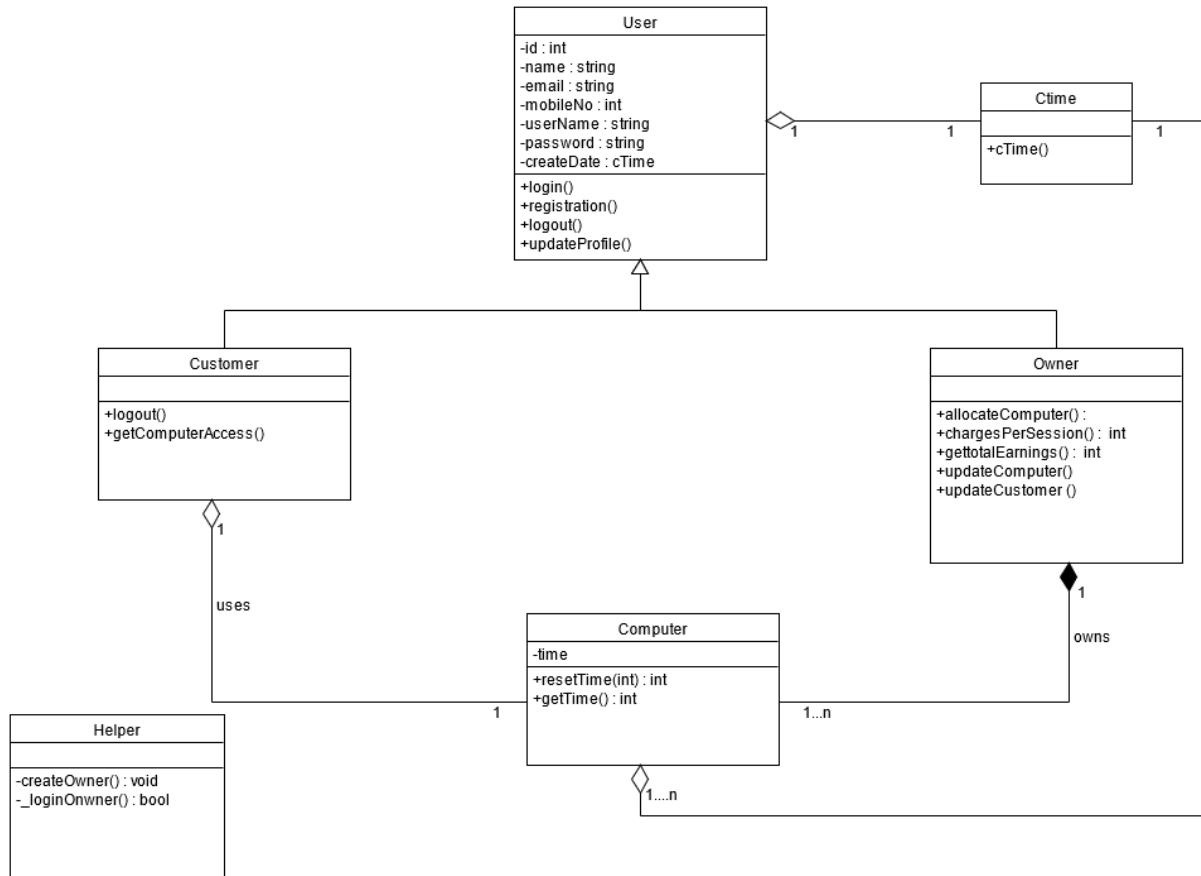
The entity Owner manages all the other entities like Computer, Customer, Charges and Daily Income.

The Customer inherits from the Charges and Computer entities. All the entities Customer, Computer, Owner, Charges, Daily Income are normalized and hence reduce duplicacy of records.

## Domain Model

Domain Modeling is a way to describe and model real world entities and the relationships between them, which collectively describe the problem domain space.

It is a representation of real-world concepts pertinent to the domain that need to be modeled in software.

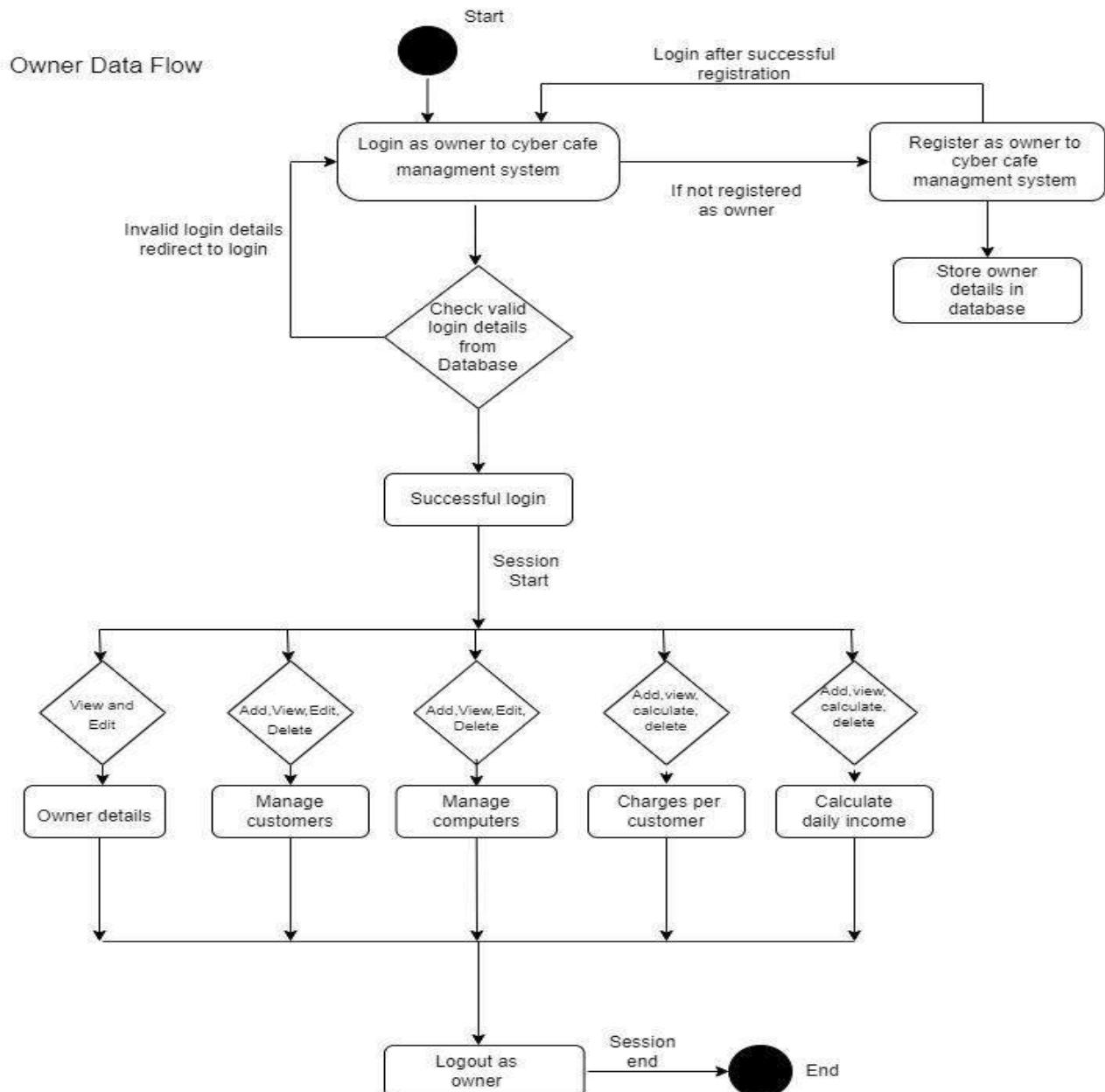


## Data Flow Diagrams

A data flow diagram is a graphical tool for future analysis.

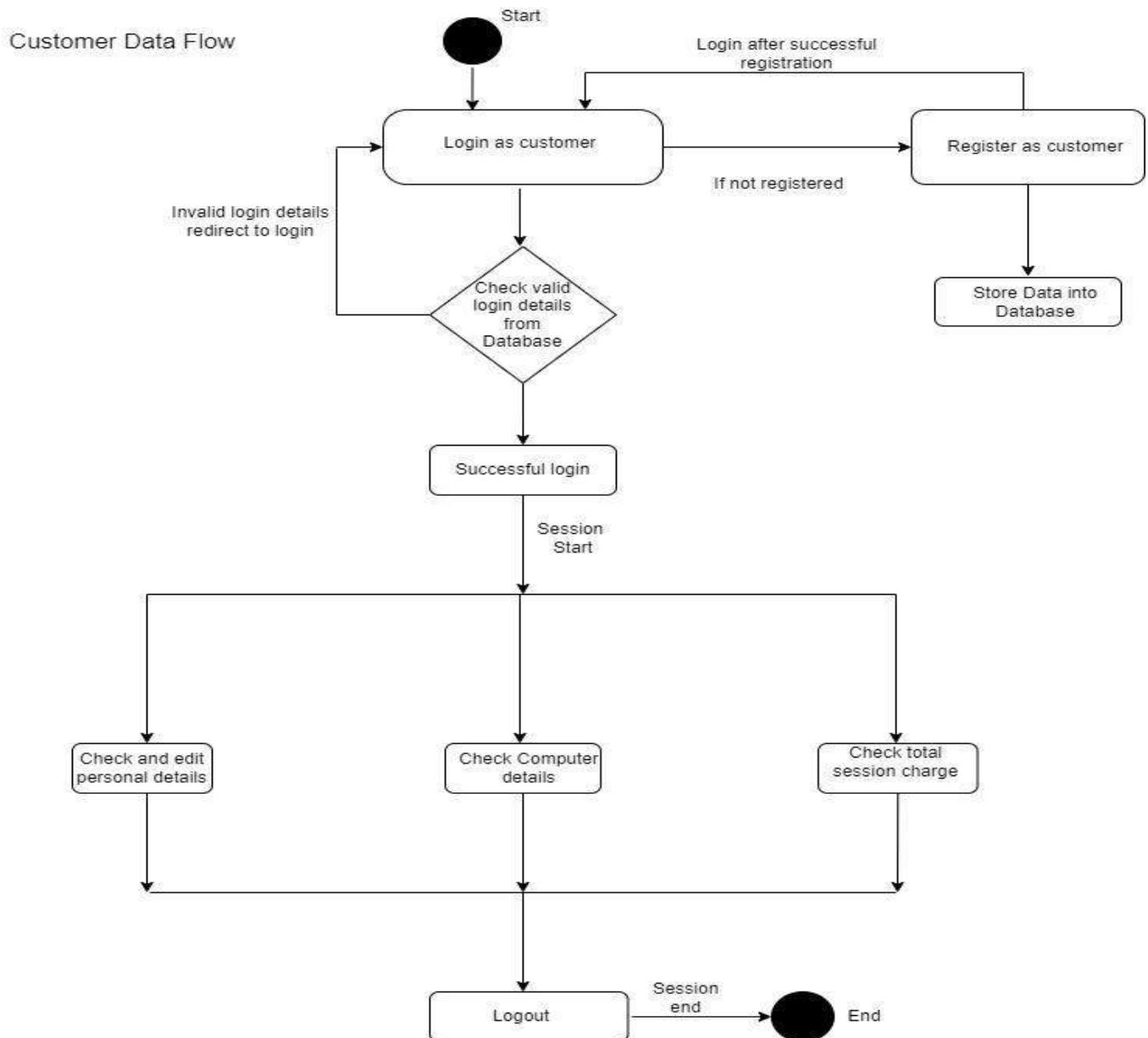
DFD modules a system by using external entities from which data flows to a process, which transforms the data and creates output data flows which go to other process or external entities or files.

### 1. Owner Data Flow Diagram



Owner DFD gives insight into the inputs and outputs of each entity and the process for owner's perspective.

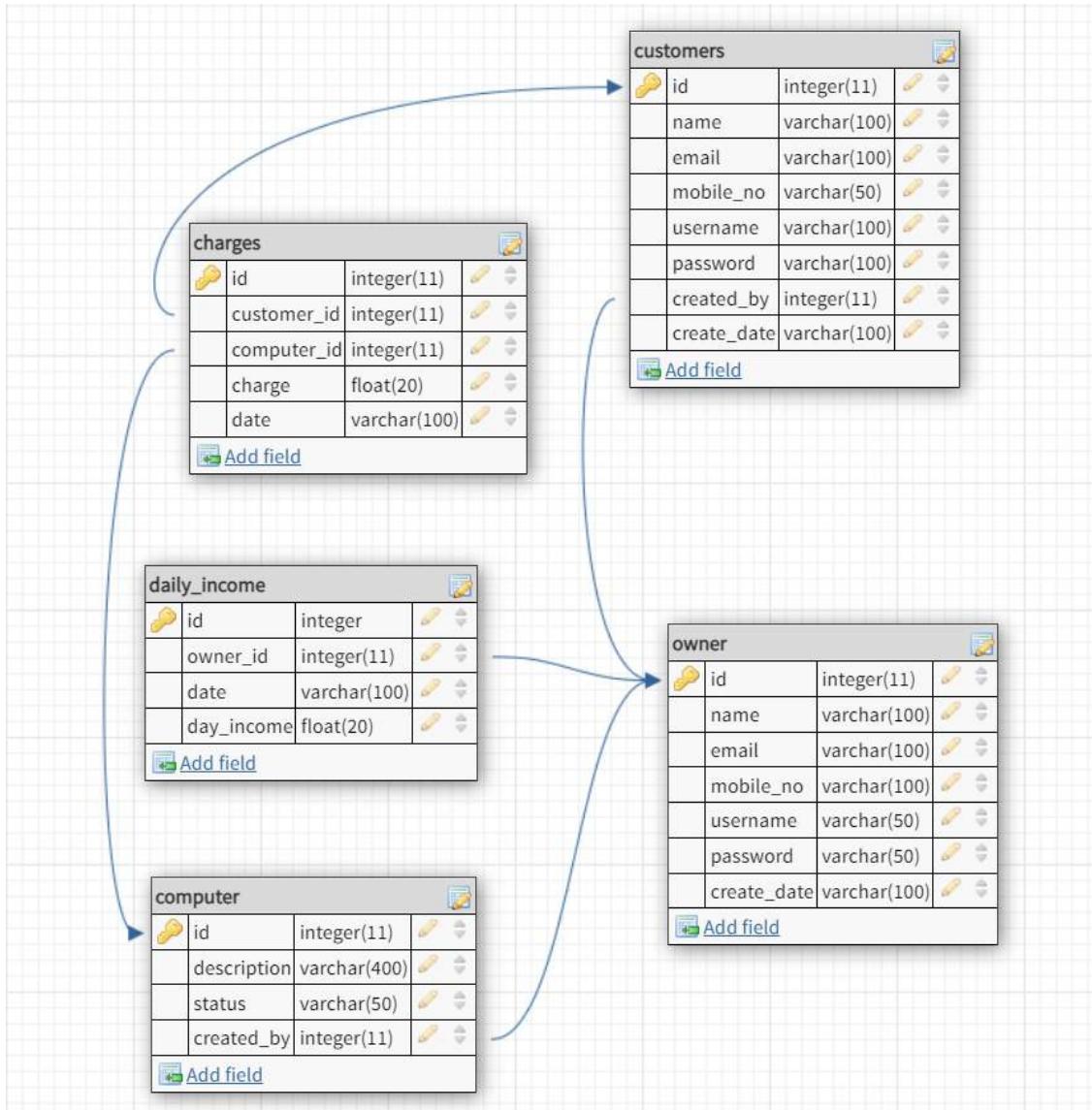
## 2. Customer Data Flow Diagram.



Customer DFD gives insight into the inputs and outputs of each entity and the process for customer's perspective.

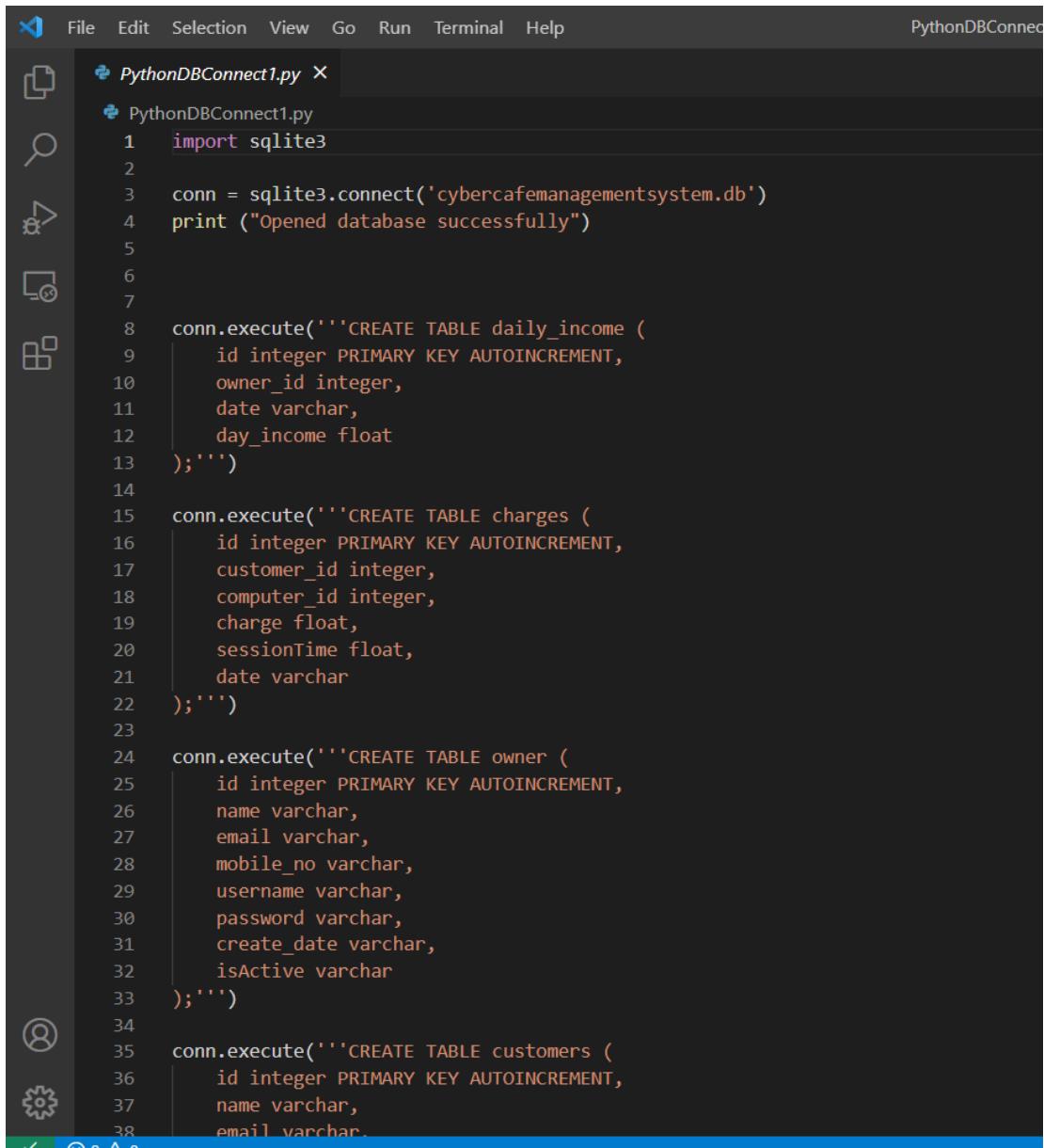
## Database Schema

A database schema represents the logical configuration of all or part of a relational database. It can exist both as a visual representation and as a set of formulas known as integrity constraints that govern a database.



## Milestone 3

### Database



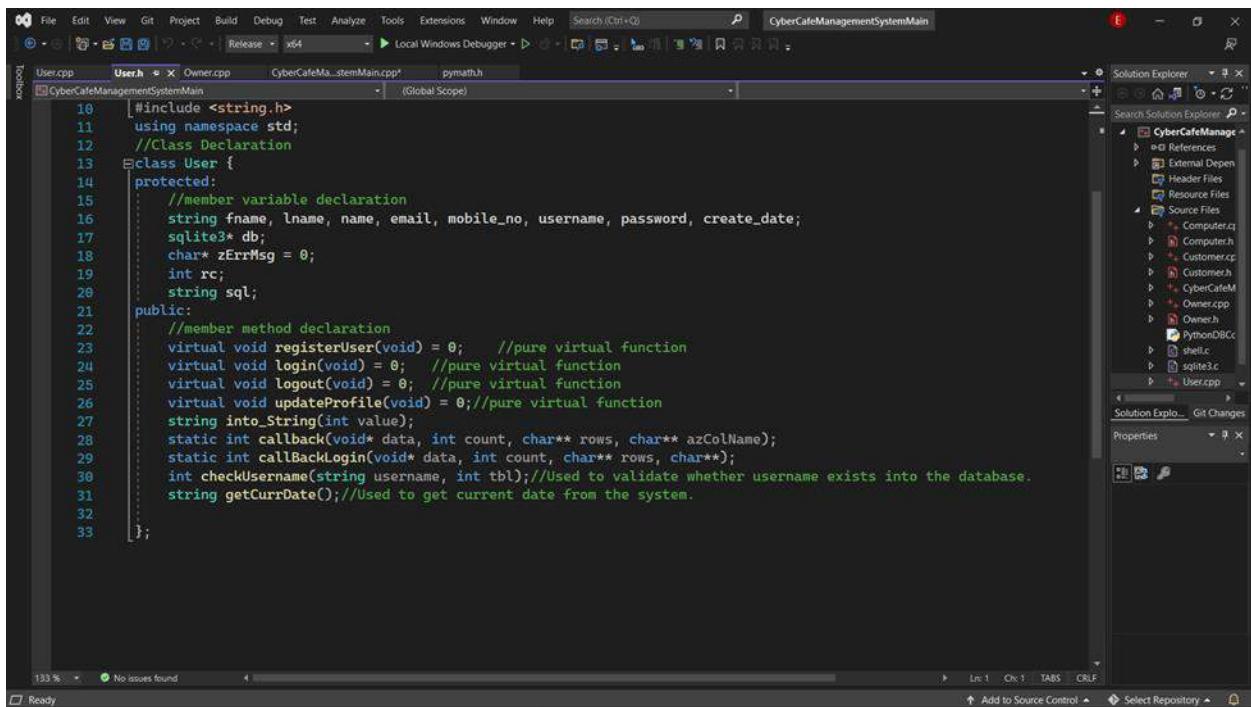
The screenshot shows a code editor interface with a dark theme. The menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. A tab bar at the top shows "PythonDBConnect1.py X" and "PythonDBConnect1.py". The left sidebar contains icons for file operations like Open, Save, Find, and Run. The main code area displays the following Python script:

```
 1 import sqlite3
 2
 3 conn = sqlite3.connect('cybercafemanagementsystem.db')
 4 print ("Opened database successfully")
 5
 6
 7
 8 conn.execute('''CREATE TABLE daily_income (
 9     id integer PRIMARY KEY AUTOINCREMENT,
10     owner_id integer,
11     date varchar,
12     day_income float
13 );''')
14
15 conn.execute('''CREATE TABLE charges (
16     id integer PRIMARY KEY AUTOINCREMENT,
17     customer_id integer,
18     computer_id integer,
19     charge float,
20     sessionTime float,
21     date varchar
22 );''')
23
24 conn.execute('''CREATE TABLE owner (
25     id integer PRIMARY KEY AUTOINCREMENT,
26     name varchar,
27     email varchar,
28     mobile_no varchar,
29     username varchar,
30     password varchar,
31     create_date varchar,
32     isActive varchar
33 );''')
34
35 conn.execute('''CREATE TABLE customers (
36     id integer PRIMARY KEY AUTOINCREMENT,
37     name varchar,
38     email varchar,
```

We integrated Python using sqlite3 module for database connectivity and table creation.

## User Class

### Class Declaration



The screenshot shows the Microsoft Visual Studio IDE interface. The title bar reads "CyberCafeManagementSystemMain". The left pane displays the code for "User.cpp", which contains the definition of a class "User". The code includes protected member variables like string fname, lname, name, email, mobile\_no, username, password, and create\_date, and protected member functions for registerUser, login, logout, and updateProfile. It also includes static methods for callback and checkUsername, and a getCurrentDate method. The right pane shows the "Solution Explorer" with the project structure, including files like Computer.cpp, Computer.h, Customer.cpp, Customer.h, CyberCafeM.cpp, Owner.cpp, PythonDBCC.cpp, shell.c, sqlite3.c, and User.cpp.

```
10  #include <string.h>
11  using namespace std;
12  //Class Declaration
13  class User {
14  protected:
15      //member variable declaration
16      string fname, lname, name, email, mobile_no, username, password, create_date;
17      sqlite3* db;
18      char* zErrMsg = 0;
19      int rc;
20      string sql;
21  public:
22      //member method declaration
23      virtual void registerUser(void) = 0;      //pure virtual function
24      virtual void login(void) = 0;    //pure virtual function
25      virtual void logout(void) = 0;   //pure virtual function
26      virtual void updateProfile(void) = 0;//pure virtual function
27      string into_String(int value);
28      static int callback(void* data, int count, char** rows, char** azColName);
29      static int callBackLogin(void* data, int count, char** rows, char**);
30      int checkUsername(string username, int tbl); //Used to validate whether username exists into the database.
31      string getCurrentDate(); //Used to get current date from the system.
32
33 };
```

User class contains virtual methods for registration, login, logout, update profile for both owner and customer which will get override after inheritance.

## Class definition

The screenshot shows the Microsoft Visual Studio IDE interface. The code editor on the left displays the file `User.cpp` with the following content:

```
1 #include "User.h"
2 #include <iostream>
3 #include "User.h"
4 #include <sstream>
5 using namespace std;
6 //Convert integer to string type function
7 string User::into_String(int value) {
8     ostringstream os;
9     os << value;
10    return os.str();
11 }
12 //callback function implementation
13 int User::callback(void* data, int count, char** rows, char** azColName) {
14
15     int i;
16
17     for (i = 0; i < count; i++) {
18         printf("%s = %s\n", azColName[i], rows[i] ? rows[i] : "NULL");
19     }
20     printf("\n");
21
22     return 0;
23 }
24 int User::callBackLogin(void* data, int count, char** rows, char**)
25 {
26     if (count == 1 && rows) {
27         *static_cast<int*>(data) = atoi(rows[0]);
28         return 0;
29     }
30     return 1;
31 }
```

The Solution Explorer on the right shows the project structure:

- CyberCafeManagement**
  - References
  - External Dependencies
  - Header Files
  - Resource Files
  - Source Files
    - Computer.cpp
    - Computer.h
    - Customer.cpp
    - Customer.h
    - CyberCafeMain.cpp
    - Owner.cpp
    - Owner.h
    - PythonDBC.cpp
    - shell.c
    - sqlite3.c
    - User.cpp
- Solution Explorer - Git Changes
- Properties

Method definition for `into_string` which convert int to string, `callBack` which returns results (records) from the SQLite database.

```
31     }
32     //Used to verify username validity from database
33     int User::checkUsername(string username, int tbl) {
34
35         rc = sqlite3_open("cybercafemanagementsystem.db", &db);
36         if (rc) {
37             fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
38         }
39         else {
40             //fprintf(stderr, "Opened database successfully\n");
41
42             if (tbl == 1) {
43                 sql = "SELECT COUNT(*) from owner where username like '" + username + "'";
44             }
45             else if ((tbl == 2)) {
46                 sql = "SELECT COUNT(*) from customers where username like '" + username + "'";
47             }
48
49
50
51             char* sqliteInsert = const_cast<char*>(sql.c_str());
52
53             int count = 0;
54             rc = sqlite3_exec(db, sqliteInsert, callBackLogin, &count, &zErrMsg);
55
56             if (count == 1) {
57                 return -1;
58             }
59             else {
60                 return 1;
61             }
62         }
63     }
64
65     sqlite3_close(db);
66
67 }
68
69     //Get current date from database
70     string User::getCurrDate() {
71         string date;
72         time_t now = time(0);
73         tm* ltm = localtime(&now);
74
75         int year = 1900 + ltm->tm_year;
76         int month = 1 + ltm->tm_mon;
77         int day = ltm->tm_mday;
78
79         date = into_String(day) + "-" + into_String(month) + "-" + into_String(year);
80
81     }
82 }
```

checkUsername checks if username exists in that table or not for owner and customer registration.

```
61     return 1;
62 }
63
64
65     sqlite3_close(db);
66
67 }
68
69     //Get current date from database
70     string User::getCurDate() {
71         string date;
72         time_t now = time(0);
73         tm* ltm = localtime(&now);
74
75         int year = 1900 + ltm->tm_year;
76         int month = 1 + ltm->tm_mon;
77         int day = ltm->tm_mday;
78
79         date = into_String(day) + "-" + into_String(month) + "-" + into_String(year);
80
81     }
82 }
```

getCurDate returns today's date in string format.

Owner class

# Class Declaration

This screenshot shows the Microsoft Visual Studio IDE interface. The main window displays the code for the `Owner.h` header file. The code defines a class `Owner` that inherits from `User`. It includes private member variables for a reference count and earnings, and public methods for user operations like registration and login. The code is part of a larger project named `CyberCafeManagementSystemMain`, which also includes files for `Computer`, `Customer`, and `PyMath`.

```
1 #pragma once
2 #include "User.h"
3 #include "Computer.h"
4 #define NUM_OF_TOTAL_COMPUTERS 10
5 //Class Declaration
6 class Owner : public User {
7 private:
8     //Member variable declaration
9     int cRefCount = 0;
10    unsigned int rate_per_min;
11    unsigned int total_earning = 0;
12    int num_of_min;
13    Computer compArr[NUM_OF_TOTAL_COMPUTERS];
14
15
16 public:
17     //Method Declaration
18     Owner(unsigned int);
19     ~Owner(void);
20
21     //User specific methods
22     void registerUser(void);
23     void login(void);
24     void logout(void);
25     //CRUD
26     void searchCustomer();
27     void deleteCustomer();
28     void showCustomerList();
29     void deleteOwner(void);
30     void showOwnerList(void);
31     void updateProfile(void);
32
33
34     //CRUD operations on computer
35     void addComputer();
36     void searchComputer();
37     void listAllComputers();
38     void deleteComputer();
39     void updateComputer();
40     //helping functions
41     int checkId(string);
42     //Owner specific methods
43     int generateBill(Computer* );
44     unsigned int chargesPerSession(Computer* );
45     void getTotalEarning(void);
46     void checkChargeSessionTimeForCustomer();
47     void allocateComputer(void);
48     Computer* returnAllocatedComputerAddress(int);
49     int getCustomerCount(void);
50     int getTime(void);
51     void getAllCharges(void);
52
53
54 };
```

This screenshot continues the code from the previous Visual Studio window, showing the completion of the `Owner` class definition. The class now includes methods for generating bills, calculating total earnings, checking session times, allocating computers, and getting customer counts. The code is still within the `CyberCafeManagementSystemMain` project.

```
28     void showCustomerList();
29     void deleteOwner(void);
30     void showOwnerList(void);
31     void updateProfile(void);
32
33
34     //CRUD operations on computer
35     void addComputer();
36     void searchComputer();
37     void listAllComputers();
38     void deleteComputer();
39     void updateComputer();
40     //helping functions
41     int checkId(string);
42     //Owner specific methods
43     int generateBill(Computer* );
44     unsigned int chargesPerSession(Computer* );
45     void getTotalEarning(void);
46     void checkChargeSessionTimeForCustomer();
47     void allocateComputer(void);
48     Computer* returnAllocatedComputerAddress(int);
49     int getCustomerCount(void);
50     int getTime(void);
51     void getAllCharges(void);
52
53
54 };
```

Owner class contains declaration of Owner authentication functions, Customer CRUD functions and owner profile update.

The screenshot shows the Microsoft Visual Studio IDE interface. The title bar reads "CyberCafeManagementSystemMain". The code editor window displays the "Owner.h" file, which contains the following declarations:

```
31 //Owner specific methods
32 int generateBill(Computer*);
33 unsigned int chargesPerSession(Computer*);
34 void getTotalEarning(void);
35 void checkChargeSessionTimeForCustomer();
36 void allocateComputer(void);
37 Computer* returnAllocatedComputerAddress(int);
38 int getCustomerCount(void);
39 int getTime(void);
40 void getAllCharges(void);
41 };
42
43
44
```

The Solution Explorer window on the right shows the project structure:

- CyberCafeManagementSystemMain
- ↳ References
- ↳ External Dependencies
- ↳ Header Files
- ↳ Resource Files
- ↳ Source Files
  - ↳ Computer.cpp
  - ↳ Computer.h
  - ↳ Customer.cpp
  - ↳ Customer.h
  - ↳ CyberCafeM
  - ↳ Owner.cpp
  - ↳ Owner.h
  - ↳ PythonDBC
  - ↳ shell.c
  - ↳ sqlite3.c
  - ↳ User.cpp

Declaration to allocate Computer, get today's total income and get the list of charges.

## Class Definition

The screenshot shows the Microsoft Visual Studio IDE interface. The title bar reads "CyberCafeManagementSystemMain". The code editor window displays the file "Owner.cpp" with the following content:

```
1 #include <iostream>
2 #include <windows.h>
3 #include "owner.h"
4 #include "Customer.h"
5 #include "Computer.h"
6
7 using namespace std;
8 //Constructor
9 Owner::Owner(unsigned int _rate_per_min = 1) : rate_per_min(_rate_per_min) {
10     std::cout << "Owner Object Created" << std::endl;
11 }
12 //Destructor
13 Owner::~Owner(void) {
14     std::cout << "Owner Object Destroyed" << std::endl;
15 }
16
17 // User specific method definition
18 //Register new owner into the database
19 void Owner::registerUser(void) {
20     rc = sqlite3_open("cybercafemanagementsystem.db", &db);
21
22     if (rc) {
23         fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
24     } else {
25         fprintf(stderr, "Opened database successfully\n");
26
27
28
29
30     create_date = getCurrDate();
31     cout << "OWNER REGISTRATION" << endl;
32 }
```

The Solution Explorer on the right shows the project structure:

- CyberCafeManagementSystemMain
- Source Files
  - Computer.cpp
  - Computer.h
  - Customer.cpp
  - Customer.h
  - CyberCafeM
  - Owner.cpp
  - Owner.h
  - PythonDBCc
  - shell.c
  - sqlite3.c
  - User.cpp

Definition of owner constructor and destructor.

The screenshot shows the Microsoft Visual Studio IDE interface. The main window displays the `Owner.cpp` file under the tab `CyberCafeManagementSystemMain`. The code is written in C++ and handles the registration of a new owner. It prompts the user to enter their first name, last name, email, mobile number, username, and password. It then checks if the username already exists in the database. If it does, it exits with an error message. Otherwise, it constructs an SQL INSERT statement and executes it using SQLite3. The code includes imports for `pymath.h` and `PythonDBC.h`.

```
cout << "OWNER REGISTRATION" << endl;
cout << "Enter first name" << endl;
cin >> fname;

cout << "Enter last name" << endl;
cin >> lname;

name = fname + " " + lname;

cout << "Enter email" << endl;
cin >> email;

cout << "Enter mobile" << endl;
cin >> mobile_no;

cout << "Enter username" << endl;
cin >> username;

int checkU = checkUsername(username, 1);
if (checkU == -1) {
    cout << "Username already exists in database. Try again." << endl;
    exit(0);
}

cout << "Enter password" << endl;
cin >> password;
```

This screenshot continues the same Microsoft Visual Studio session, showing the completion of the registration process. The `Owner.cpp` file is still open, and the code now handles the execution of the SQL insert statement. It checks if the operation was successful. If it was, it prints a success message; otherwise, it prints an error message from the SQLite library. The code also includes imports for `pymath.h` and `PythonDBC.h`.

```
cin >> password;

if (name.size() == 0 || email.size() == 0 || mobile_no.size() == 0 || username.size() == 0 || password.size() == 0) {
    cout << "Field cannot be empty" << endl;
}
else {
    sql = "INSERT INTO owner(name, email, mobile_no, username, password, create_date, isActive) VALUES('" + na
    //sql = "SELECT * FROM owner WHERE";
    char* sqliteInsert = const_cast<char*>(sql.c_str());

    rc = sqlite3_exec(db, sqliteInsert, callback, 0, &zErrMsg);

    if (rc != SQLITE_OK) {
        fprintf(stderr, "SQL error: %s\n", zErrMsg);
        sqlite3_free(zErrMsg);
    }
    else {
        fprintf(stdout, "\nRegistration successfully\n");
    }
}
```

The screenshot shows the Microsoft Visual Studio interface. The top menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and Search (Ctrl+Q). The title bar says "Cyber\_cafe\_new". The left sidebar has "Toolbox" and "Task List". The main area shows the "Microsoft Visual Studio Debug Console" with the following text:

```
Owner Object created
Opened database successfully
OWNER REGISTRATION
Enter first name
Kadir
Enter last name
Sheikh
Enter email
sheikhkadri2@gmail.com
Enter mobile
7972112884
Enter username
kadir123
Enter password
1234

Registration successfully

D:\COPSY\Cyber_cafe_new\Cyber_cafe_new\Debug\Cyber_cafe_new.exe (process 33500) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

The Solution Explorer on the right shows the project structure:

- Cyber\_cafe\_new (D:\COPSY\Cyber\_cafe\_new)
- Debug
- sqlite
- x64
- Cyber\_cafe\_new.cpp
- Cyber\_cafe\_new.vcxproj
- Cyber\_cafe\_new.vcxproj.filters
- Cyber\_cafe\_new.vcxproj.user
- ↳ ddl.cpp
- ↳ owner.cpp
- ↳ owner.obj
- PY python.py

The Properties and Git Changes tabs are also visible in the Solution Explorer.

Definition of owner registration function which is overriding from User class virtual registration function.

The screenshot shows the Microsoft Visual Studio interface with the code editor open to the "Owner.cpp" file. The code is as follows:

```
88
89
90 }
91
92     sqlite3_close(db);
93
94 }
95 //Owner logging in to the system
96 void Owner::login(void) {
97     rc = sqlite3_open("cybercafemanagementsystem.db", &db);
98
99     if (rc) {
100         fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
101     }
102     else {
103         fprintf(stderr, "Opened database successfully\n");
104
105         cout << "OWNER LOGIN" << endl;
106         cout << "Enter Username" << endl;
107         cin >> username;
108
109         cout << "Enter Password" << endl;
110         cin >> password;
111
112
113         sql = "SELECT COUNT(*) as count FROM owner WHERE username = '" + username + "' AND password = '" + password
114         //sql = "SELECT * FROM owner";
115
116
117         char* sqliteInsert = const_cast<char*>(sql.c_str());
118 }
```

The Solution Explorer on the right shows the project structure for "CyberCafeManagementSystemMain":

- CyberCafeManagementSystemMain
- Owner.h
- User.cpp
- User.h
- Owner.cpp
- CyberCafeManagementSystemMain.cpp
- pymath.h

The Properties and Git Changes tabs are also visible in the Solution Explorer.

```
118     int count = 0;
119     rc = sqlite3_exec(db, sqliteInsert, callBackLogin, &count, &zErrMsg);
120
121     if (rc != SQLITE_OK) {
122         fprintf(stderr, "SQL error: %s\n", zErrMsg);
123         sqlite3_free(zErrMsg);
124     }
125     else {
126         fprintf(stdout, "\nQuery Executed successfully\n");
127         if (count == 1) {
128
129             sql = "UPDATE owner SET isActive='Y' WHERE username='" + username + "'";
130             char* sqliteInsert = const_cast<char*>(sql.c_str());
131             rc = sqlite3_exec(db, sqliteInsert, callback, 0, &zErrMsg);
132
133             if (rc != SQLITE_OK) {
134                 fprintf(stderr, "SQL error: %s\n", zErrMsg);
135                 sqlite3_free(zErrMsg);
136             }
137             else {
138                 fprintf(stdout, "\nLogin Successful\n");
139             }
140
141         }
142         else {
143             cout << "Login Failed" << endl;
144         }
145     }
146 }
147
148 }
```

```
Owner Object Created
Opened database successfully
Owner Login
Enter Username
kadir123
Enter Password
1234

Login Successful
Query Executed successfully

D:\CCMS\Cyber_cafe_new\Cyber_cafe_new\Debug\Cyber_cafe_new.exe (process 25172) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Definition of owner login function which is overriding from User class virtual login function.

The screenshot shows the Microsoft Visual Studio IDE interface. The title bar reads "CyberCafeManagementSystemMain". The left pane displays the code editor with the "Owner.cpp" file open. The code implements an "Owner" class with a "logout" method. It uses SQLite3 to update the database, changing the "isActive" field of the "owner" table to 'N' and the "status" field of the "computer" table to 'N'. The right pane shows the "Solution Explorer" with the project structure, including files like "Owner.h", "User.cpp", "User.h", "CyberCafeManagementSystemMain.cpp", and "pymath.h".

```
151     }
152     sqlite3_close(db);
153 }
154 }
155 //Owner logging out from system
156 void Owner::logout(void) {
157     rc = sqlite3_open("cybercafemanagementsystem.db", &db);
158
159     if (rc) {
160         fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
161     }
162     else {
163         fprintf(stderr, "Opened database successfully\n");
164         cout << "Please confirm your username to logout as owner" << endl;
165         cin >> username;
166
167         sql = "UPDATE owner SET isActive='N' WHERE username='" + username + "'";
168         char* sqliteInsert = const_cast<char*>(sql.c_str());
169         rc = sqlite3_exec(db, sqliteInsert, callback, 0, &zErrMsg);
170
171         if (rc != SQLITE_OK) {
172             fprintf(stderr, "SQL error: %s\n", zErrMsg);
173             sqlite3_free(zErrMsg);
174         }
175         else {
176
177             sql = "UPDATE computer SET status ='N'";
178             char* sqliteInsert = const_cast<char*>(sql.c_str());
179             rc = sqlite3_exec(db, sqliteInsert, callback, 0, &zErrMsg);
180
181         }
182     }
183     sqlite3_close(db);
184 }
185
186
187
188     fprintf(stdout, "\nLogout Successful\n");
189 }
190
191 }
192
193 }
194 }
195
196 sqlite3_close(db);
197
198 }
199
200 //Generate bill for the usage functions
201 int Owner::generateBill(Computer* cPtr) {
202     //variable declaration
203     unsigned int amount_to_pay;
204     char str[255];
205
206     //code
207     amount_to_pay = chargesPerSession(cPtr);
208
209
210     return amount_to_pay;
211 }
```

This screenshot continues the "Owner.cpp" file from the previous one. It shows the completion of the "generateBill" function. The function takes a pointer to a "Computer" object and calculates the amount to pay based on the charges per session. The code then returns this value. The rest of the file contains standard C++ code, including comments and blank lines.

```
200 //Generate bill for the usage functions
201 int Owner::generateBill(Computer* cPtr) {
202     //variable declaration
203     unsigned int amount_to_pay;
204     char str[255];
205
206     //code
207     amount_to_pay = chargesPerSession(cPtr);
208
209
210     return amount_to_pay;
211 }
```

```
Microsoft Visual Studio Debug Console
Owner Object Created
Opened database successfully
Please confirm your username to logout as owner
kadir

Logout Successful

D:\CCMS\Cyber_cafe_new\Cyber_cafe_new\Debug\Cyber_cafe_new.exe (process 33116) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Definition of owner logout function and generate bill which returns total amount of money for the session spent by the customer on the allocated computer.

The screenshot shows the Microsoft Visual Studio interface. The title bar reads "CyberCafeManagementSystemMain". The code editor displays the "Owner.cpp" file with the following content:

```
199 //Generate bill for the usage functions
200 int Owner::generateBill(Computer* cPtr) {
201     //variable declaration
202     unsigned int amount_to_pay;
203     char str[255];
204
205     //code
206     amount_to_pay = chargesPerSession(cPtr);
207
208     return amount_to_pay;
209 }
210
211 //Calculate charge per session method
212
213 unsigned int Owner::chargesPerSession(Computer* _cPtr) {
214     //code
215     time_t end_time = time(NULL);
216
217     _cPtr->logout_comp(end_time, &num_of_min); //actually it is number of sec
218
219     total_earning = total_earning + (num_of_min * rate_per_min);
220     cRefCount--;
221     return (num_of_min * rate_per_min);
222 }
223
224 //Method to find total earning of owner in day
225 void Owner::getTotalEarning(void) {
```

The Solution Explorer on the right shows the project structure with files like Computer.cpp, Computer.h, Customer.cpp, Customer.h, CyberCafeM.cpp, Owner.cpp, Owner.h, PythonDBCc.cpp, shell.c, and sqlite3.c.

Definition of charge per session which calculate total amount of money.

```
226 [{}  
227  
228     //Method to find total earning of owner in day  
229     void Owner::getTotalEarning(void) {  
230         //code  
231         rc = sqlite3_open("cybercafemanagementsystem.db", &db);  
232  
233         if (rc) {  
234             fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));  
235         }  
236         else {  
237             cout << getCurrDate() << "(Today's) total income so far is as follows." << endl;  
238  
239             sql = "SELECT SUM(charge) as TOTAL FROM charges WHERE date= '" + getCurrDate() + "'";  
240  
241             char* sqliteInsert = const_cast<char*>(sql.c_str());  
242             rc = sqlite3_exec(db, sqliteInsert, callback, 0, &zErrMsg);  
243  
244             if (rc != SQLITE_OK) {  
245                 fprintf(stderr, "SQL error: %s\n", zErrMsg);  
246                 sqlite3_free(zErrMsg);  
247             }  
248             else {  
249                 cout << "Enter your owner username to save daily income in database." << endl;  
250                 cin >> username;  
251  
252                 sql = "SELECT COUNT(*) as count FROM daily_income WHERE date = '" + getCurrDate() + "'";  
253  
254  
255  
256
```

```
256         sql = "SELECT COUNT(*) as count FROM daily_income WHERE date = '" + getCurrDate() + "'";  
257  
258         char* sqliteInsert = const_cast<char*>(sql.c_str());  
259  
260         int count = 0;  
261         rc = sqlite3_exec(db, sqliteInsert, callBackLogin, &count, &zErrMsg);  
262  
263  
264  
265         if (count == 1) {  
266             //sql = "INSERT INTO daily_income (owner_id , date , day_income) VALUES((SELECT id from owner WHERE  
267             //username ='" + username + "'))";  
268             sql = "UPDATE daily_income SET owner_id=(SELECT id from owner WHERE username = '" + username + "' A  
269             //nd day_income ='" + dayIncome + "');  
270  
271             char* sqliteInsert = const_cast<char*>(sql.c_str());  
272             rc = sqlite3_exec(db, sqliteInsert, callback, 0, &zErrMsg);  
273  
274         }  
275         else {  
276  
277             sql = "INSERT INTO daily_income (owner_id , date , day_income) VALUES((SELECT id from owner WHERE u  
278             //sername ='" + username + "'))";  
279             char* sqliteInsert = const_cast<char*>(sql.c_str());  
280             rc = sqlite3_exec(db, sqliteInsert, callback, 0, &zErrMsg);  
281  
282         }  
283  
284  
285         if (rc != SQLITE_OK) {  
286             fprintf(stderr, "SQL error: %s\n", zErrMsg);  
287         }
```

The screenshot shows the Microsoft Visual Studio IDE interface. The top window displays the code for `CyberCafeManagementSystemMain`, specifically the `Owner.cpp` file. The code includes logic for handling database operations and user input for saving daily income.

```

286     fprintf(stderr, "SQL_error: %s\n", zErrMsg);
287     sqlite3_free(zErrMsg);
288 }
289 else {
290     fprintf(stderr, "Daily income added to database successfully\n");
291 }
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299 }
300 }
301
302 void Owner::checkChargeSessionTimeForCustomer() {
303     rc = sqlite3_open("cybercafemanagementsystem.db", &db);
304
305     if (rc) {
306         fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
307     }
308     else {
309
310         cout << "Enter username" << endl;
311         cin >> username;
312
313         int checkU = checkUsername(username, 2);
314         if (checkU == -1) {
315
316

```

The bottom window shows the output of the application. It displays a message about an owner object being created and provides the total daily income.

```

Owner Object Created
29-12-2021(Today's) total income so far is as follows.
TOTAL = 10.0

Enter your owner username to save daily income in database.

```

The Diagnostic Tools window on the right shows performance metrics over a 12-second session, including Process Memory usage and CPU usage.

**Definition of get today's total earning (Daily income).**

The screenshot shows the Microsoft Visual Studio IDE interface. The top menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and Search (Ctrl+Q). The toolbar below has icons for New, Open, Save, Build, Run, and Stop. The status bar at the bottom indicates 'Ready'.

The code editor displays the `Owner.cpp` file from the `CyberCafeManagementSystemMain` project. The code handles various database operations using SQLite3. It includes functions like `getCustomerCount` and `searchCustomer`. The code uses standard C++ syntax with comments explaining the logic.

```
331     sqlite3_free(zErrMsg);
332 }
333 }
334 }
335 }
336 else {
337     cout << "No Computer has been allocated to you by owner today." << endl;
338     exit(0);
339 }
340 }
341 else {
342     cout << "Customer for this username is not registered yet." << endl;
343     exit(0);
344 }
345 }
346 }
347 }
348 }

// Use to get number of customer into the database
349 Eint Owner::getCustomerCount(void) {
350     return cRefCount;
351 }
352 // Search list of available customers from database
353 void Owner::searchCustomer() {
354     string username;
355     cout << "Enter the username of customer to be searched";
356     cin >> username;
357     rc = sqlite3_open("cybercafemanagementsystem.db", &db);

358     if (rc) {
359         fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
360     }
361 }
```

This screenshot continues the `Owner.cpp` file from the previous one. It shows the implementation of the `checkChargeSessionTimeForCustomer` function. The code connects to the database, performs a query to find the total charge and session time for a given customer, and prints the results.

```
301 Evoid Owner::checkChargeSessionTimeForCustomer() {
302     rc = sqlite3_open("cybercafemanagementsystem.db", &db);

303     if (rc) {
304         fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
305     }
306     else {

307         cout << "Enter username" << endl;
308         cin >> username;

309         int checkU = checkUsername(username, 2);
310         if (checkU == -1) {

311             sql = "SELECT COUNT(*) as count FROM charges WHERE customer_id = (SELECT id from customers WHERE username = ?)";
312             char* sqliteInsert = const_cast<char*>(sql.c_str());
313             int count = 0;
314             rc = sqlite3_exec(db, sqliteInsert, callBackLogin, &count, &zErrMsg);

315             if (count > 0) {

316                 sql = "SELECT charge as TOTAL , sessionTime as TIME FROM charges WHERE customer_id = (SELECT id from customers WHERE username = ?)";
317                 char* sqliteInsert = const_cast<char*>(sql.c_str());
318                 rc = sqlite3_exec(db, sqliteInsert, callback, 0, &zErrMsg);

319                 if (rc != SQLITE_OK) {
320                     fprintf(stderr, "SQL error: %s\n", zErrMsg);
321                     sqlite3_free(zErrMsg);
322                 }
323             }
324         }
325     }
326 }
```

Definition of check Charge Time for Customer for which takes username of customer and prints its total charge and total session timing.

```
361     fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
362 
363     else {
364         fprintf(stderr, "Opened database successfully\n");
365 
366         sql = "SELECT * FROM customers WHERE username = '" + username + "'";
367 
368         char* sqliteInsert = const_cast<char*>(sql.c_str());
369 
370         rc = sqlite3_exec(db, sqliteInsert, callback, 0, &zErrMsg);
371 
372         if (rc != SQLITE_OK) {
373             fprintf(stderr, "SQL error: %s\n", zErrMsg);
374             sqlite3_free(zErrMsg);
375         }
376         else {
377             fprintf(stdout, "\nQuery Executed successfully\n");
378         }
379 
380         sqlite3_close(db);
381 
382     }
383 
384     //Delete a customer record from database
385     void Owner::deleteCustomer() {
386 
387         rc = sqlite3_open("cybercafemanagementsystem.db", &db);
388 
389         if (rc != SQLITE_OK) {
390             fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
391 
392         }
393         else {
394             fprintf(stderr, "Opened database successfully\n");
395             cout << "Enter the username of customer to be searched";
396             cin >> username;
397             int checkU = checkUsername(username, 2);
398             if (checkU == -1) {
399                 sql = "Delete FROM customers WHERE username = '" + username + "'";
400 
401                 char* sqliteInsert = const_cast<char*>(sql.c_str());
402 
403                 rc = sqlite3_exec(db, sqliteInsert, 0, 0, &zErrMsg);
404 
405                 if (rc != SQLITE_OK) {
406                     fprintf(stderr, "SQL error: %s\n", zErrMsg);
407                     sqlite3_free(zErrMsg);
408                 }
409                 else {
410                     fprintf(stdout, "\nCustomer Deleted successfully\n");
411                 }
412 
413             }
414             else {
415                 cout << "No such username found" << endl;
416             }
417 
418         }
419 
420     }
421 }
```

```
391     if (rc != SQLITE_OK) {
392         fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
393     }
394     else {
395         fprintf(stderr, "Opened database successfully\n");
396         cout << "Enter the username of customer to be searched";
397         cin >> username;
398         int checkU = checkUsername(username, 2);
399         if (checkU == -1) {
400             sql = "Delete FROM customers WHERE username = '" + username + "'";
401 
402             char* sqliteInsert = const_cast<char*>(sql.c_str());
403 
404             rc = sqlite3_exec(db, sqliteInsert, 0, 0, &zErrMsg);
405 
406             if (rc != SQLITE_OK) {
407                 fprintf(stderr, "SQL error: %s\n", zErrMsg);
408                 sqlite3_free(zErrMsg);
409             }
410             else {
411                 fprintf(stdout, "\nCustomer Deleted successfully\n");
412             }
413 
414         }
415         else {
416             cout << "No such username found" << endl;
417         }
418     }
419 
```

Function definition of delete customer by username from customer table.

The screenshot shows the Microsoft Visual Studio IDE interface. The main window displays the code for the `Owner::showCustomerList` function in the `Owner.cpp` file. The code uses SQLite3 to query a database named `cybercafemanagementsystem.db` to retrieve all customer data. It handles errors by printing them to `stderr` and exits if no user is found. The Solution Explorer on the right shows the project structure for the CyberCafeManagementSystem.

```
421     cout << "No such username found" << endl;
422     exit(0);
423 }
424 }
425 }
426 }
427 sqlite3_close(db);
428 }
429 }
430 //Display whole data of customers came into the shop
431 void Owner::showCustomerList(void) {
432
433     rc = sqlite3_open("cybercafemanagementsystem.db", &db);
434
435     if (rc) {
436         fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
437     }
438     else {
439         fprintf(stderr, "Opened database successfully\n");
440
441         sql = "Select * FROM customers";
442
443         char* sqliteInsert = const_cast<char*>(sql.c_str());
444
445         rc = sqlite3_exec(db, sqliteInsert, callback, 0, &zErrMsg);
446
447         if (rc != SQLITE_OK) {
448
449             if (rc != SQLITE_OK) {
450                 fprintf(stderr, "SQL error: %s\n", zErrMsg);
451                 sqlite3_free(zErrMsg);
452             }
453             else {
454                 fprintf(stdout, "\nQuery Executed successfully\n");
455             }
456         }
457         sqlite3_close(db);
458     }
459 }
460 }
461 }
462
463 //Used to allocate a vacant computer to customer
464 void Owner::allocateComputer(void) {
465
466     string cid;
467
468     rc = sqlite3_open("cybercafemanagementsystem.db", &db);
469
470     if (rc) {
471         fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
472     }
473     else {
474
475         cout << "COMPUTER ALLOCATION" << endl;
476
477         sql = "SELECT COUNT(*) as count FROM computer WHERE status = 'N'";
478         char* sqliteInsert = const_cast<char*>(sql.c_str());
479
480         rc = sqlite3_exec(db, sqliteInsert, callback, 0, &zErrMsg);
481
482         if (rc != SQLITE_OK) {
483
484             if (rc != SQLITE_OK) {
485                 fprintf(stderr, "SQL error: %s\n", zErrMsg);
486                 sqlite3_free(zErrMsg);
487             }
488             else {
489                 fprintf(stdout, "\nQuery Executed successfully\n");
490             }
491         }
492         sqlite3_close(db);
493     }
494 }
```

## Function definition of show list of all customers.

The screenshot shows the Microsoft Visual Studio IDE interface. The main window displays the code for the `Owner::allocateComputer` function in the `Owner.cpp` file. This function also uses SQLite3 to interact with the same database. It prints a message for computer allocation and performs a query to count the number of computers with status 'N'. The Solution Explorer on the right shows the project structure for the CyberCafeManagementSystem.

```
451     if (rc != SQLITE_OK) {
452         fprintf(stderr, "SQL error: %s\n", zErrMsg);
453         sqlite3_free(zErrMsg);
454     }
455     else {
456         fprintf(stdout, "\nQuery Executed successfully\n");
457     }
458     sqlite3_close(db);
459 }
460 }
461 }
462
463 //Used to allocate a vacant computer to customer
464 void Owner::allocateComputer(void) {
465
466     string cid;
467
468     rc = sqlite3_open("cybercafemanagementsystem.db", &db);
469
470     if (rc) {
471         fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
472     }
473     else {
474
475         cout << "COMPUTER ALLOCATION" << endl;
476
477         sql = "SELECT COUNT(*) as count FROM computer WHERE status = 'N'";
478         char* sqliteInsert = const_cast<char*>(sql.c_str());
479
480         rc = sqlite3_exec(db, sqliteInsert, callback, 0, &zErrMsg);
481
482         if (rc != SQLITE_OK) {
483
484             if (rc != SQLITE_OK) {
485                 fprintf(stderr, "SQL error: %s\n", zErrMsg);
486                 sqlite3_free(zErrMsg);
487             }
488             else {
489                 fprintf(stdout, "\nQuery Executed successfully\n");
490             }
491         }
492         sqlite3_close(db);
493     }
494 }
```

```
481     int count = 0;
482     rc = sqlite3_exec(db, sqliteInsert, callBackLogin, &count, &zErrMsg);
483
484     if (count > 0) {
485
486         cout << "LIST OF IDLE COMPUTERS" << endl;
487         sql = "SELECT * FROM computer WHERE status = 'N'";
488
489         char* sqliteInsert = const_cast<char*>(sql.c_str());
490         rc = sqlite3_exec(db, sqliteInsert, callback, 0, &zErrMsg);
491
492         if (rc != SQLITE_OK) {
493             fprintf(stderr, "SQL error: %s\n", zErrMsg);
494             sqlite3_free(zErrMsg);
495         }
496     }
497     else {
498
499         cout << "Enter username of registered customer." << endl;
500         cin >> username;
501
502         int checkU = checkUsername(username, 2);
503         if (checkU == 1) {
504             cout << "Customer for this username does not exist in Database. Register first." << endl;
505             exit(0);
506         }
507     }
508
509     else {
510
511         sql = "SELECT COUNT(*) as count FROM charges WHERE customer_id = (SELECT id from customers WHERE
```

The screenshot shows the Microsoft Visual Studio IDE interface. The top menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and Search (Ctrl+Q). The title bar displays "CyberCafeManagementSystemMain". The left sidebar shows the Solution Explorer with files like Computer.h, Computer.cpp, Customer.h, Customer.cpp, Owner.h, Owner.cpp, and Shell.h. The main code editor window shows the "Owner.cpp" file with the following content:

```
511     char* sqliteInsert = const_cast<char*>(sql.c_str());
512
513     int count = 0;
514     rc = sqlite3_exec(db, sqliteInsert, callBackLogin, &count, &zErrMsg);
515
516
517
518
519     if (count == 1) {
520
521         cout << "Computer of following ID has already been assigned to this customer of username " <
522             sql = "SELECT computer_id as ID FROM charges WHERE customer_id=(SELECT id from customers WHE
523             char* sqliteInsert = const_cast<char*>(sql.c_str());
524
525
526             rc = sqlite3_exec(db, sqliteInsert, callback, 0, &zErrMsg);
527             if (rc != SQLITE_OK) {
528                 fprintf(stderr, "SQL error: %s\n", zErrMsg);
529                 sqlite3_free(zErrMsg);
530             }
531             exit(0);
532
533
534     }
535     else {
536
537         cout << "Enter ID from list of above idle computer list." << endl;
538         cin >> cid;
539
540
541 }
```

```
cin >> cid;

        create_date = getCurrDate();

    sql = "INSERT INTO charges (customer_id , computer_id , charge , sessionTime , date) VALUES( ";
    char* sqliteInsert = const_cast<char*>(sql.c_str());
    rc = sqlite3_exec(db, sqliteInsert, callback, 0, &zErrMsg);

    if (rc != SQLITE_OK) {
        fprintf(stderr, "SQL error: %s\n", zErrMsg);
        sqlite3_free(zErrMsg);
    }
    else {

        sql = "UPDATE computer SET status='Y' WHERE id = '" + cid + "'";

        char* sqliteInsert = const_cast<char*>(sql.c_str());
        rc = sqlite3_exec(db, sqliteInsert, callback, 0, &zErrMsg);

        if (rc != SQLITE_OK) {
            fprintf(stderr, "SQL error: %s\n", zErrMsg);
            sqlite3_free(zErrMsg);
        }
        else {
            fprintf(stderr, "Computer allocated successfully\n");
        }
    }
}
```

```
Owner.h User.cpp User.h Owner.cpp x CyberCafeManagementSystemMain.cpp* pymath.h
CyberCafeManagementSystemMain (Global Scope)
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583     else {
584
585         cout << "NO IDLE COMPUTER FOUND." << endl;
586
587     }
588
589
590
591
592 }
593
594 //Return the index of computer
595 Computer* Owner::returnAllocatedComputerAddress(int id) {
596
597     return &compArr[id - 1];
598 }
```

```

Microsoft Visual Studio Debug Console
Owner: Object Created
OWNER ALLOCATION
LIST OF IDLE COMPUTERS
id = 2
description = description2
status = N
created_by = 1

id = 3
description = description3
status = N
created_by = 1

id = 4
description = description4
status = N
created_by = 1

Enter username of registered customer.
test
Enter ID from list of above idle computer list.
2
Computer allocated successfully

D:\CCMS\cyber_cafe_new\Debug\cyber_cafe_new.exe (process 11764) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .

```

**Function definition to allocate computers to registered customer from idle computer list.**

```

592     }
593     }
594     //Return the index of computer
595     Computer* Owner::returnAllocatedComputerAddress(int id) {
596
597         return &compArr[id - 1];
598
599     }
600
601     //getter method to get time
602     Eint Owner::getTime() {
603         return num_of_min;
604     }
605
606     //Method to see list of owners into the database
607     void Owner::showOwnerList(void) {
608
609         rc = sqlite3_open("cybercafemanagementsystem.db", &db);
610
611         if (rc) {
612             fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
613         }
614         else {
615             fprintf(stderr, "Opened database successfully\n");
616
617             sql = "Select * FROM owner";
618
619             char* sqliteInsert = const_cast<char*>(sql.c_str());
620
621
622

```

```
619     char* sqliteInsert = const_cast<char*>(sql.c_str());
620
621     rc = sqlite3_exec(db, sqliteInsert, callback, 0, &zErrMsg);
622
623     if (rc != SQLITE_OK) {
624         fprintf(stderr, "SQL error: %s\n", zErrMsg);
625         sqlite3_free(zErrMsg);
626     }
627     else {
628         fprintf(stdout, "\nQuery Executed successfully\n");
629     }
630
631     sqlite3_close(db);
632
633 }
634
635 //Delete owner from database
636
637 void Owner::deleteOwner() {
638
639     rc = sqlite3_open("cybercafemanagementsystem.db", &db);
640
641     if (rc) {
642         fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
643     }
644     else {
645         fprintf(stderr, "Opened database successfully\n");
646         cout << "Enter the username of owner to be deleted";
647         cin >> username;
648
649     }
650
651     cin >> username;
652     int checkU = checkUsername(username, 1);
653     if (checkU == -1) {
654         sql = "Delete FROM owner WHERE username = '" + username + "';";
655
656
657         char* sqliteInsert = const_cast<char*>(sql.c_str());
658
659         rc = sqlite3_exec(db, sqliteInsert, 0, 0, &zErrMsg);
660
661         if (rc != SQLITE_OK) {
662             fprintf(stderr, "SQL error: %s\n", zErrMsg);
663             sqlite3_free(zErrMsg);
664         }
665         else {
666             fprintf(stdout, "\nOwner Deleted successfully\n");
667         }
668
669     }
670
671     else {
672         cout << "No such username found" << endl;
673         exit(0);
674     }
675
676 }
677
678 sqlite3_close(db);
679
```

Function definition to show all owner list and delete owner by username.

```
649     cin >> username;
650     int checkU = checkUsername(username, 1);
651     if (checkU == -1) {
652         sql = "Delete FROM owner WHERE username = '" + username + "';";
653
654
655         char* sqliteInsert = const_cast<char*>(sql.c_str());
656
657         rc = sqlite3_exec(db, sqliteInsert, 0, 0, &zErrMsg);
658
659         if (rc != SQLITE_OK) {
660             fprintf(stderr, "SQL error: %s\n", zErrMsg);
661             sqlite3_free(zErrMsg);
662         }
663         else {
664             fprintf(stdout, "\nOwner Deleted successfully\n");
665         }
666
667     }
668
669     else {
670         cout << "No such username found" << endl;
671         exit(0);
672     }
673
674 }
675
676
677
678 sqlite3_close(db);
679
```

The screenshot shows the Microsoft Visual Studio IDE interface. The top menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and Search (Ctrl+Q). The tabs at the top show Owner.h, User.cpp, User.h, Owner.cpp (selected), CyberCafeMa\_stemMain.cpp, and pymath.h. The main code editor displays the following C++ code:

```
679 }
680 //Update owner profile
681 void Owner::updateProfile(void)
682 {
683     rc = sqlite3_open("cybercafemanagementsystem.db", &db);
684
685     if (rc) {
686         fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
687     }
688     else {
689         fprintf(stderr, "Opened database successfully\n");
690         int choice = 0;
691         cout << "What do you want to update\n 1. Update Name \n 2. Update Email Address \n 3. Update Contact Number";
692         cin >> choice;
693
694         switch (choice)
695         {
696             case 1:
697             {
698                 cout << "\nEnter your First Name";
699                 cin >> fname;
700                 cout << "\nEnter your Last Name";
701                 cin >> lname;
702                 name = fname + " " + lname;
703
704                 sql = "UPDATE owner SET name = '" + name + "' WHERE isActive='Y'"; //Owner Update
705
706
707
708
709             char* sqliteInsert = const_cast<char*>(sql.c_str());
```

The screenshot continues the code from the previous window. The code editor shows the continuation of the updateProfile function:

```
709
710
711
712     rc = sqlite3_exec(db, sqliteInsert, 0, 0, &zErrMsg);
713
714
715     if (rc != SQLITE_OK) {
716         fprintf(stderr, "SQL error: %s\n", zErrMsg);
717         sqlite3_free(zErrMsg);
718     }
719     else {
720         fprintf(stdout, "\nQuery Executed successfully\n");
721     }
722
723
724 }
725 break;
726 case 2:
727 {
728     cout << "\nEnter Email Address" << endl;
729     cin >> email;
730
731     sql = "UPDATE owner SET email = '" + email + "' WHERE isActive='Y'"; //Owner Update
732
733
734
735
736     char* sqliteInsert = const_cast<char*>(sql.c_str());
737
738
739     rc = sqlite3_exec(db, sqliteInsert, 0, 0, &zErrMsg);
```

```
739     rc = sqlite3_exec(db, sqliteInsert, 0, 0, &zErrMsg);
740
741     if (rc != SQLITE_OK) {
742         fprintf(stderr, "SQL error: %s\n", zErrMsg);
743         sqlite3_free(zErrMsg);
744     }
745     else {
746         fprintf(stdout, "\nQuery Executed successfully\n");
747     }
748     break;
749     case 3:
750     {
751         cout << "\nEnter your Contact Number";
752         cin >> mobile_no;
753
754         sql = "UPDATE owner SET mobile_no = '" + mobile_no + "' WHERE isActive='Y'"; //Owner Update
755
756         const char (32) UPDATE owner SET mobile_no =
757
758         Search Online
759
760         char* sqliteInsert = const_cast<char*>(sql.c_str());
761
762         rc = sqlite3_exec(db, sqliteInsert, 0, 0, &zErrMsg);
763
764         if (rc != SQLITE_OK) {
765             fprintf(stderr, "SQL error: %s\n", zErrMsg);
766             sqlite3_free(zErrMsg);
767         }
768     }
769 }
```

Function definition for update profile for owner where owner can update its name, mobile number, and email.

```
769     sqlite3_free(zErrMsg);
770 }
771 else {
772     fprintf(stdout, "\nQuery Executed successfully\n");
773 }
774 break;
775 case 4:
776 {
777 }
778
779 }
780
781
782
783
784     sqlite3_close(db);
785
786 }
787
788 }
789
790 }
791 //Get all the charges paid by customers
792 void Owner::getAllCharges(void) {
793     rc = sqlite3_open("cybercafemanagementsystem.db", &db);
794
795     if (rc) {
796         fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
797     }
798     else {
799         fprintf(stderr, "Opened database successfully\n");
800     }
801 }
```

**Screenshot 1: Code Editor - Owner.cpp**

```

793     rc = sqlite3_open("cybercafemanagementsystem.db", &db);
794
795     if (rc) {
796         fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
797     } else {
798         fprintf(stderr, "Opened database successfully\n");
799
800         sql = "Select * FROM charges";
801
802         char* sqliteInsert = const_cast<char*>(sql.c_str());
803
804         rc = sqlite3_exec(db, sqliteInsert, callback, 0, &zErrMsg);
805
806         if (rc != SQLITE_OK) {
807             fprintf(stderr, "SQL error: %s\n", zErrMsg);
808             sqlite3_free(zErrMsg);
809         } else {
810             fprintf(stderr, "\nQuery Executed successfully\n");
811         }
812     }
813
814     sqlite3_close(db);
815
816 }
817
818
819
820
821
822
823

```

**Screenshot 2: Code Editor - CyberCafeManagementSystemMain.cpp**

```

22
23     Customer pC1;
24     Computer c;
25
26     pObj = new Owner(2);
27     pObj->searchCustomer();
28     pObj->deleteCustomer();
29     pObj->showCustomerList();
30     pObj->showOwnerList();
31     pObj->updateProfile();
32
33     /*
34     pObj->addComputer();
35     pObj->searchComputer();
36     pObj->listAllComputers();
37     pObj->deleteComputer();
38     pObj->updateComputer();
39     */
40     //pC1 = new Customer(pObj);
41     //pC1->registerUser();
42     //pObj->showCustomerList();
43

```

**Solution Explorer**

- CyberCafeManagementSystem
  - External Dependencies
  - Header Files
  - Resource Files
  - Source Files
    - Computer.cpp
    - Customer.cpp
    - CyberCafeManagementSystemMain.cpp
    - Owner.cpp
    - Owner.h
    - PythonDBCObject.h
    - shell.c
    - sqlite3.c
    - User.cpp
    - User.h

**Properties**

```

Microsoft Visual Studio Debug Console
Owner: Object Created
Enter the username of customer to be searchedPranjal123
Opened database successfully
id = 3
name = Pranjal Enchaluwar
email = enchaluwarpranjal@gmail.com
mobile_no = 8359025929
username = Pranjal123
password = 1234
created_by = NULL
create_date = 28-12-2021

Query Executed successfully
Opened database successfully
Enter the username of customer to be searchedPranjal123

Customer Deleted successfully
Opened database successfully
id = 3
name = Akash Shikare
email = akashshikare@gmail.com
mobile_no = 9896234567
username = Akash123
password = 1234
created_by = 1
create_date = 29-12-2021

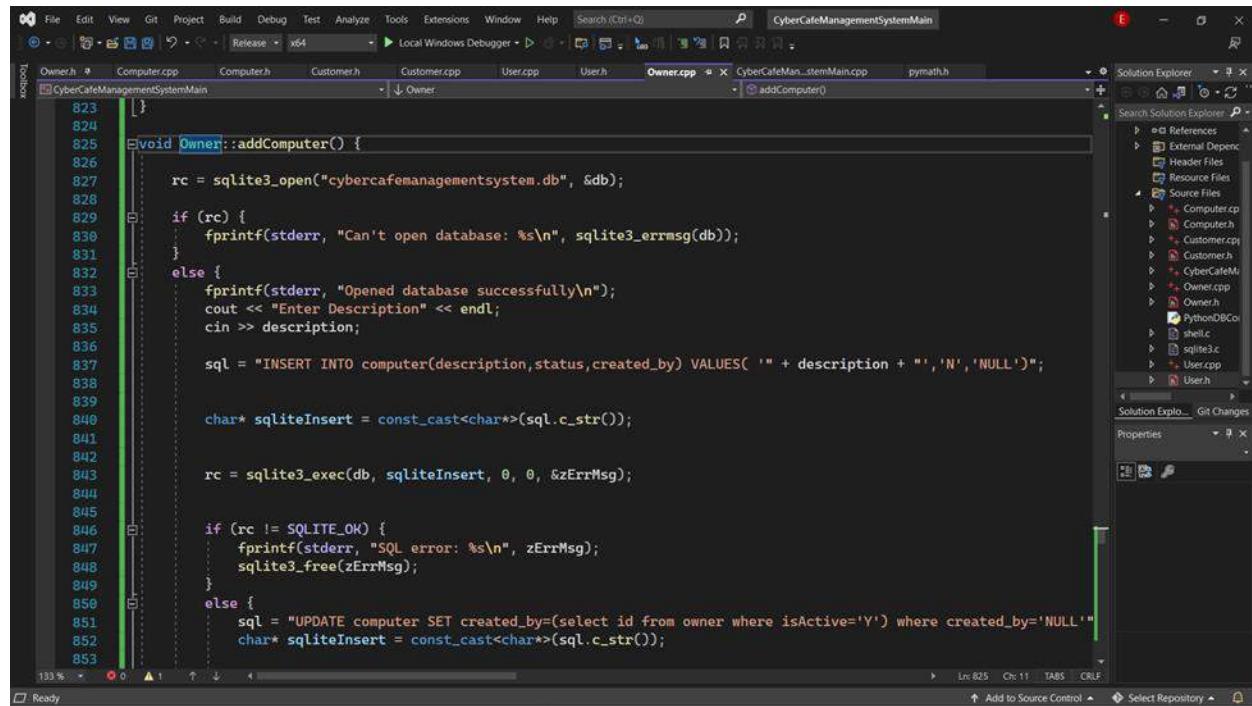
Query Executed successfully
Opened database successfully
id = 3
name = Kadir Sheikh
email = sheikhkadir02@gmail.com
mobile_no = 7972112894
username = kadir123
password = 1234
create_date = 28-12-2021
isactive = Y

Query Executed successfully
Opened database successfully
what Do you want to update
1. Update Name
2. Update Email Address
3. Update Contact Number
4. exit 1

Enter your First NameKadir R Sheikh
Enter your Last Name

```

Function definition to get all charges which prints a charges table that shows entry of session time, charge for that session and the computer and customer id.



```

823
824
825     void Owner::addComputer() {
826
827         rc = sqlite3_open("cybercafemanagementsystem.db", &db);
828
829         if (rc) {
830             fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
831         }
832         else {
833             fprintf(stderr, "Opened database successfully\n");
834             cout << "Enter Description" << endl;
835             cin >> description;
836
837             sql = "INSERT INTO computer(description,status,created_by) VALUES( '" + description + "','N','NULL')";
838
839
840             char* sqliteInsert = const_cast<char*>(sql.c_str());
841
842
843             rc = sqlite3_exec(db, sqliteInsert, 0, 0, &zErrMsg);
844
845
846             if (rc != SQLITE_OK) {
847                 fprintf(stderr, "SQL error: %s\n", zErrMsg);
848                 sqlite3_free(zErrMsg);
849             }
850             else {
851                 sql = "UPDATE computer SET created_by=(select id from owner where isActive='Y') where created_by='NULL'";
852                 char* sqliteInsert = const_cast<char*>(sql.c_str());
853             }
}

```

This screenshot shows the Microsoft Visual Studio IDE interface with the code editor displaying the `Owner.cpp` file. The code implements a method to add a computer to a SQLite database. It includes error handling for database operations and user input validation.

```
853     rc = sqlite3_exec(db, sqliteInsert, 0, 0, &zErrMsg);
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 sqlite3_close(db);
862 }
863 }
864 //Method to search the computer from list of computers
865 void Owner::searchComputer() {
866     rc = sqlite3_open("cybercafemanagementsystem.db", &db); //Open database and store it into rc pointer
867     if (rc) {
868         fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db)); //check on whether database opened or not
869     } else {
870         fprintf(stderr, "Opened database successfully\n"); //if database is open perform required operations
871         string cid;
872         cout << "Enter Id of Computer";
873         cin >> cid;
874         int check = checkId(cid);
875         if (check == -1) {
876             cout << "Id does not exists try with different id";
877             exit(0);
878         } else {
879             if (rc != SQLITE_OK) { //Checks query is valid or not
880                 fprintf(stderr, "SQL error: %s\n", zErrMsg);
881             }
882         }
883     }
884 }
```

This screenshot continues the code for the `searchComputer()` function. It reads the user-entered computer ID, performs a query to find the computer record, and prints the result to the console.

```
871     fprintf(stderr, "Opened database successfully\n"); //if database is open perform required operations
872     string cid;
873     cout << "Enter Id of Computer";
874     cin >> cid;
875     int check = checkId(cid);
876     if (check == -1) {
877         cout << "Id does not exists try with different id";
878         exit(0);
879     } else {
880
881         if (rc != SQLITE_OK) { //Checks query is valid or not
882             fprintf(stderr, "SQL error: %s\n", zErrMsg);
883             sqlite3_free(zErrMsg); //returns error with the query
884         } else {
885             sql = "SELECT * from computer where id='"
886             + cid + "'";
887
888             char* sqliteInsert = const_cast<char*>(sql.c_str()); //Convert char pointer to string
889
890             rc = sqlite3_exec(db, sqliteInsert, callback, 0, &zErrMsg); //Execute query on database
891             cout << "Successfully shown the result" << endl;
892         }
893     }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
```

Screenshot of Visual Studio showing the `Owner.cpp` file code. The code implements methods for managing computers in a SQLite database. It includes functions for adding a computer, listing all computers, checking if an ID exists, and deleting a computer.

```
898     }
899     sqlite3_close(db); //closes the database after execution of query
900
901 }
902 //Method to list all the computers into the list
903 void Owner::listAllComputers() {
904     rc = sqlite3_open("cybercafemanagementsystem.db", &db);
905     if (rc) {
906         fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
907     } else {
908         fprintf(stderr, "Opened database successfully\n");
909
910         sql = "SELECT * from computer";
911
912         char* sqliteInsert = const_cast<char*>(sql.c_str());
913
914         rc = sqlite3_exec(db, sqliteInsert, callback, 0, &zErrMsg);
915
916         if (rc != SQLITE_OK) {
917             fprintf(stderr, "SQL error: %s\n", zErrMsg);
918             sqlite3_free(zErrMsg);
919         } else {
920             cout << "Successfully shown the result" << endl;
921         }
922     }
923 }
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961 }
```

Continuation of the `Owner.cpp` file code. It includes the implementation of the `checkId` method, which queries the database to check if a given ID exists. It also includes the `deleteComputer` method, which deletes a computer from the database.

```
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2289
2290
2291
2292
2293
2294
2295
2296
2297
2297
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2489
2490
2491
2492
2493
2494
2495
2496
2497
2497
2498
2499
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2598
2599
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2698
2699
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2798
2799
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2898
2899
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2959
2960
2961
2962
2963
2964

```

```
961     void Owner::deleteComputer() {
962         rc = sqlite3_open("cybercafemanagementsystem.db", &db);
963         string cid;
964         if (rc) {
965             fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
966         }
967         else {
968             fprintf(stderr, "Opened database successfully\n");
969             cout << "Enter Id of computer to be deleted" << endl;
970             cin >> cid;
971             int check = checkId(cid);
972             if (check == -1) {
973                 cout << "Id doesn't exist Try again with different id." << endl;
974                 exit(0);
975             }
976             else {
977                 sql = "DELETE FROM computer WHERE id like '" + cid + "'";
978
979                 char* sqliteInsert = const_cast<char*>(sql.c_str());
980
981                 rc = sqlite3_exec(db, sqliteInsert, 0, 0, &zErrMsg);
982
983                 if (rc != SQLITE_OK) {
984                     fprintf(stderr, "SQL error: %s\n", zErrMsg);
985                     sqlite3_free(zErrMsg);
986                 }
987                 else {
988                     cout << "Computer successfully deleted" << endl;
989                 }
990             }
991         }
992     }
993
994
995
996     }
997     sqlite3_close(db);
998
999 }
1000 //Method to update computer description
1001 void Owner::updateComputer() {
1002     rc = sqlite3_open("cybercafemanagementsystem.db", &db);
1003     string cid;
1004     if (rc) {
1005         fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
1006     }
1007     else {
1008         fprintf(stderr, "Opened database successfully\n");
1009         cout << "Enter Id of computer to be updated" << endl;
1010         cin >> cid;
1011         int check = checkId(cid);
1012         if (check == -1) {
1013             cout << "Id doesn't exist Try again with different id." << endl;
1014             exit(0);
1015         }
1016         else {
1017             string newDescription;
1018             cout << "Enter the new description" << endl;
1019             cin >> newDescription;
1020             sql = "UPDATE computer SET description = '" + newDescription + "' WHERE id like '" + cid + "'";
1021         }
1022     }
1023 }
```

The screenshot shows the Microsoft Visual Studio IDE interface. The code editor displays a file named `Owner.cpp` with line numbers from 1015 to 1043. The code implements a function to update a computer's description in a SQLite database. The Solution Explorer on the right lists the project files: `CyberCafeManagementSystemMain`, `Owner.h`, `Computer.cpp`, `Computer.h`, `Customer.h`, `Customer.cpp`, `User.cpp`, `User.h`, `CyberCafeMan...stemMain.cpp`, and `pymath.h`. The Properties window is also visible.

```
1015 }  
1016 else {  
1017     string newDescription;  
1018     cout << "Enter the new description" << endl;  
1019     cin >> newDescription;  
1020     sql = "UPDATE computer SET description = '" + newDescription + "' where id like '" + cid + "'";  
1021  
1022     char* sqliteInsert = const_cast<char*>(sql.c_str());  
1023  
1024     rc = sqlite3_exec(db, sqliteInsert, 0, 0, &zErrMsg);  
1025  
1026     if (rc != SQLITE_OK) {  
1027         fprintf(stderr, "SQL error: %s\n", zErrMsg);  
1028         sqlite3_free(zErrMsg);  
1029     }  
1030     else {  
1031         cout << "Computer successfully updated" << endl;  
1032     }  
1033 }  
1034  
1035 sqlite3_close(db);  
1036  
1037  
1038  
1039  
1040  
1041  
1042  
1043 }
```

## Execution

```
Customer* pC1;
Computer c;

pObj = new Owner(2);
/* pObj->searchCustomer();
pObj->deleteCustomer();
pObj->showCustomerList();
pObj->showOwnerList();
pObj->updateProfile();
*/
pObj->addComputer();
pObj->searchComputer();
pObj->listAllComputers();
pObj->deleteComputer();
pObj->updateComputer();

//pC1 = new Customer(pObj);
//pC1->registerUser();
//pObj->showCustomerList();
/**/

133 % No issues found
```

```
Show output from: Debug
'C:\Windows\System32\vcruntime140.dll'.
The thread 0x26c has exited with code 0 (0x0).
'C:\Windows\System32\kernel32.dll'.
'C:\Windows\System32\kernel.appcore.dll'.
'C:\Windows\System32\msvcr7.dll'.
The thread 0x1c4 has exited with code 0 (0x0).
The thread 0x26c has exited with code 0 (0x0).
The program '[13664] CyberCafeManagementSystemMain.exe' has exited with code 0 (0x0).
```

```
Ready
Microsoft Visual Studio Debug Console
Owner Object Created
Opened database successfully
Enter Description
Computer2
Opened database successfully
Enter Id of Computer1
id = 1
description = Computer1
status = N
created_by = 1

Successfully shown the result
Opened database successfully
id = 1
description = Computer1
status = N
created_by = 1

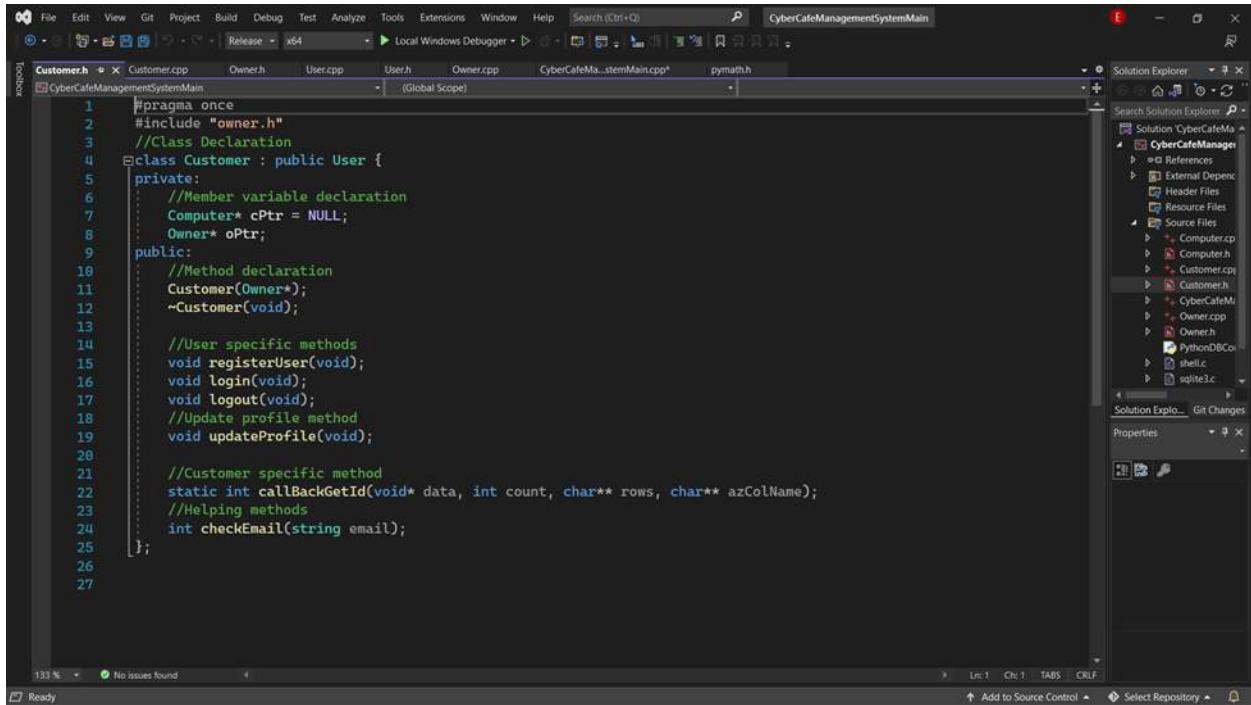
id = 2
description = Computer2
status = N
created_by = 1

Successfully shown the result
Opened database successfully
Enter Id of computer to be deleted
1
Computer successfully deleted
Opened database successfully
Enter Id of computer to be updated
2
Enter the new description
Computer2.0
Computer successfully updated

C:\Users\Acer\source\repos\CyberCafeManagementSystemMain\x64\Release\CyberCafeManagementSystemMain.exe (process 13664) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

## Customer

## Class declaration



The screenshot shows the Microsoft Visual Studio IDE interface. The left pane displays the code for `Customer.h`, which contains the class definition for `Customer`. The right pane shows the `Solution Explorer` with the project structure for `CyberCafeManagementSystemMain`, including files like `Customer.cpp`, `Owner.h`, and `User.h`.

```
1 #pragma once
2 #include "owner.h"
3 //Class Declaration
4 class Customer : public User {
5 private:
6     //Member variable declaration
7     Computer* cPtr = NULL;
8     Owner* oPtr;
9 public:
10    //Method declaration
11    Customer(Owner*);
12    ~Customer(void);
13
14    //User specific methods
15    void registerUser(void);
16    void login(void);
17    void logout(void);
18    //Update profile method
19    void updateProfile(void);
20
21    //Customer specific method
22    static int callBackGetId(void* data, int count, char** rows, char** azColName);
23    //Helping methods
24    int checkEmail(string email);
25};
26
27
```

Function declaration for customer constructor, destructor, customer authentication functions, update profile for customer, check email.

## Class definition

This screenshot shows the Visual Studio IDE interface with the code editor displaying the `Customer.cpp` file. The code implements the `registerUser` method for the `Customer` class. It starts by opening a SQLite database and checking if it's successfully opened. If successful, it retrieves the current date and prompts the user to enter their first name. The code then enters a loop where it prompts for last name, email, mobile number, and username, and checks if they already exist in the database.

```
1 #include <iostream>
2 #include <windows.h>
3 #include "Customer.h"
4 #include<sstream>
5
6 using namespace std;
7
8 Customer::Customer(Owner* _oPtr) : oPtr(_oPtr) {} //Constructor for customer class
9
10 Customer::~Customer(void) {} //Destructor for customer class
11
12 // User specific method definition (inherited)
13 // Method to implement customer registration
14 void Customer::registerUser(void) {
15
16     rc = sqlite3_open("cybercafemanagementsystem.db", &db);
17
18     if (rc) {
19         fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
20     }
21     else {
22         fprintf(stderr, "Opened database successfully\n");
23
24
25         create_date = getCurDate();
26
27         cout << "CUSTOMER REGISTRATION" << endl;
28
29         cout << "Enter first name" << endl;
30         cin >> fname;
31
32
33
34         cout << "Enter last name" << endl;
35         cin >> lname;
36
37
38         name = fname + " " + lname;
39
40         cout << "Enter email" << endl;
41         cin >> email;
42
43         int check = checkEmail(email);
44         if (check == -1) {
45             cout << "Email already exists in database. Try again with different email." << endl;
46             exit(0);
47         }
48
49
50         cout << "Enter mobile" << endl;
51         cin >> mobile_no;
52
53
54         cout << "Enter username" << endl;
55         cin >> username;
56
57         int checkU = checkUsername(username, 2);
58         if (checkU == -1) {
59             cout << "Username already exists in database. Try again." << endl;
60             exit(0);
61     }
```

This screenshot continues the code editor view of the `Customer.cpp` file, showing the completion of the `registerUser` method. After exiting the loop, the code prints a success message and exits the application. The rest of the file contains standard header and footer boilerplate.

```
62
63     cout << "Registration successful" << endl;
64
65     cout << "Press any key to exit" << endl;
66     getch();
67 }
```

This screenshot shows the Microsoft Visual Studio IDE interface with the Customer.cpp file open in the main editor window. The code implements a registration function for a customer. It prompts the user for name, email, mobile number, username, and password. It then constructs an SQL INSERT statement to add the new customer to a SQLite database. If the insertion fails, it prints an error message; otherwise, it prints a success message.

```
61
62
63
64
65     cout << "Enter password" << endl;
66     cin >> password;
67
68
69
70     if (name.size() == 0 || email.size() == 0 || mobile_no.size() == 0 || username.size() == 0 || password.size(
71
72         cout << "Field cannot be empty" << endl;
73
74     }
75
76     else {
77
78         sql = "INSERT INTO customers(name, email, mobile_no, username, password, created_by, create_date) VALUES('";
79
80         char* sqliteInsert = const_cast<char*>(sql.c_str());
81         rc = sqlite3_exec(db, sqliteInsert, callback, 0, &zErrMsg);
82
83         if (rc != SQLITE_OK) {
84             fprintf(stderr, "SQL error: %s\n", zErrMsg);
85             sqlite3_free(zErrMsg);
86         }
87         else {
88             sql = "UPDATE customers set created_by=(select id from owner where isActive='Y') where username='";
89
90             char* sqliteUpdate = const_cast<char*>(sql.c_str());
91
92             rc = sqlite3_exec(db, sqliteUpdate, callback, 0, &zErrMsg);
93
94             if (rc != SQLITE_OK) {
95                 fprintf(stderr, "SQL error: %s\n", zErrMsg);
96                 sqlite3_free(zErrMsg);
97             }
98             else {
99                 fprintf(stderr, "Customer Registered successfully\n");
100            }
101
102
103
104
105
106
107
108
109
110
111
112             sqlite3_close(db);
113
114
115     //Method for customer login into system
116     void Customer::login(void) {
117
118         rc = sqlite3_open("cybercafemanagementsystem.db", &db);
119
120         if (rc) {
121             fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
```

This screenshot shows the Microsoft Visual Studio IDE interface with the Customer.cpp file open in the main editor window. The code defines a login method for a customer. It attempts to open a SQLite database named "cybercafemanagementsystem.db". If the database cannot be opened, it prints an error message.

```
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116     void Customer::login(void) {
117
118         rc = sqlite3_open("cybercafemanagementsystem.db", &db);
119
120         if (rc) {
121             fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
```

```

Microsoft Visual Studio Debug Console
Owner Object Created
Opened database successfully
CUSTOMER REGISTRATION
Enter first name
kadir
Enter last name
sınan
Enter email
kadir@gmail.com
Enter mobile
123456789
Enter username
kadir111
Enter password
1234
Customer Registered successfully

D:\CCS\Cyber_cafe_new\Cyber_cafe_new\Debug\Cyber_cafe_new.exe (process 30700) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . .

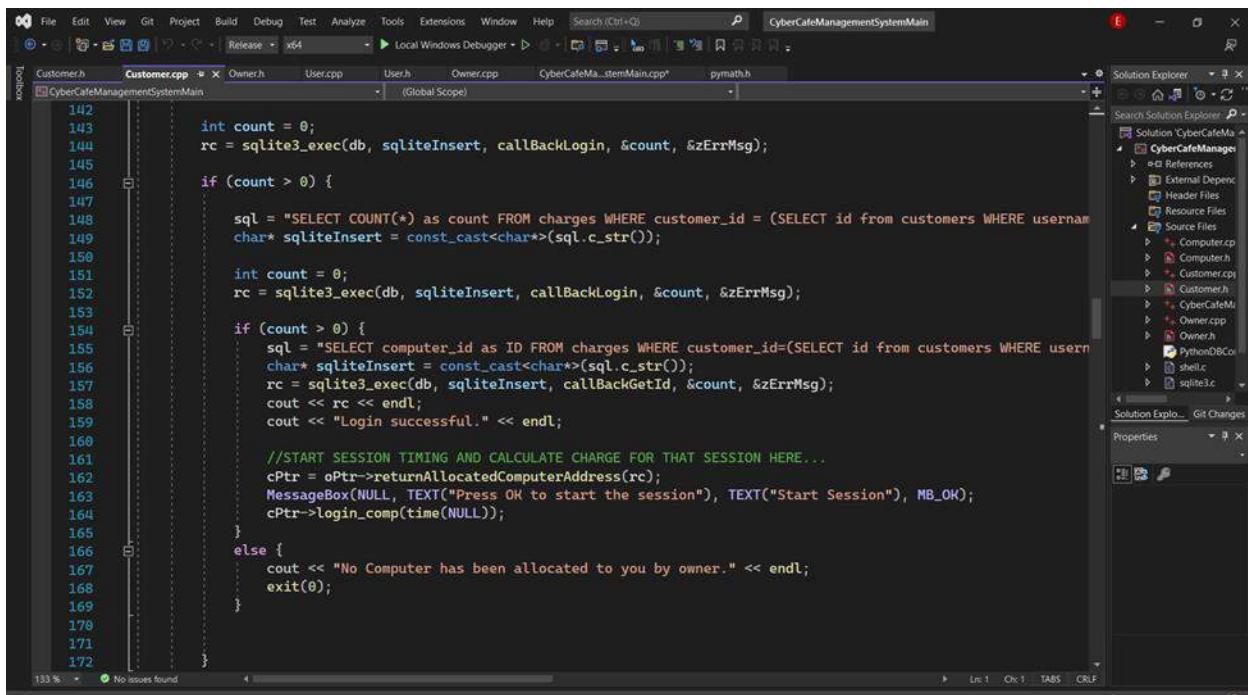
```

**Function definition for customer registration which is overriding User's class virtual function.**

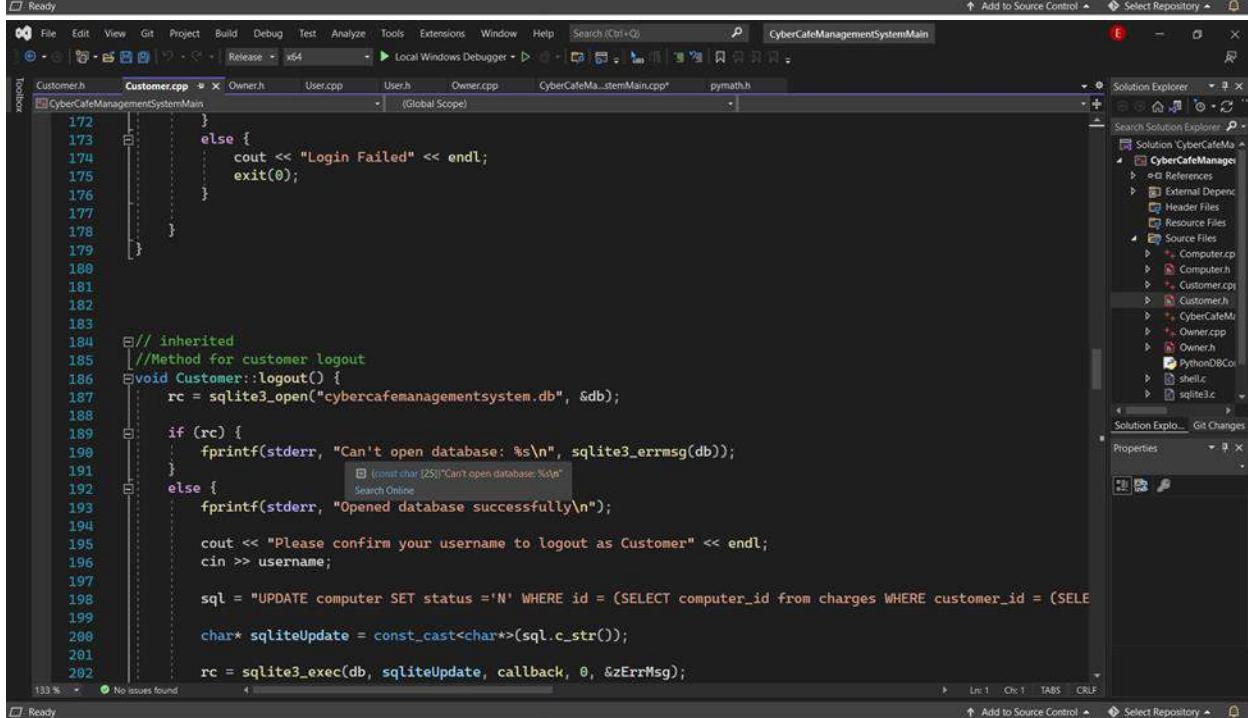
```

112
113
114
115     //Method for customer login into system
116 void Customer::login(void)
117 {
118     rc = sqlite3_open("cybercafemanagementsystem.db", &db);
119
120     if (rc) {
121         fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
122     }
123     else {
124         fprintf(stderr, "Opened database successfully\n");
125
126         cout << "CUSTOMER LOGIN" << endl;
127
128         cout << "Enter Username" << endl;
129         cin >> username;
130
131         cout << "Enter Password" << endl;
132         cin >> password;
133
134
135
136
137         sql = "SELECT COUNT(*) as count FROM customers WHERE username = '" + username + "' AND password = '" + password + "'";
138         //sql = "SELECT * FROM owner";
139
140
141         char* sqliteInsert = const_cast<char*>(sql.c_str());
142

```



```
142     int count = 0;
143     rc = sqlite3_exec(db, sqliteInsert, callBackLogin, &count, &zErrMsg);
144
145     if (count > 0) {
146
147         sql = "SELECT COUNT(*) as count FROM charges WHERE customer_id = (SELECT id from customers WHERE username = ?)";
148         char* sqliteInsert = const_cast<char*>(sql.c_str());
149
150         int count = 0;
151         rc = sqlite3_exec(db, sqliteInsert, callBackLogin, &count, &zErrMsg);
152
153         if (count > 0) {
154             sql = "SELECT computer_id as ID FROM charges WHERE customer_id=(SELECT id from customers WHERE username = ?)";
155             char* sqliteInsert = const_cast<char*>(sql.c_str());
156             rc = sqlite3_exec(db, sqliteInsert, callBackGetId, &count, &zErrMsg);
157             cout << rc << endl;
158             cout << "Login successful." << endl;
159
160             //START SESSION TIMING AND CALCULATE CHARGE FOR THAT SESSION HERE...
161             cPtr = oPtr->returnAllocatedComputerAddress(rc);
162             MessageBox(NULL, TEXT("Press OK to start the session"), TEXT("Start Session"), MB_OK);
163             cPtr->login_comp(time(NULL));
164         }
165         else {
166             cout << "No Computer has been allocated to you by owner." << endl;
167             exit(0);
168         }
169     }
170 }
171 }
```



```
172     }
173     else {
174         cout << "Login Failed" << endl;
175         exit(0);
176     }
177 }
178 }
```

```
179
180
181
182
183
184 // inherited
185 //Method for customer logout
186 void Customer::logout() {
187     rc = sqlite3_open("cybercafemanagementsystem.db", &db);
188
189     if (rc) {
190         fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
191     }
192     else {
193         fprintf(stderr, "Opened database successfully\n");
194
195         cout << "Please confirm your username to logout as Customer" << endl;
196         cin >> username;
197
198         sql = "UPDATE computer SET status ='N' WHERE id = (SELECT computer_id from charges WHERE customer_id = (SELECT id from customers WHERE username = ?))";
199
200         char* sqliteUpdate = const_cast<char*>(sql.c_str());
201
202         rc = sqlite3_exec(db, sqliteUpdate, callback, 0, &zErrMsg);
```

Function definition for customer login which is overriding User's class virtual function.

The screenshot shows the Visual Studio IDE interface with the code editor displaying the `Customer.cpp` file. The code implements a `Logout` function:

```
202     rc = sqlite3_exec(db, sqliteUpdate, callback, 0, &zErrMsg);
203
204     if (rc != SQLITE_OK) {
205         fprintf(stderr, "SQL error: %s\n", zErrMsg);
206         sqlite3_free(zErrMsg);
207     }
208     else {
209
210         int charge = oPtr->generateBill(cPtr);
211         int timeT = oPtr->getTime();
212         sql = "UPDATE charges SET charge ='" + into_String(charge) + "' , sessionTime = '" + into_String(timeT)
213
214         char* sqliteUpdate = const_cast<char*>(sql.c_str());
215
216         rc = sqlite3_exec(db, sqliteUpdate, callback, 0, &zErrMsg);
217
218         if (rc != SQLITE_OK) {
219             fprintf(stderr, "SQL error: %s\n", zErrMsg);
220             sqlite3_free(zErrMsg);
221         }
222         else {
223             fprintf(stdout, "\nLogout Successful\n");
224             oPtr = NULL;
225             cPtr = NULL;
226             delete(this);
227         }
228     }
229 }
230
231 }
```

The screenshot shows the Visual Studio IDE interface with the code editor displaying the `Customer.cpp` file. The code implements a `checkEmail` function:

```
238     //Callback method for ID
239     int Customer:: callBackGetId(void* data, int count, char** rows, char** azColName) {
240
241         int i;
242
243         /*for (i = 0; i < count; i++) {
244             printf("%s = %s\n", azColName[i], rows[i] ? rows[i] : "NULL");
245         }*/
246         return int(rows[0]);
247         //printf("\n");
248
249     }
250
251     //Method for checking email exists into the database
252     int Customer::checkEmail(string email) {
253         rc = sqlite3_open("cybercafemanagementsystem.db", &db);
254         if (rc) {
255             fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
256         }
257         else {
258             //fprintf(stderr, "Opened database successfully\n");
259
260             sql = "SELECT COUNT(*) from customers where email like '" + email + "'";
261             char* sqliteInsert = const_cast<char*>(sql.c_str());
262
263             int count = 0;
264             rc = sqlite3_exec(db, sqliteInsert, callBackLogin, &count, &zErrMsg);
265
266             if (count == 1) {
267                 return -1;
268             }
269             else {
```

Function definition for customer logout which is overriding User's class virtual function.

Customer.h Customer.cpp Owner.h User.cpp User.h Owner.cpp CyberCafeManagementSystemMain pymath.h

```
274 }
275 //Method to update Profile of customer
276 void Customer::updateProfile(void)
277 {
278     rc = sqlite3_open("cybercafemanagementsystem.db", &db);
279
280     if (rc) {
281         fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
282     }
283     else {
284         fprintf(stderr, "Opened database successfully\n");
285         int choice = 0;
286         cout << "What do you want to update\n 1. Update Name \n 2. Update Email Address \n 3. Update Contact Number\n";
287         cin >> choice;
288
289         switch (choice)
290         {
291             case 1:
292                 {
293                     cout << "\nEnter your First Name";
294                     cin >> fname;
295                     cout << "\nEnter your Last Name";
296                     cin >> lname;
297                     name = fname + " " + lname;
298
299                     sql = "UPDATE customers SET name = '" + name + "' WHERE username like '%" + username + "%'; //Customer";
300
301                     char* sqliteInsert = const_cast<char*>(sql.c_str());
302
303                 }
304         }
305     }
306 }
```

Customer.h Customer.cpp Owner.h User.cpp User.h Owner.cpp CyberCafeManagementSystemMain pymath.h

```
313 }
314     else {
315         fprintf(stdout, "\nQuery Executed successfully\n");
316     }
317 }
318 }
319 }
320 }
321 break;
322 case 2:
323 {
324     cout << "\nEnter Email Address" << endl;
325     cin >> email;
326
327     sql = "UPDATE customers SET email = '" + email + "' WHERE username like '%" + username + "%'; //customer";
328
329 }
330 }
331 }
332 }
333 }
334 }
335 }
336 }
337 }
338 }
339 }
340 }
341 }
342 }
343 }
```

```
Customer.h Customer.cpp Owner.h User.cpp User.h Owner.cpp CyberCafeManagementSystemMain.cpp pymath.h
```

```
337:     if (rc != SQLITE_OK) {
338:         fprintf(stderr, "SQL error: %s\n", zErrMsg);
339:         sqlite3_free(zErrMsg);
340:     }
341:     else {
342:         fprintf(stdout, "\nQuery Executed successfully\n");
343:     }
344:     break;
345: case 3:
346: {
347:     cout << "\nEnter your Contact Number";
348:     cin >> mobile_no;
349:
350:     sql = "UPDATE customers SET mobile_no = '" + mobile_no + "' WHERE username like '" + username + "'";
351:
352:
353:
354:
355:
356:     char* sqliteInsert = const_cast<char*>(sql.c_str());
357:
358:
359:
360:
361:
362:     rc = sqlite3_exec(db, sqliteInsert, 0, 0, &zErrMsg);
363:
364:
365:
366:
367:     if (rc != SQLITE_OK) {
368:         fprintf(stderr, "SQL error: %s\n", zErrMsg);
369:         sqlite3_free(zErrMsg);
370:     }
371:     else {
372:         fprintf(stdout, "\nQuery Executed successfully\n");
373:     }
374:
375:
376:
377:
378:
379:     sqlite3_close(db);
380:
381:
382:
383:
384:
385:
386:
387:
```

The screenshot shows the Microsoft Visual Studio IDE interface for the CyberCafeManagementSystemMain project. The Solution Explorer on the right lists files such as Customer.h, Customer.cpp, Computer.h, Computer.cpp, Owner.h, Owner.cpp, PythonDBConfig.h, shell.c, sqlite3.c, User.cpp, and User.h. The Code Editor window displays Customer.cpp with code related to managing computer objects and customer profiles. The Output window shows the results of a debug build, including logs for opening a database and performing a successful login ('Akash123') with password '1234'.

```
Output
Show output from: Debug
The thread 0x2990 has exited with code 0 (0x0).
'CyberCafeManagementSystemMain.exe' (Win32): Loaded 'C:\Windows\System32\kernel32.dll'.
'CyberCafeManagementSystemMain.exe' (Win32): Loaded 'C:\Windows\System32\msvcrt.dll'.
The thread 0x267c has exited with code 0 (0x0).
The thread 0x3710 has exited with code 0 (0x0).
The program '[21676] CyberCafeManagementSystemMain.exe' has exited with code 0 (0x0).

Error List: Output
Ready
```

```
Microsoft Visual Studio Debug Console
Owner Object Created
Opened database successfully
id = 2
name = Akash Shikare
email = akashshikare@gmail.com
mobile_no = 9890234567
username = Akash123
password = 1234
created_by = 1
create_date = 29-12-2021

Query Executed successfully
Opened database successfully
CUSTOMER LOGIN
Enter Username
Akash123
Enter Password
1234
No computer has been allocated to you by owner.

C:\Users\Acer\source\repos\CyberCafeManagementSystemMain\x64\Release\CyberCafeManagementSystemMain.exe (process 21676) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Function definition to update customer profile in which customer can update its name, mobile number, email.

# Computer class

## Class declaration

```
1 #pragma once
2 #include "sqlite/sqlite3.h"
3 #include <ctime>
4 #include<string.h>
5 using namespace std;
6 //Class Declaration
7 class Computer {
8 private:
9     //Member variable declaration
10    int flag = 0;
11    time_t start_time = NULL;
12    time_t end_time = NULL;
13    sqlite3* db;
14    char* zErrMsg = 0;
15    int rc;
16    string sql;
17    string description, status, created_by;
18 public:
19     //Method declaration
20     Computer(void);
21     Computer(time_t, time_t);
22
23     // getters and setters
24     void setStartTime(time_t);
25     void setEndTime(time_t);
26     time_t getStartTime(void);
27     time_t getEndTime(void);
28
29     void login_comp(time_t);
30     void logout_comp(time_t, int*);
31
32
33
34     //callback functions
35     static int callback(void* data, int count, char** rows, char** azColName);
36     static int callBackLogin(void* data, int count, char** rows, char**);
37
38
39
40
41 }
```

```
//Including header files
#include <iostream>
#include <Windows.h>
#include <ctime>
#include "Computer.h"
#include "sqlite/sqlite3.h"

using namespace std;

Computer::Computer(void) {}//Default Constructor

Computer::Computer(time_t _start_time = NULL, time_t _end_time = NULL) : start_time(_start_time), end_time(_end_time)

void Computer::login_comp(time_t _start_time) { start_time = _start_time; } //Function used to set start time for the computer

void Computer::logout_comp(time_t _end_time, int* accessTime) { //Function used to find total session time for the computer
    end_time = _end_time;
    *accessTime = difftime(end_time, start_time);
    start_time = NULL;
    end_time = NULL;
}

//Getter and setter methods for start and ending time
void Computer::setStartTime(time_t _start_time) { start_time = _start_time; }
void Computer::setEndTime(time_t _end_time) { end_time = _end_time; }
time_t Computer::getStartTime(void) { return start_time; }
time_t Computer::getEndTime(void) { return end_time; }


```

Class declaration of Computer which contain declaration of computer data members getter, setters and Computer class default and parameterized constructor definition.

```
//Including header files
#include <iostream>
#include <Windows.h>
#include <ctime>
#include "Computer.h"
#include "sqlite/sqlite3.h"

using namespace std;

Computer::Computer(void) {}//Default Constructor

Computer::Computer(time_t _start_time = NULL, time_t _end_time = NULL) : start_time(_start_time), end_time(_end_time)

void Computer::login_comp(time_t _start_time) { start_time = _start_time; } //Function used to set start time for the computer

void Computer::logout_comp(time_t _end_time, int* accessTime) { //Function used to find total session time for the computer
    end_time = _end_time;
    *accessTime = difftime(end_time, start_time);
    start_time = NULL;
    end_time = NULL;
}

//Getter and setter methods for start and ending time
void Computer::setStartTime(time_t _start_time) { start_time = _start_time; }
void Computer::setEndTime(time_t _end_time) { end_time = _end_time; }
time_t Computer::getStartTime(void) { return start_time; }
time_t Computer::getEndTime(void) { return end_time; }


```

The screenshot shows two identical instances of Microsoft Visual Studio running side-by-side. Both instances have the following configuration:

- Title Bar:** CyberCafeManagementSystemMain
- File Menu:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help
- Search Bar:** Search (Ctrl+Q)
- Toolbars:** Standard, Status
- Code Editor:** The file Computer.cpp is open, showing C++ code related to a Computer class. The code includes methods for setting and getting start and end times, and a callback function for SQLite database queries.
- Solution Explorer:** Shows the project structure with files like Computer.h, Customer.h, Customer.cpp, User.h, Owner.h, CyberCafeManagementSystemMain.cpp, and pymath.h.
- Properties:** Properties pane is visible on the right.
- Status Bar:** Ready, Add to Source Control, Select Repository, Ln: 34, Ch: 1, TABS, CRLF

The code in Computer.cpp is as follows:

```
25 //Getter and setter methods for start and ending time
26 void Computer::setStartTime(time_t _start_time) { start_time = _start_time; }
27 void Computer::setEndTime(time_t _end_time) { end_time = _end_time; }
28 time_t Computer::getStartTime(void) { return start_time; }
29 time_t Computer::getEndTime(void) { return end_time; }
30
31
32 //Method to add a computer into the database
33
34 //callback function is called whenever sqlite3_exec() returns results (records) from the SQLite database
35 int Computer::callback(void* data, int count, char** rows, char** azColName) {
36     //data->is the database object, count is number of rows, rows is entries in rows, azColName is column name in table
37
38     int i;
39
40     for (i = 0; i < count; i++) {
41         printf("%s = %s\n", azColName[i], rows[i] ? rows[i] : "NULL");
42     }
43     printf("\n");
44
45     return 0;
46 }
47
48 //callback to return count
49 int Computer::callBackLogin(void* data, int count, char** rows, char**)
50 {
51     if (count == 1 && rows) {
52         *static_cast<int*>(data) = atoi(rows[0]);
53     }
54     return 1;
55 }
```

```
C:\Users\Acer\source\repos\CyberCafeManagementSystemMain\x64\Release\CyberCafeManagementSystemMain.exe
1. Register New Owner
2. Owner Login

1.
Enter the rate for today
2.
Owner Object Created
Opened database successfully
OWNER REGISTRATION
Enter first name
Hrushikesh
Enter last name
Ahire
Enter email
hrushikeshahire@gmail.com
Enter mobile
9899999999
Enter Username
Hrushi123
Enter Password
1234

Registration successfully
Successfully registered
Opened database successfully
OWNER LOGIN
Enter Username
Hrushi123
Enter Password
1234

Query Executed successfully
Login Successful
```

Message

Owner Logged In Successfully

OK

## Static and Dynamic libraries

### 1) Static Libraries :

- A static library (or archive) contains code that is linked to users' programs at compile time.
- The executable file generated keeps its own copy of the library code.

### 2) Dynamic libraries :

- A dynamic library consists of routines that are loaded into your application at run time.
- When you compile a program that uses a dynamic library, the library does not become part of your executable , it remains as a separate unit.

### 3) Creating Static libraries and dynamic libraries :

- a) Run the given command on the command line to generate .obj files.

```
cl.exe /c /EHsc file1.cpp file2.cpp file3.cpp
```

```
D:\CCMS\TEST_ccms\TEST_ccms\Server>cl.exe /c /EHsc CyberCafeManagementServer.cpp User.cpp Owner.cpp Customer.cpp Computer.cpp sqlite3.c shell.c
Microsoft (R) C/C++ Optimizing Compiler Version 19.30.30706 for x64
Copyright (C) Microsoft Corporation. All rights reserved.

CyberCafeManagementServer.cpp
User.cpp
Owner.cpp
Customer.cpp
Customer.cpp(161): warning C4311: '<function-style-cast>': pointer truncation from 'char *' to 'int'
Computer.cpp
Generating Code...
Compiling...
sqlite3.c
shell.c
Generating Code...
```

- b) The /c command-line option instructs the compiler to compile and create .obj (object files).
- c) The /EHsc command-line option instructs the compiler to enable standard C++ exception handling behaviour and synchronization.
- d) Now we have to link the .obj files to create an .exe file or .dll files.

```
link.exe /DLL /DEF:file1.def file1.obj file2.obj user32.lib
```

- e) Module definition file (.def) is a file which shows the list of functions which are library is exporting.

## Cmake and make

### 1) Cmake :

- CMake is an open-source, cross-platform family of tools designed to build, test and package software.
- CMake is used to control the software compilation process using simple platform and compiler independent configuration files, and generate native makefiles.

## Steps to build Project using Cmake and make

- 1) First Create a CMakeLists.txt file in the project folder.

```
M CMakeLists.txt
1 cmake_minimum_required (VERSION 3.10)
2 set(CMAKE_BUILD_TYPE Debug)
3 set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++14")
4
5 project (CCMS)
6 add_executable(CyberCafeManagement CyberCafeManagementSystemMain.cpp)
7
8 include_directories(CyberCafeManagementSystemMain sqlite3)
```

- 2) The above code is a configuration file Code which tells the Cmake tool to build the project in Debug mode or to set the executable file name etc.
- 3) Run "cmake ." command to build makefile.

```
PS D:\CCMS\CCMS_NEW> cmake .
-- Building for: Visual Studio 17 2022
-- Selecting Windows SDK version 10.0.19041.0 to target Windows 10.0.19043.
-- The C compiler identification is MSVC 19.30.30706.0
-- The CXX compiler identification is MSVC 19.30.30706.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: c:/Program Files/Microsoft Visual Studio/2022/Community/VC/Tools/MSVC/14.30.30705/bin/Hostx64/x64/cl.exe - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: c:/Program Files/Microsoft Visual Studio/2022/Community/VC/Tools/MSVC/14.30.30705/bin/Hostx64/x64/cl.exe - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: D:/CCMS/CCMS_NEW
PS D:\CCMS\CCMS_NEW>
```

- 4) This will create a makeFile.

```

M Makefile
1 ##### Makefile Template #####
2 ##### Makefile Template #####
3 #####
4
5 # Compiler settings - Can be customized.
6 CC = g++
7 CXXFLAGS = -std=c++11 -Wall
8 LDFLAGS =
9
10 # Makefile settings - Can be customized.
11 APPNAME = CCMS
12 EXT = .cpp
13 SRCDIR = .
14 OBJDIR = .
15
16 ##### Do not change anything from here downwards! #####
17 SRC = $(wildcard $(SRCDIR)/*$(EXT))
18 OBJ = $(SRC:$(SRCDIR)/%$(EXT)=$(OBJDIR)/%.o)
19 DEP = $(OBJ:$(OBJDIR)/%.o=%.d)
20 # UNIX-based OS variables & settings
21 RM = rm
22 DELOBJ = $(OBJ)
23 # Windows OS variables & settings
24 DEL = del
25 EXE = .exe
26 WDELOBJ = $(SRC:$(SRCDIR)/%$(EXT)=$(OBJDIR)\\%.o)
27
28 #####
29 ##### Targets beginning here #####
30 #####
31
32 all: $(APPNAME)
33
34 # Builds the app
35 $(APPNAME): $(OBJ)
36     $(CC) $(CXXFLAGS) -o $@ $^ $(LDFLAGS)
37

```

- 5) Run the "make" command to Actually build the files of the project and .exe to Run the project.

## Explicitly Creating MakeFile

## (Different Approach)

- 1) First Step : Creating the MakeFile to Create the sqlite3.obj file.  
We have the sqlite3 files in the directory.

```
M Makefile
1 $(warning Starting Makefile)
2
3 CXX=g++
4 CXXFLAGS = -std=c++17 -Wall
5
6
7
8 sqlite3.obj: sqlite3.c
9     $(CXX) -c sqlite3.c -o sqlite3.obj -DTHREADSAFE=1
10
11
12
13
14 clean:
15     -rm *.obj
```

- 2) After writing this code in the makeFile we run "make" command.

```
PS D:\CCMS\TEST_ccms\TEST_ccms> make
makefile:1: Starting Makefile
make: `sqlite3.obj' is up to date.
PS D:\CCMS\TEST_ccms\TEST_ccms> []
```

- 3) We create the project .obj files using same method.

```

M Makefile
1  $(warning Starting Makefile)
2
3  CXX=g++
4  CXXFLAGS = -std=c++17 -Wall
5
6  main.exe: sqlite3.obj Owner.obj Computer.obj Customer.obj User.obj Helper.obj
7      $(warning main bit not done)
8
9
10
11 sqlite3.obj: sqlite3.c
12     $(CXX) -c sqlite3.c -o sqlite3.obj -DTHREADSAFE=1
13
14
15
16
17 User.obj: User.cpp User.h sqlite3.h
18     $(CXX) -c User.cpp -o User.obj
19
20 Owner.obj: Owner.cpp Owner.h Computer.h Customer.h sqlite3.h
21     $(CXX) -c Owner.cpp -o Owner.obj
22
23 Computer.obj: Computer.cpp Computer.h sqlite3.h
24     $(CXX) -c Computer.cpp -o Computer.obj
25
26 Customer.obj: Customer.cpp Customer.h sqlite3.h
27     $(CXX) -c Customer.cpp -o Customer.obj
28
29 Helper.obj: Helper.cpp Helper.h sqlite3.h
30     $(CXX) -c Customer.cpp -o Customer.obj
31
32 clean:
33     -rm *.obj
34     -rm *.exe

```

The diagram shows a callout box labeled "Code To Create obj files." with three arrows pointing to the definitions of User.obj, Computer.obj, and Customer.obj respectively.

- 4) Run the make Command again.

```

PS D:\CCMS\TEST_ccms\TEST_ccms> make
makefile:1: Starting Makefile
make: Circular Computer.obj <- Computer.obj dependency dropped.
g++ -c Customer.cpp -o Customer.obj
makefile:7: main bit not done
PS D:\CCMS\TEST_ccms\TEST_ccms>

```

- 5) Creating .lib file by using all the .obj files.

```

11
12
13 sqlite_demo.lib: sqlite3.obj Owner.obj Computer.obj Customer.obj User.obj Helper.obj
14     ar cq $@ Owner.obj
15     ar cq $@ Computer.obj
16     ar cq $@ Customer.obj
17     ar cq $@ User.obj
18     ar cq $@ Helper.obj
19     ar cq $@ sqlite3.obj]
20
21 sqlite3.obj: sqlite3.c
22     $(CXX) -c sqlite3.c -o sqlite3.obj -DTHREADSAFE=1
23
24
25 #CyberCafeManagementSystemMain.obj: CyberCafeManagementSystemMain.cpp
26 #    sqlite3.h Owner.h Computer.h Customer.h User.h Helper.h
27 #    $(CXX) -c CyberCafeManagementSystemMain.cpp -o CyberCafeManagementSystemMain.obj
28
29 User.obj: User.cpp User.h sqlite3.h
30     $(CXX) -c User.cpp -o User.obj
31
32 Owner.obj: Owner.cpp Owner.h Computer.h Customer.h sqlite3.h
33     $(CXX) -c Owner.cpp -o Owner.obj
34
35 Computer.obj: Computer.obj Computer.h sqlite3.h
36     $(CXX) -c Computer.cpp -o Computer.obj
37

```

- 6) Run “make” command again.

```

PS D:\CCMS\TEST_ccms\TEST_ccms> make
makefile:1: Starting Makefile
make: Circular Computer.obj <- Computer.obj dependency dropped.
g++ -c Customer.cpp -o Customer.obj
ar cq sqlite_demo.lib Owner.obj
ar cq sqlite_demo.lib Computer.obj
ar cq sqlite_demo.lib Customer.obj
ar cq sqlite_demo.lib User.obj
ar cq sqlite_demo.lib Helper.obj
ar cq sqlite_demo.lib sqlite3.obj

```

- 7) Last step to create .exe file for our project.

```

M MakeFile
1  $(warning Starting Makefile)
2
3  CXX=g++
4  CXXFLAGS = -std=c++17 -Wall
5
6  #main.exe: sqlite3.obj Owner.obj Computer.obj Customer.obj User.obj Helper.obj
7  #  $(warning main bit not done)
8
9  TEST_ccms.exe: sqlite_demo.lib TEST_ccms.cpp
10   $(CXX) -s TEST_ccms.cpp -o TEST_ccms.exe -wl,sqlite_demo.lib
11
12
13  sqlite_demo.lib: sqlite3.obj Owner.obj Computer.obj Customer.obj User.obj Helper.obj
14    ar cq $@ Owner.obj
15    ar cq $@ Computer.obj
16    ar cq $@ Customer.obj
17    ar cq $@ User.obj
18    ar cq $@ Helper.obj
19    ar cq $@ sqlite3.obj
20
21  sqlite3.obj: sqlite3.c
22   $(CXX) -c sqlite3.c -o sqlite3.obj -DTHREADSAFE=1
23
24
25  #CyberCafeManagementSystemMain.obj: CyberCafeManagementSystemMain.cpp
26  #      sqlite3.h Owner.h Computer.h Customer.h User.h Helper.h
27  #  $(CXX) -c CyberCafeManagementSystemMain.cpp -o CyberCafeManagementSystemMain.obj
28
29  User.obj: User.cpp User.h sqlite3.h
30   $(CXX) -c User.cpp -o User.obj
31
32  Owner.obj: Owner.cpp Owner.h Computer.h Customer.h sqlite3.h
33   $(CXX) -c Owner.cpp -o Owner.obj
34
35  Computer.obj: Computer.cpp Computer.h sqlite3.h
36   $(CXX) -c Computer.cpp -o Computer.obj
37

```

For .exe file

- 8) Run “make” to get the .exe file for our project.

```

PS D:\CCMS\TEST_ccms\TEST_ccms> make
makefile:1: Starting Makefile
make: Circular Computer.obj <- Computer.obj dependency dropped.
g++ -c Customer.cpp -o Customer.obj
g++ -s TEST_ccms.cpp -o TEST_ccms.exe -wl,sqlite_demo.lib

```

## Multi-Threading

## What is Multithreading?

Multithreading is a specialized form of multitasking and multitasking is the feature that allows your computer to run two or more programs concurrently. In general, there are two types of multitasking: process-based and thread-based.

Process-based multitasking handles the concurrent execution of programs. Thread-based multitasking deals with the concurrent execution of pieces of the same program.

A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution.

1. Here first we creating a thread which is in suspended state.

```
int main(int argc, char *argv[], char *envp[])
{
    HANDLE hConsole;

    // variable declarations
    int ch_01, ch_1, ch_2, ch_3;
    char ch = 'Y';

    //For registration
    static HANDLE hThread1 = NULL;
    hThread1 = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)registerThread, NULL, CREATE_SUSPENDED, NULL);
}
```

2.1. Now resuming the suspended thread when the owner object needs to be created.

2.2. In order to synchronize the working of the hThread1 (the one which we created) and the main thread of our program, we wait for execution of hThread1 and after its completion main thread resume its execution.

2.3. After the execution of thread, we close the thread using CloseHandle()

```
        switch (ch_1)
    {
        case 1:
            // register owner
            //_createOwner();
            ResumeThread(hThread1);
            WaitForSingleObject(hThread1, INFINITE);
            CloseHandle(hThread1);
```

3. registerThread() is the function which is executed by our hThread1 thread.

```
5
6
7    DWORD WINAPI registerThread(LPVOID param)
8    {
9        _createOwner();
10       return 0;
11    }
12
```

## Milestone 4

### Assembly Code

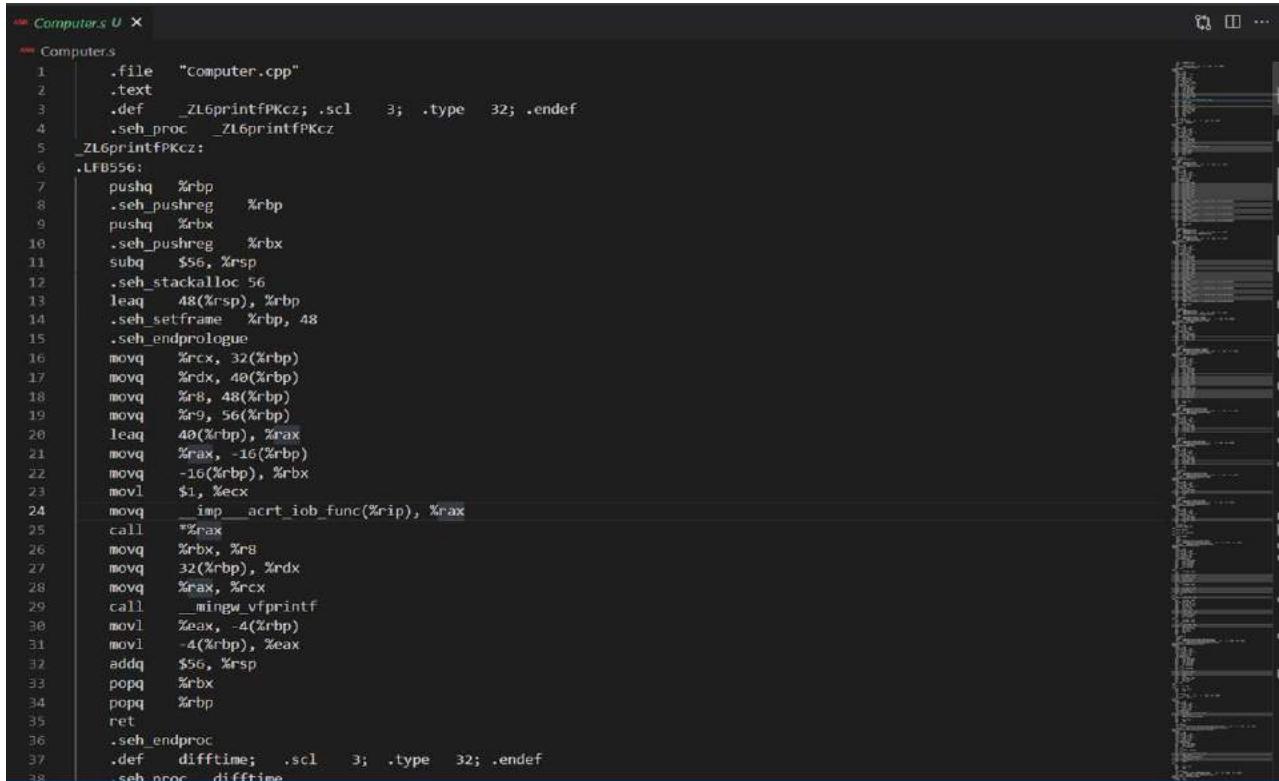
An assembly language is a low-level programming language designed for a specific type of processor.

It may be produced by compiling source code from a high-level programming language (such as C/C++) but can also be written from scratch.

Assembly code can be converted to machine code using an assembler.

- Commands

g++ -S filename.cpp



```
Computer.s X
Computers
1 .file "computer.cpp"
2 .text
3 .def _ZL6printfPKcz; .scl 3; .type 32; .edef
4 .seh_proc _ZL6printfPKcz
5 _ZL6printfPKcz:
6 .LFB556:
7     pushq %rbp
8     .seh_pushreg %rbp
9     pushq %rbx
10    .seh_pushreg %rbx
11    subq $56, %rsp
12    .seh_stackalloc 56
13    leaq 48(%rsp), %rbp
14    .seh_setframe %rbp, 48
15    .seh_endprologue
16    movq %rcx, 32(%rbp)
17    movq %rdx, 40(%rbp)
18    movq %r8, 48(%rbp)
19    movq %r9, 56(%rbp)
20    leaq 40(%rbp), %rax
21    movq %rax, -16(%rbp)
22    movq -16(%rbp), %rbx
23    movl $1, %ecx
24    movq _imp__acrt_iob_func(%rip), %rax
25    call *%rax
26    movq %rbx, %rb
27    movq 32(%rbp), %rdx
28    movq %rax, %rcx
29    call _mingw_vfprintf
30    movl %eax, -4(%rbp)
31    movl -4(%rbp), %eax
32    addq $56, %rsp
33    popq %rbx
34    popq %rbp
35    ret
36    .seh_endproc
37    .def difftime; .scl 3; .type 32; .edef
38    .seh_proc difftime
```

Each indented line in the above code corresponds to a single machine instruction. For example, the **pushq** instruction indicates that the contents of register **%rbp** should be pushed onto the program stack.

## Description

While assembly languages differ between processor architectures, they often include similar instructions and operators. Below are some examples of instructions :

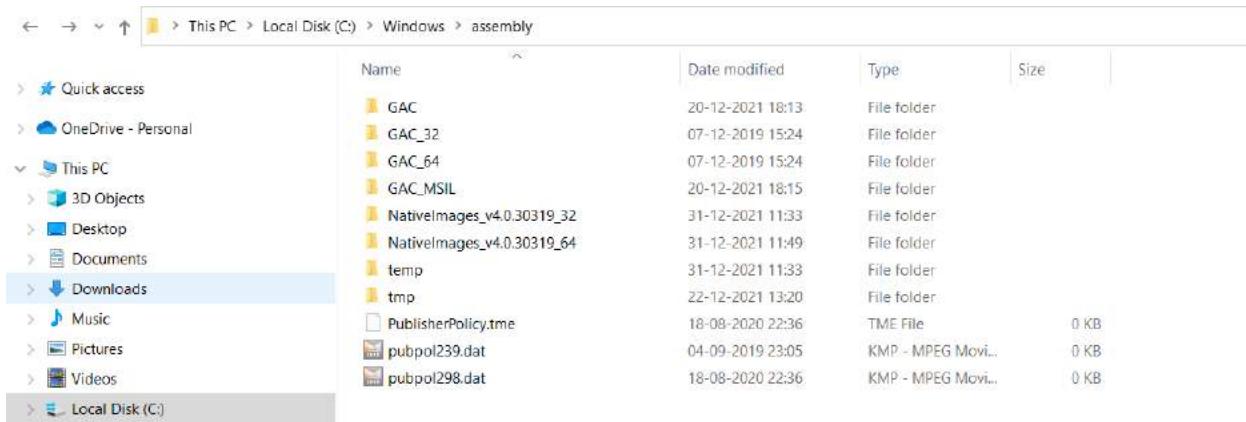
- **MOV** - move data from one location to another
- **ADD** - add two values

- SUB - subtract a value from another value
- PUSH - push data onto a stack
- POP - pop data from a stack
- JMP - jump to another location
- INT - interrupt a process

## Global Assembly Code

GAC is a shared location in your machine that keeps assemblies, DLLs for common sharing purposes. Assume that you are using *system.data.dll* in your project. So one option is you can place the DLL with your project. But assume you are creating another project which is also using the same .NET assembly *system.data.dll*, so it is better to have it in a common place where that can be referred.

Since it is a common framework DLL and used separately from your application logic, it is good to place in a shared place for ease of maintenance and update process.

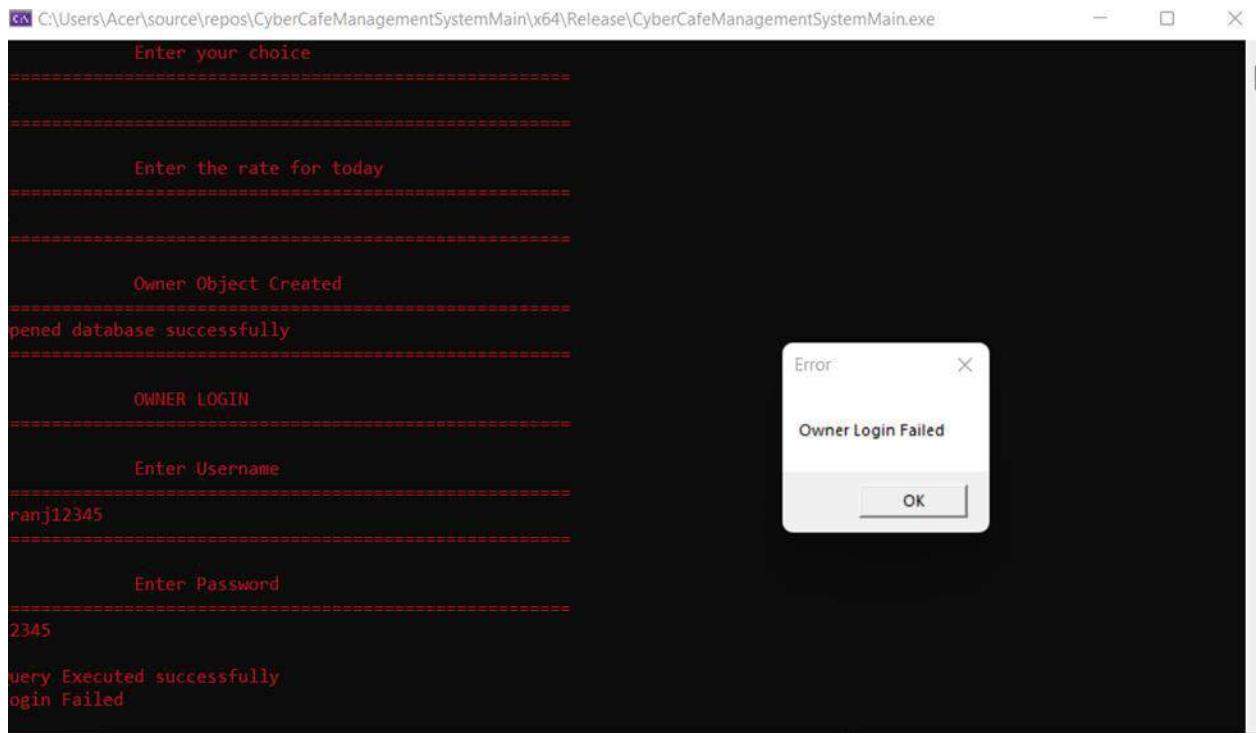


The screenshot shows a Windows File Explorer window with the following path: This PC > Local Disk (C:) > Windows > assembly. The 'Downloads' folder is selected in the left sidebar. The right pane displays a list of folders and files:

Name	Date modified	Type	Size
GAC	20-12-2021 18:13	File folder	
GAC_32	07-12-2019 15:24	File folder	
GAC_64	07-12-2019 15:24	File folder	
GAC_MSIL	20-12-2021 18:15	File folder	
NativeImages_v4.0.30319_32	31-12-2021 11:33	File folder	
NativeImages_v4.0.30319_64	31-12-2021 11:49	File folder	
temp	31-12-2021 11:33	File folder	
tmp	22-12-2021 13:20	File folder	
PublisherPolicy.tme	18-08-2020 22:36	TME File	0 KB
pubpol239.dat	04-09-2019 23:05	KMP - MPEG Movi...	0 KB
pubpol298.dat	18-08-2020 22:36	KMP - MPEG Movi...	0 KB

## Debugging(Windows)

- 1) Found a bug : Here first we did a false Owner login, It said Login Failed.



- 2) Then After login failed it goes to the home Page. If After trying to login again. It says login failed without even asking the username and password.

```
C:\Users\Acer\source\repos\CyberCafeManagementSystemMain\x64\Release\CyberCafeManagementSystemMain.exe

Enter Password
=====
12345

Query Executed successfully
Login Failed
=====
Welcome to Cyber Cafe
=====
Login as:-->
=====
1.Owner
2.Customer
3.Exit
=====

Enter your choice
=====
1
=====
1. Register New Owner
2. Owner Login
3. Go Back
=====

Enter your choice
=====
2
```

Error X  
Owner Login Failed  
OK

3) Starting to Debug: Breakpoint at main.

```
14
15 int main(int argc, char* argv[], char* envp[]) {
16     HANDLE hConsole;
```

4) Started Debugging: Started watching gpOwner.

CyberCafeManagementSystemMain

```

27 cout << "=====";
28 cout << "\n      Welcome to Cyber Cafe " << endl;
29 cout << "=====";
30 cout << "\n      Login as:-->" << endl;
31 cout << "=====";
32 cout << "\n          1.Owner\n          2.Customer\n";
33 cout << "=====";
34 cout << "\n      Enter your choice" << endl;
35 cout << "=====";
36 cin >> ch_01;
37 switch (ch_01) {
38     case 1:
39         label:
40             hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
41             SetConsoleTextAttribute(hConsole, 4);
42             cout << "=====";
43             std::cout << "\n          1. Register New Owner\n";
44             cout << "=====";
45             cout << "\n      Enter your choice" << endl;
46             cout << "=====";
47             std::cin >> ch_1;

```

Welcome to Cyber Cafe  
Login as:-->  
1.Owner  
2.Customer  
3.Exit  
Enter your choice  
1  
1. Register New Owner  
2. Owner Login  
3. Go Back  
Enter your choice

Watch 1

Name	Value	Type
h	0x0000000000000000 -NULL-	Owner *

## 5) Entered 1 to login as Owner.

CyberCafeManagementSystemMain

```

38 case 1:
39 label:
40     hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
41     SetConsoleTextAttribute(hConsole, 4);
42     cout << "=====";
43     std::cout << "\n          1. Register New Owner\n";
44     cout << "=====";
45     cout << "\n      Enter your choice" << endl;
46     cout << "=====";
47     std::cin >> ch_1;
48     switch (ch_1) {
49         case 1:
50             // register owner
51             _createOwner();
52
53         case 2:
54             // login owner
55             if (_LoginOwner()) {
56                 MessageBox(NULL, TEXT("Owner Logged in Success"));
57             }
58             do {
59                 hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
60                 SetConsoleTextAttribute(hConsole, 4);
61                 cout << "=====";
62                 std::cout << "\n          1. Register New Owner\n";
63                 cout << "=====";
64                 cout << "\n          2. Owner Login\n";
65                 cout << "=====";
66                 cout << "\n          3. Go Back\n";
67                 cout << "=====";
68                 cout << "\nEnter the rate for today\n";
69                 cout << "=====";
70                 std::cin >> rate_per_min;
71                 cout << "=====";
72                 cout << "\n      Enter your choice" << endl;
73                 cout << "=====";
74                 std::cin >> choice;
75             } while (choice != 3);
76
77             if (choice == 1) {
78                 _createOwner();
79             }
80             else if (choice == 2) {
81                 _LoginOwner();
82             }
83             else {
84                 cout << "=====";
85                 cout << "\n      Enter your choice" << endl;
86                 cout << "=====";
87             }
88         }
89     }
90 }

```

Enter your choice  
1  
1. Register New Owner  
2. Owner Login  
3. Go Back  
Enter your choice  
Enter the rate for today  
20  
Owner Object Created.  
Open database successfully  
OWNER LOGIN  
Enter Username

Watch 1

Name	Value	Type
h	0x0000000000000000 -NULL-	Owner *

## 6) Entered 2 to Owner Login and entered the rate\_per\_min.

```
cout << "=====\n";
cout << "\n      Do you want to continue(Y/y)" << endl;
cout << "=====\n";
cin >> ch;
} while (ch == 'Y' || ch == 'y');

else {
    MessageBox(NULL, TEXT("Owner Login Failed"), TEXT("Error"), MB_OK);
}
```

Enter your choice  
y  
Enter the rate for today  
10  
Owner Object Created  
Opened database successfully  
OWNER LOGIN  
Enter Username  
pran  
Enter Password  
1234  
Query Executed successfully  
Login Failed

## 7) Entered Wrong Credentials For First Login.

```
cout << "=====\n";
cout << "\n      Do you want to continue(Y/y)" << endl;
cout << "=====\n";
cin >> ch;
} while (ch == 'Y' || ch == 'y');

else {
    MessageBox(NULL, TEXT("Owner Login Failed"), TEXT("Error"), MB_OK);
}
```

Error  
Owner Login Failed  
OK

## 8) Login Failed.

```
cout << "===== Welcome to Cyber Cafe =====" << endl;
cout << "===== Enter Username =====" << endl;
cout << "===== Enter Password =====" << endl;
cout << "===== Enter your choice" << endl;
cout << "===== 1. Owner\n 2. Customer\n =====" << endl;
cin >> ch_01;
switch (ch_01) {
    case 1:
        label:
            hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
            SetConsoleTextAttribute(hConsole, 4);
            cout << "===== 1. Register New Owner\n";
            cout << "===== Enter your choice" << endl;
            cout << "===== 1. Register New Owner\n";
            cout << "===== Enter your choice" << endl;
            cout << "===== 1. Register New Owner\n";
            cout << "===== Enter your choice" << endl;
```

The screenshot shows the application's main menu. The user has entered '1' for 'Owner'. The program then asks for a password ('Enter Password') and a choice ('Enter your choice'). The user has entered '1234'. The program then asks for a choice again ('Enter your choice').

## 9) Went to homepage.

```
case 1:
label:
    hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    SetConsoleTextAttribute(hConsole, 4);
    cout << "===== 1. Register New Owner\n";
    cout << "===== Enter your choice" << endl;
    cout << "===== 1. Register New Owner\n";
    cout << "===== Enter your choice" << endl;
    cout << "===== 1. Register New Owner\n";
    cout << "===== Enter your choice" << endl;
    std::cin >> ch_1;
    switch (ch_1) {
```

The screenshot shows the application's main menu. The user has entered '1' for 'Owner'. The program then asks for a choice ('Enter your choice'). The user has entered '1'. The program then asks for a choice again ('Enter your choice'). The user has entered '1'. The program then asks for a choice again ('Enter your choice').

## 10) Entered 1 for Login as Owner. And entered 2 for Owner Login.

The screenshot shows a Microsoft Visual Studio IDE during a debugging session. The code editor displays a portion of the `CyberCafeManagementSystemMain.cpp` file. A tooltip 'Owner Login Failed' is shown above a line of code. A message box titled 'Owner Login Failed' is open, prompting the user to 'Enter your choice'. The message box has three options: '1. Register New Owner', '2. Owner Login', and '3. Go Back'. In the background, a terminal window shows a login process with messages like 'Enter Password', 'Login Failed', and 'Welcome to Cyber Cafe'.

11) Without asking the username and password it says Login Failed and the gpOwner values still doesn't change. It Remains Null.

The screenshot shows the 'Watch 1' window in Microsoft Visual Studio. It contains two entries: 'gpOwner' and 'rate\_per\_minute'. The 'gpOwner' entry is of type 'Owner\*' and has a value of '0x0000000000000000 <NULL>'. The 'rate\_per\_minute' entry is of type 'int' and has a value of '2'. There is also a note 'Add item to watch'.

12) Tried Watching rate\_per\_minuite as well.

```
cout << "======" << endl;
cout << "\n      Enter the rate for today" << endl;
cin >> rate_per_minute;
Owner* obj = new Owner(rate_per_minute);
if (obj->login()) {
    gpOwner = obj;
    return true;
}
else {
    return false;
}
```

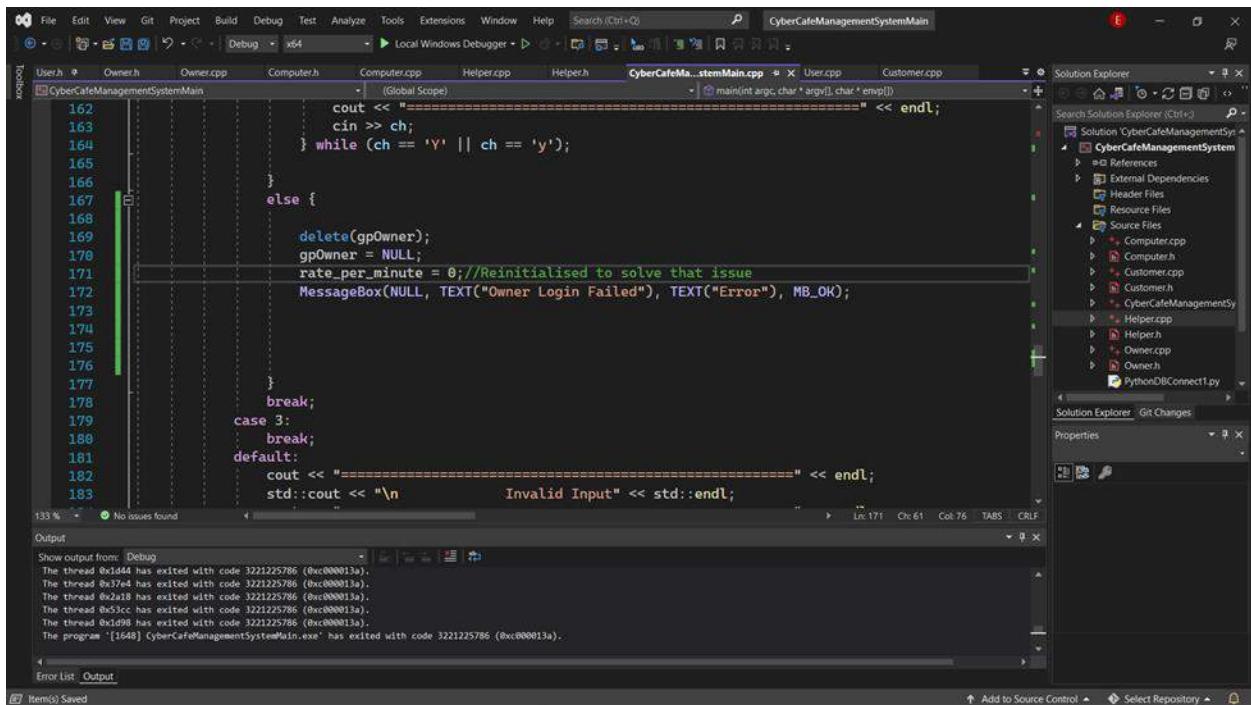
The thread 0x1d44 has exited with code 3221225786 (0xc000013a).
The thread 0x37e4 has exited with code 3221225786 (0xc000013a).
The thread 0x2a18 has exited with code 3221225786 (0xc000013a).
The thread 0x53cc has exited with code 3221225786 (0xc000013a).
The thread 0xd98 has exited with code 3221225786 (0xc000013a).
The program '[1648] CyberCafeManagementSystemMain.exe' has exited with code 3221225786 (0xc000013a).

13) First Time Rate\_per\_minuite was initialized to 2. But when login failed it was not reinitialized to 0. Which was a important condition for owner Login.

```
if (rate_per_minute == 0) //This was causing issue for owner login failed when trying again
{
    cout << "======" << endl;
    cout << "\n      Enter the rate for today" << endl;
    cout << "======" << endl;
    cin >> rate_per_minute;
    Owner* obj = new Owner(rate_per_minute);
    if (obj->login()) {
        gpOwner = obj;
        return true;
    }
    else {
        return false;
    }
}
```

The thread 0x1d44 has exited with code 3221225786 (0xc000013a).
The thread 0x37e4 has exited with code 3221225786 (0xc000013a).
The thread 0x2a18 has exited with code 3221225786 (0xc000013a).
The thread 0x53cc has exited with code 3221225786 (0xc000013a).
The thread 0xd98 has exited with code 3221225786 (0xc000013a).
The program '[1648] CyberCafeManagementSystemMain.exe' has exited with code 3221225786 (0xc000013a).

#### 14) Reinitialized rate\_per\_minitue to 0 when login failed.



```
162 cout << "======" << endl;
163 cin >> ch;
164 } while (ch == 'Y' || ch == 'y');
165
166 else {
167     delete(gpOwner);
168     gpOwner = NULL;
169     rate_per_minute = 0; //Reinitialised to solve that issue
170     MessageBox(NULL, TEXT("Owner Login Failed"), TEXT("Error"), MB_OK);
171
172     break;
173 case 3:
174     break;
175 default:
176     cout << "======" << endl;
177     std::cout << "\n      Invalid Input" << std::endl;
178 }
```

## Open-Source Debugging

The purpose of debugging is to handle the error regarding "login". During execution, when the Owner is logging in with wrong credentials the login fails and after we try to login again with correct credentials, the login still fails.

1. Used the command "g++ \*.cpp -g -l sqlite3" to initialize the compiler for debugging.

```
spark@spark-virtual-machine:~/Cyber1/CCMS$ g++ *.cpp -g -l sqlite3
```

2. The g++ compiler returns an output file named "a.out" which is used with gdb to start the debugging process and we enter into the debugger.

```
spark@spark-virtual-machine: ~/Cyber1/CCMS
spark@spark-virtual-machine:~/Cyber1/CCMS$ gdb a.out
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from a.out...
(gdb) 
```

3. Now we use "break" command to set breakpoint at line number 58.

```
(gdb) break 58
Breakpoint 1 at 0x4fc2: file CyberCafeManagementSystemMain.cpp, line 58.
(gdb) 
```

4. Now we use the "run" command to start the execution.

```
(gdb) run
Starting program: /home/spark/Cyber1/CCMS/a.out
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
=====
        Welcome to Cyber Cafe
=====
        Login as:-->
=====
        1.Owner
        2.Customer
        3.Exit
=====
        Enter your choice
=====

Enter your choice
=====
1
=====
    1. Register New Owner
    2. Owner Login
    3. Go Back
=====

        Enter your choice
=====
2
Breakpoint 1, main (argc=1, argv=0x7fffffff3e8, envp=0x7fffffff3f8) at CyberCafeManagementSystemMain.cpp:58
warning: Source file is more recent than executable.
58                      if (_loginOwner()) {
(gdb) 
```

## 5. We use the command "s" to step over.

```
(gdb) s
_loginOwner () at Helper.cpp:33
33      bool _loginOwner(void) {
(gdb) s
34          if (rate_per_minute == 0) {//This was causing issue for owner login failed when trying again
(gdb) s
36              cout << "====="
(gdb) s
37              cout << "\n"           Enter the rate for today" << endl;
(gdb) s
38              cout << "====="
(gdb) s
39              cin >> rate_per_minute;
10
41          Owner* obj = new Owner(rate_per_minute);
(gdb) 
```

## 6. Now we use "c" to continue the debugging flow till the next execution which leads to the creation of the Owner object.

```
(gdb) s
Owner::Owner (
    this=0x7fffff7b3b2d4 <__GI__libc_malloc+116>,
_rate_per_min=0) at Owner.cpp:10
10      Owner::Owner(unsigned int _rate_per_min =
1) : rate_per_min(_rate_per_min) {
(gdb) s
User::User (this=0xfffffffffffffff90) at User.h:13
13      class User {
(gdb) s
Computer::Computer (this=0x55555557a6d0)
    at Computer.cpp:9
9       Computer::Computer(void) {}//Default Const
ructor
(gdb) s
Computer::Computer (this=0x55555557a840)
    at Computer.cpp:9
9       Computer::Computer(void) {}//Default Const
ructor
(gdb) s
Computer::Computer (this=0x55555557a8f0)
    at Computer.cpp:9
9       Computer::Computer(void) {}//Default Const
ructor
(gdb) c
Continuing.
=====
=====
          Owner Object Created
=====
```

7. We use a wrong username and password to execute where login fails.

```
spark@spark-virtual-machine: ~/Cyber1/CCMS
```

```
        Enter Username  
=====  
=====  
jyuo  
=====  
  
        Enter Password  
=====  
lklo  
=====  
Query Executed successfully  
Login Failed  
=====  
        Welcome to Cyber Cafe  
=====  
=====  
Login as:-->  
=====  
1.Owner  
2.Customer  
3.Exit  
=====  
  
        Enter your choice  
=====  
1  
=====  
1. Register New Owner  
2. Owner Login  
3. Go Back  
=====  
  
        Enter your choice  
=====
```

8. Now we face the error where the execution fails without asking for username and password, the login fails.

```

        Enter your choice
=====
2

Breakpoint 1, main (argc=1, argv=0x7fffffff3e8,
    envp=0x7fffffff3f8)
    at CyberCafeManagementSystemMain.cpp:58
58             if (_loginOwner()) {
(gdb) s
_loginOwner () at Helper.cpp:33
33     bool _loginOwner(void) {
(gdb) s
34         if (rate_per_minute == 0) { //This was causing issue for owner login failed when trying again
(gdb) s
52     }
(gdb) c
Continuing.

=====Owner's Menu=====
1.Manage Computer
2.Manage Customer
3.Update Profile
4.Get daily charges
5.Logout
6.Go back

        Enter your choice
=====
1
=====
```

9. The program receives a segmentation fault and on continuing it terminates with the fault.

```

spark@spark-virtual-machine: ~/Cyber1/CCMS
2.Manage Customer
3.Update Profile
4.Get daily charges
5.Logout
6.Go back

        Enter your choice
=====
1

1.Add New Computer
2.Edit Computer
3.Delete Computer
4.Search Computer
5.Show Computer Status
6.Go back

        Enter your choice
=====

4

Program received signal SIGSEGV, Segmentation fault.
0x00001f2f1f1f5861 in ?? ()
    from /lib/x86_64-linux-gnu/libsqlite3.so.0
(gdb) c
Continuing.

Program terminated with signal SIGSEGV, Segmentation fault.
The program no longer exists.
(gdb) █
```

10. Now we use the "quit" command to return to the directory.

```
the program no longer exists.  
(gdb) quit  
spark@spark-virtual-machine:~/Cyber1/CCMS$
```

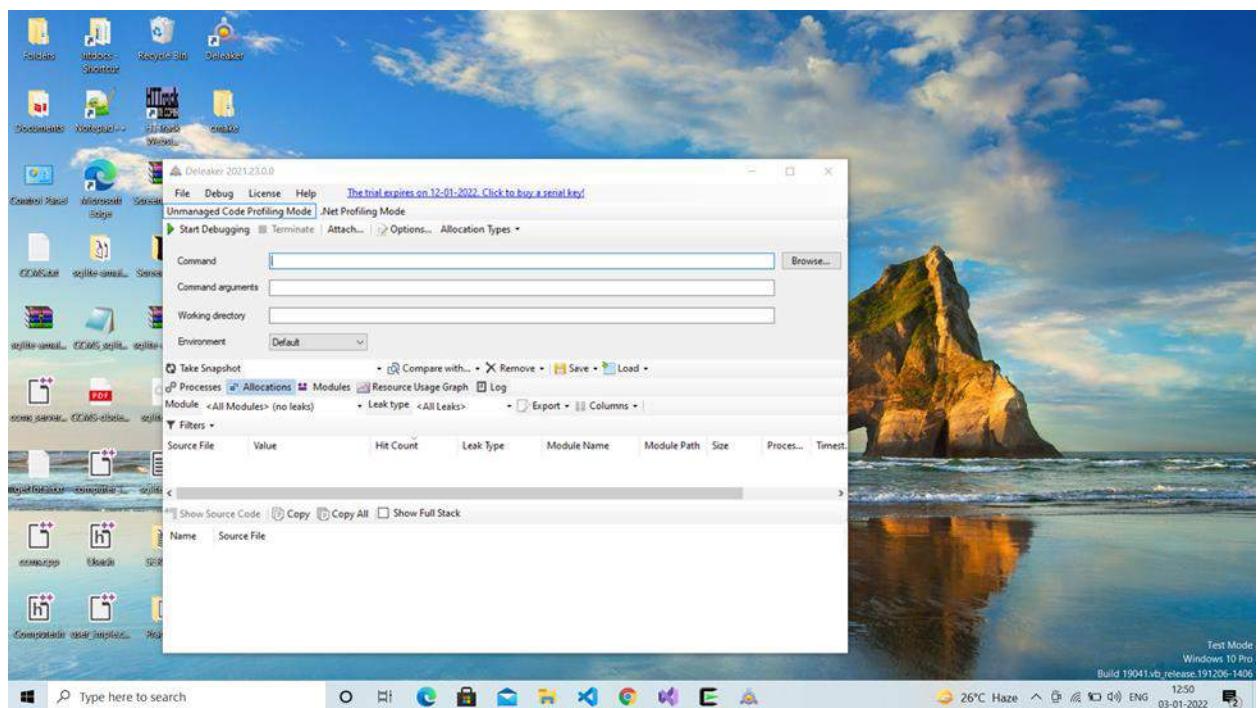
11. First time the rate\_per\_minute was initialized to 2, but when login failed it was not reinitialized to 0. So, we set gpOwner to NULL and reinitialize the rate\_per\_minute to 0.

```
170  
171     else {  
172         delete(gpOwner);  
173         gpOwner = NULL;  
174         rate_per_minute = 0;//Reinitialised to solve that issue
```

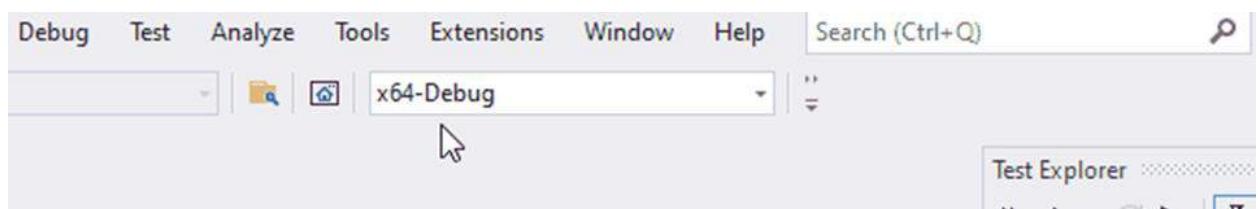
## Memory Leak (Windows)

**Using “Deleaker” windows Software for Memory Leak Detection.**

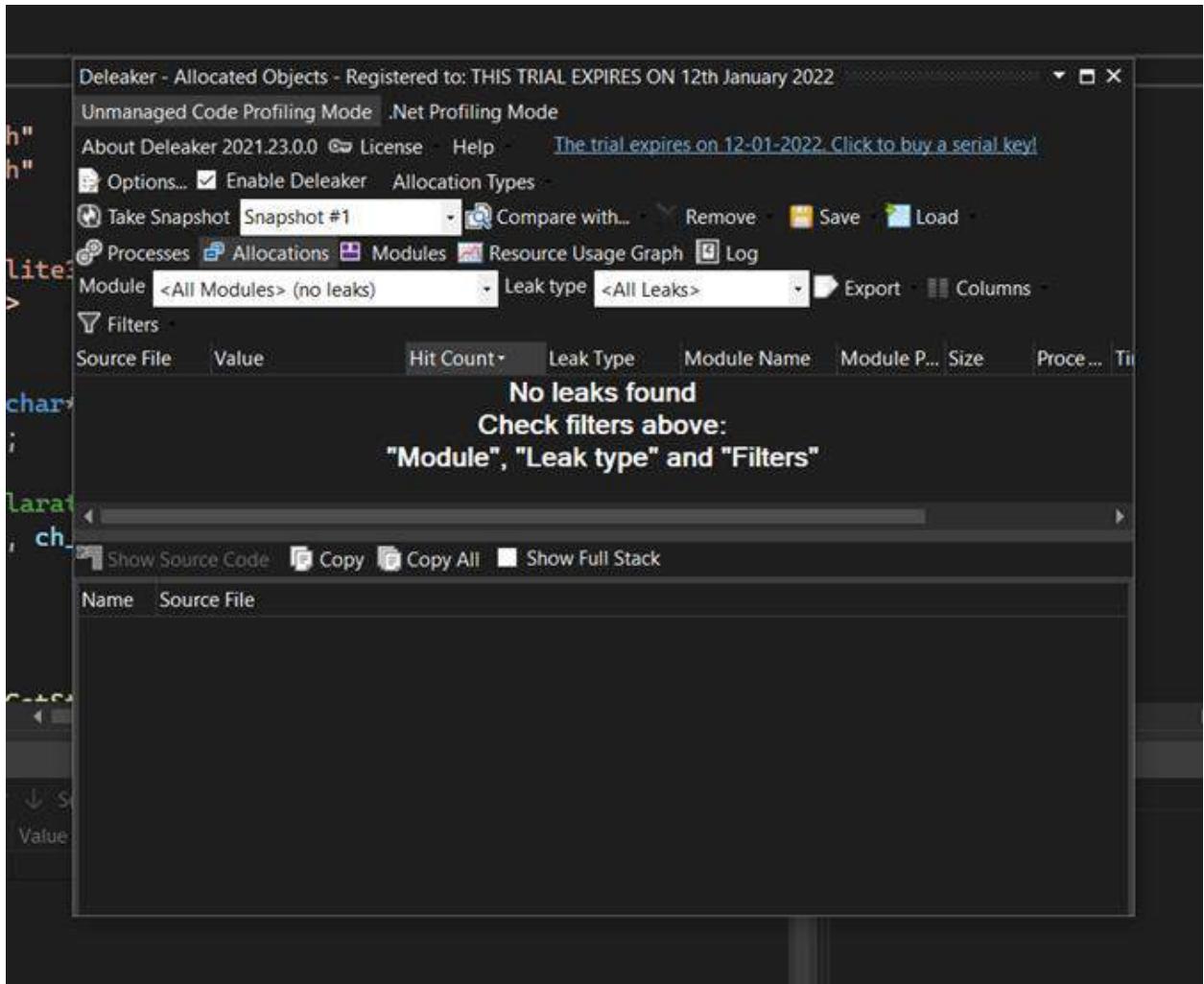
1) Using Deleaker. Download the free trial which can be used for 14 days.



- 2) Once downloaded it will register itself automatically with the Visual studio.
- 3) Open Visual Studio. Go to Menu->Extensions->Deleaker Window.



4) This window will appear.



- 5) Select the Unmanaged code Profiling Method. And enable Deleaker if not enabled.
- 6) Start the visual studio Debugger and the it will start allocating object to detect Memory leak.

Deleaker - Allocated Objects - Registered to: THIS TRIAL EXPIRES ON 12th January 2022

Unmanaged Code Profiling Mode .Net Profiling Mode

About Deleaker 2021.23.0.0 License Help The trial expires on 12-01-2022. Click to buy a serial key!

Options...  Enable Deleaker Allocation Types

Take Snapshot Snapshot #1  Compare with... Remove  Save  Load

Processes  Allocations  Modules  Resource Usage Graph  Log

Module <All Modules> (4 leaks) Leak type <All Leaks> Export Columns

Filters 67 hidden 4 of 71 shown

Source File	Value	Hit Count	Leak Type	Module Name	Module P...	Size	Proce...	Times...	Seque...	Thread Id
vcstartup_int...	0x000001e4ce1cf550	1	Heap memo...	MemoryLeakC...	D:\CCMS...	120	20192	03-01...	177	1332
MemoryLeak...	0x000001e4ce1ed5...	1	Heap memo...	MemoryLeakC...	D:\CCMS...	4148	20192	03-01...	188	1332
ostream, line ...	0x000001e4ce1ec4...	1	Heap memo...	MemoryLeakC...	D:\CCMS...	4148	20192	03-01...	187	1332
utility.cpp, lin...	0x000001e4ce1c73...	1	Heap memo...	MemoryLeakC...	D:\CCMS...	308	20192	03-01...	176	1332

Show Source Code  Copy  Copy All  Show Full Stack

Name	Source File
ntdll.d...	
ucrtba...	
Memo...	d:\a01\work\20\src\vtctools\crt\vcstartup\inc\vcstartup_internath.h, line 418
Memo...	d:\a01\work\20\src\vtctools\crt\vcstartup\src\startup\exe_common.inl, line 167
ucrtba...	
Memo...	d:\a01\work\20\src\vtctools\crt\vcstartup\inc\startup\exe_common.inl, line 253
Memo...	d:\a01\work\20\src\vtctools\crt\vcstartup\src\startup\exe_common.inl, line 330
Memo...	d:\a01\work\20\src\vtctools\crt\vcstartup\src\startup\exe_main.cpp, line 16
KERN...	
ntdll.d...	

## Memory Leak detection

(Open source)

## 2) Memory Leak Detection (Open Source) :

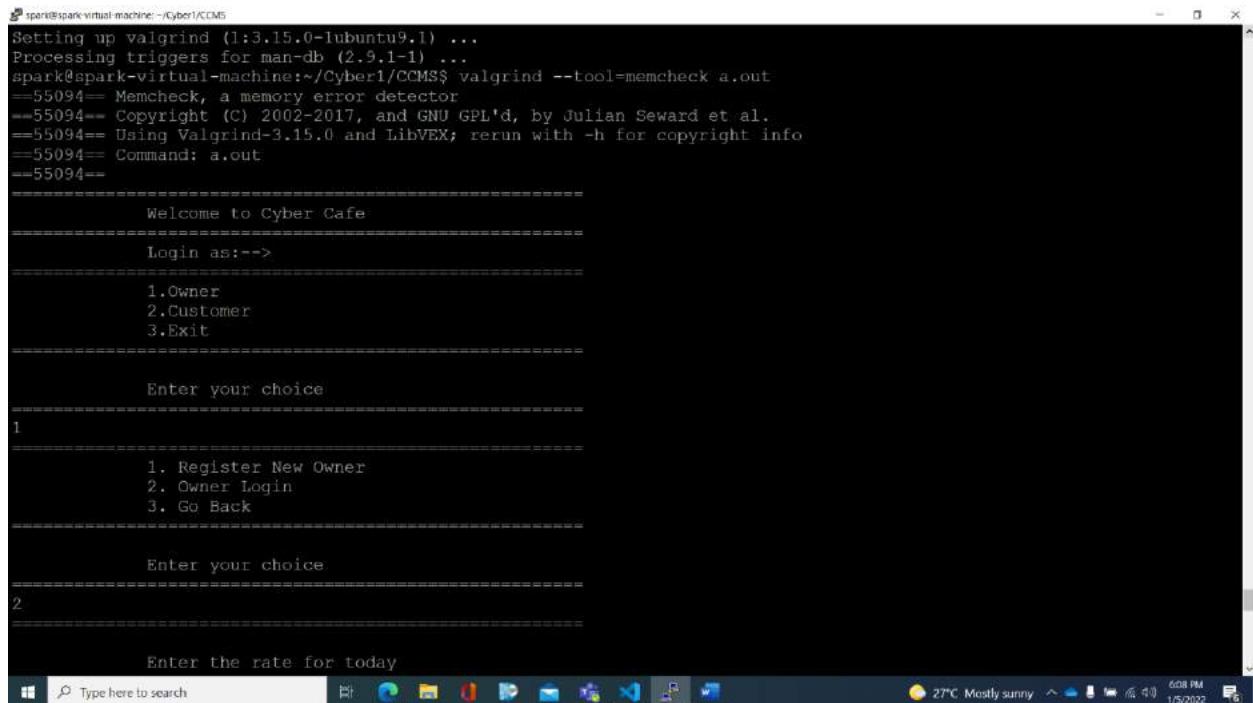
- Step 1 : Install “valgrind”.

Sudo apt install valgrind

- Step 2 :

Compile the code. And run the following command.

valgrind --tool = memcheck a.out



The screenshot shows a terminal window on a Windows desktop. The terminal output is as follows:

```
spark@spark-virtual-machine:~/Cyber1/CCMS
Setting up valgrind (1:3.15.0-1ubuntu9.1) ...
Processing triggers for man-db (2.9.1-1) ...
spark@spark-virtual-machine:~/Cyber1/CCMS$ valgrind --tool=memcheck a.out
==55094== Memcheck, a memory error detector
==55094== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==55094== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==55094== Command: a.out
==55094==

=====
        Welcome to Cyber Cafe
=====
        Login as:-->
=====
        1.Owner
        2.Customer
        3.Exit
=====
Enter your choice
=====
1
=====
    1. Register New Owner
    2. Owner Login
    3. Go Back
=====
Enter your choice
=====
2
=====
    Enter the rate for today
=====
```

The terminal shows the Valgrind tool running on a Linux system, followed by the execution of the 'a.out' program. The program displays a menu for 'Cyber Cafe' with options for 'Owner', 'Customer', and 'Exit'. The user selects 'Owner' and then 'Owner Login'. Finally, the user enters '2' at the prompt 'Enter the rate for today'.

- Step 3 : At the exit of the code, we get the output of the memory leaked.

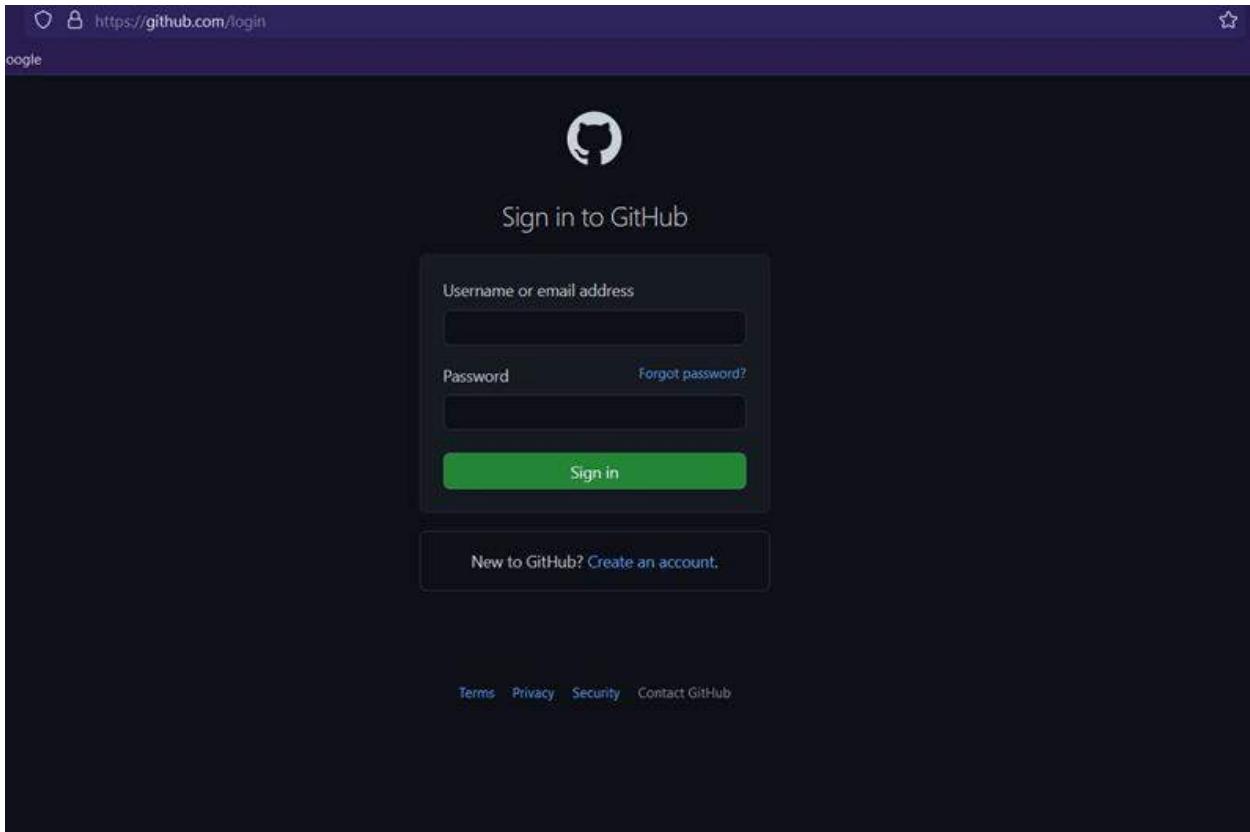
```
spark@spark-virtual-machine:~/Cyber1/CCMS$  
3  
=====  
Welcome to Cyber Cafe  
=====  
Login as:-->  
1.Owner  
2.Customer  
3.Exit  
=====  
Enter your choice  
=====  
3  
==55094==  
==55094== HEAP SUMMARY:  
==55094==     in use at exit: 48,503 bytes in 229 blocks  
==55094== total heap usage: 395 allocs, 166 frees, 160,390 bytes allocated  
==55094==  
==55094== LEAK SUMMARY:  
==55094==   definitely lost: 0 bytes in 0 blocks  
==55094==   indirectly lost: 0 bytes in 0 blocks  
==55094==   possibly lost: 1,032 bytes in 1 blocks  
==55094==   still reachable: 47,471 bytes in 228 blocks  
==55094==           of which reachable via heuristic:  
==55094==                 length64      : 45,232 bytes in 226 blocks  
==55094==   suppressed: 0 bytes in 0 blocks  
==55094== Rerun with --leak-check=full to see details of leaked memory  
==55094==  
==55094== For lists of detected and suppressed errors, rerun with: -s  
==55094== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)  
spark@spark-virtual-machine:~/Cyber1/CCMS$
```



## Milestone 5

### Version Control (GitHub)

- 1) Login into github : visit [github.com](https://github.com/login) to login or signup.



## 2) Creating Repository : Create a git repository

The screenshot shows the GitHub interface for creating a new repository. At the top, there is a navigation bar with icons for Pull requests, Issues, Marketplace, and Explore. Below this, the main heading is "Create a new repository". A sub-instruction says, "A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository." The "Owner" field is set to "KadirSheikh". The "Repository name" field contains "Management\_System(Team\_Sharda)" with a green checkmark. A tooltip for the repository name field states, "Great repository names are descriptive, unique, and easy to remember. Your new repository will be created as Cyber\_Cafe\_Management\_System-Team\_Sharda.". There is an optional "Description" field which is currently empty. Below these fields are two radio button options: "Public" (selected) and "Private". The "Public" option is described as "Anyone on the internet can see this repository. You choose who can commit." The "Private" option is described as "You choose who can see and commit to this repository." At the bottom, there is a section titled "Initialize this repository with:" with a note "Skip this step if you're importing an existing repository."

## 3) Inviting Teammates to collaborate :



4) Initialize Git On Local Machine : Use \$ git init Command.

```
MINGW64:/d/Cyber_Cafe
kadir_sheikh@NGL001300 MINGW64 /d/cyber_cafe
$ git init
Initialized empty Git repository in D:/cyber_Cafe/.git/
kadir_sheikh@NGL001300 MINGW64 /d/cyber_cafe (master)
$
```

## 5) Use Git commands :

```
MINGW64:/d/Cyber_Cafe
kadir_sheikh@NGL001300 MINGW64 /d/cyber_cafe
$ git init
Initialized empty Git repository in D:/Cyber_Cafe/.git/
kadir_sheikh@NGL001300 MINGW64 /d/Cyber_Cafe (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    Documents/

nothing added to commit but untracked files present (use "git add" to track)
kadir_sheikh@NGL001300 MINGW64 /d/Cyber_Cafe (master)
$ |
```

\$ git status command to see untracked files.

```
MINGW64:/d/Cyber_Cafe
kadir_sheikh@NGL001300 MINGW64 /d/Cyber_Cafe
$ git init
Initialized empty Git repository in D:/Cyber_Cafe/.git/
kadir_sheikh@NGL001300 MINGW64 /d/Cyber_Cafe (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    Documents/

nothing added to commit but untracked files present (use "git add" to track)
kadir_sheikh@NGL001300 MINGW64 /d/Cyber_Cafe (master)
$ git add .

kadir_sheikh@NGL001300 MINGW64 /d/Cyber_Cafe (master)
$
```

\$ git add to track files.

```
MINGW64:/d/Cyber_Cafe
kadir_sheikh@NGL001300 MINGW64 /d/Cyber_Cafe (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    Documents/

nothing added to commit but untracked files present (use "git add" to track)

kadir_sheikh@NGL001300 MINGW64 /d/cyber_Cafe (master)
$ git add .

kadir_sheikh@NGL001300 MINGW64 /d/cyber_Cafe (master)
$ git commit -m "Documents"
[master (root-commit) 71e107b] Documents
Committer: Kadir Sheikh <kadir_sheikh@persistent.com>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

  git config --global --edit

After doing this, you may fix the identity used for this commit with:

  git commit --amend --reset-author

13 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 Documents/Customer_Flow_Diagram/Customer_Flow_Chart.jpg
create mode 100644 Documents/Customer_Flow_Diagram/Customer_Flow_Chart.pdf
create mode 100644 Documents/Customer_Flow_Diagram/Customer_Flow_Chart.png
create mode 100644 "Documents/Cyber Caf\303\251 management System.pptx"
create mode 100644 Documents/Cyber_Cafe_Class_Diagram.jpg
create mode 100644 Documents/Cyber_Cafe_Customer_Sequence.png
create mode 100644 Documents/Cyber_Cafe_Management_Project_Synopsis.docx
create mode 100644 Documents/Cyber_Cafe_Owner_Sequence.png
create mode 100644 Documents/Cyber_Cafe_Use_Case.jpg
create mode 100644 Documents/ER Diagram_Cyber Cafe Management_1-01.png
create mode 100644 Documents/Owner_Flow_Diagram/Owner_Flow_Chart.jpg
create mode 100644 Documents/Owner_Flow_Diagram/Owner_Flow_Chart.pdf
create mode 100644 Documents/Owner_Flow_Diagram/Owner_Flow_Chart.png

kadir_sheikh@NGL001300 MINGW64 /d/cyber_Cafe (master)
$ |
```

\$ git commit to commit files on remote repository.

## 6) File Structure of Project on GitHub :

The screenshot shows a GitHub repository page. At the top, there's a search bar and navigation links for Pull requests, Issues, Marketplace, and Explore. Below the header, the repository name 'KadirSheikh/Cyber\_Cafe\_Management\_System-Team\_Sharda-' is displayed, along with a 'Private' link. To the right are buttons for Unwatch (1), Fork, Star (0), and Settings.

The main content area shows a file tree under the 'Code' tab. It includes a master branch, one branch, and zero tags. A commit by 'ENCHALWARPRANJAL' titled 'Update README.md' is shown, made 30 minutes ago. Another commit by 'Documents' titled 'Update README.md' is shown, made 44 minutes ago. A file named 'README.md' is also listed.

On the right side, there's an 'About' section with a note: 'No description, website, or topics provided.' It lists 'Readme', '0 stars', '1 watching', and '0 forks'. Below that is a 'Releases' section with a note: 'No releases published' and a 'Create a new release' button. Finally, there's a 'Packages' section.

## Checking untracked files using "git status" command

The terminal window shows the following output of the 'git status' command:

```
kadir_sheikh@NGL001300 MINGW64 /d/cyber_cafe (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

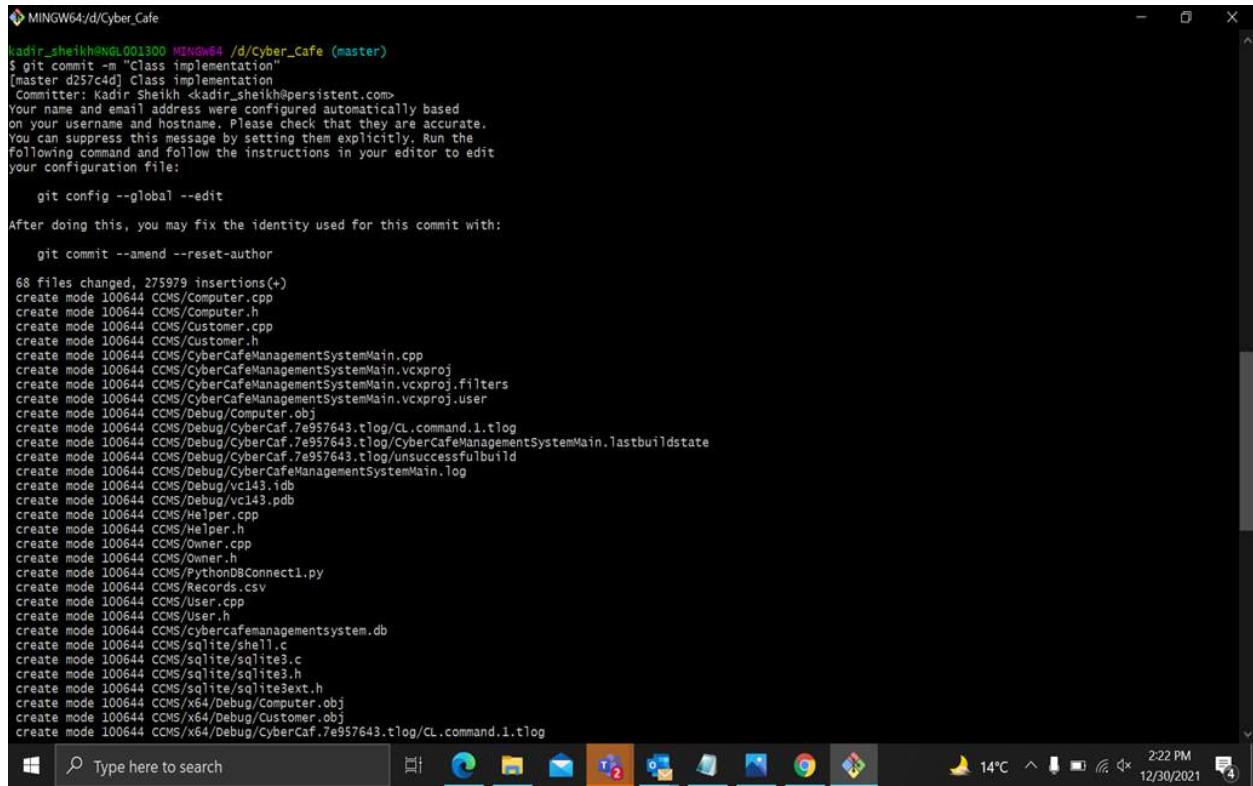
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    CCMS/
nothing added to commit but untracked files present (use "git add" to track)

kadir_sheikh@NGL001300 MINGW64 /d/cyber_cafe (master)
$ git commit -m "Class implementation"
[CCMS: master]
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    CCMS/
nothing added to commit but untracked files present (use "git add" to track)

kadir_sheikh@NGL001300 MINGW64 /d/cyber_cafe (master)
$ git add .
warning: LF will be replaced by CRLF in CCMS/Computer.cpp.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in CCMS/Computer.h.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in CCMS/Customer.cpp.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in CCMS/Customer.h.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in CCMS/CyberCafeManagementSystemMain.cpp.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in CCMS/CyberCafeManagementSystemMain.vcxproj.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in CCMS/CyberCafeManagementSystemMain.vcxproj.filters.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in CCMS/CyberCafeManagementSystemMain.vcxproj.user.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in CCMS/Debug/Cybercaf_7e9576431.log/CyberCafeManagementSystemMain.lastbuildstate.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in CCMS/Debug/CyberCafeManagementSystemMain.Tog.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in CCMS/Helper.cpp.
The file will have its original line endings in your working directory
```

## Adding "git add." to track all the untracked files



```
MINGW64/d/Cyber_Cafe
kadir_sheikh@MGL001300 MINGW64 /d/cyber_Cafe (master)
$ git commit -m "Class implementation"
[master d257c4d] Class implementation
 1 file changed, 275979 insertions(+)
 create mode 100644 CCMS/Computer.cpp
 create mode 100644 CCMS/Computer.h
 create mode 100644 CCMS/Customer.cpp
 create mode 100644 CCMS/Customer.h
 create mode 100644 CCMS/CyberCafeManagementSystemMain.cpp
 create mode 100644 CCMS/CyberCafeManagementSystemMain.vcxproj
 create mode 100644 CCMS/CyberCafeManagementSystemMain.vcxproj.filters
 create mode 100644 CCMS/CyberCafeManagementSystemMain.vcxproj.user
 create mode 100644 CCMS/Debug/Computer.obj
 create mode 100644 CCMS/Debug/CyberCaf.7e957643.tlog/CL.command.1.tlog
 create mode 100644 CCMS/Debug/CyberCaf.7e957643.tlog/CyberCafeManagementSystemMain.lastbuildstate
 create mode 100644 CCMS/Debug/CyberCaf.7e957643.tlog/unsuccessfulbuild
 create mode 100644 CCMS/Debug/CyberCafeManagementSystemMain.log
 create mode 100644 CCMS/Debug/vc143.idb
 create mode 100644 CCMS/Debug/vc143.pdb
 create mode 100644 CCMS/Helper.cpp
 create mode 100644 CCMS/Helper.h
 create mode 100644 CCMS/Owner.cpp
 create mode 100644 CCMS/Owner.h
 create mode 100644 CCMS/PythonDBConnect1.py
 create mode 100644 CCMS/Records.csv
 create mode 100644 CCMS/User.cpp
 create mode 100644 CCMS/User.h
 create mode 100644 CCMS/cybercafemanagementsystem.db
 create mode 100644 CCMS/sqlite/shell.c
 create mode 100644 CCMS/sqlite/sqlite3.c
 create mode 100644 CCMS/sqlite/sqlite3.h
 create mode 100644 CCMS/sqlite/sqlite3ext.h
 create mode 100644 CCMS/x64/Debug/Computer.obj
 create mode 100644 CCMS/x64/Debug/Customer.obj
 create mode 100644 CCMS/x64/Debug/CyberCaf.7e957643.tlog/CL.command.1.tlog
```

git commit -m "message" to commit files on remote repository.

```
MINGW64/d/Cyber_Cafe
create mode 100644 CCMS/x64/Debug/CyberCaf.7e957643.tlog/link.read.1.tlog
create mode 100644 CCMS/x64/Debug/CyberCaf.7e957643.tlog/link.write.1.tlog
create mode 100644 CCMS/x64/Debug/cyberCafeManagementSystemMain.exe.recipe
create mode 100644 CCMS/x64/Debug/cyberCafeManagementSystemMain.lib
create mode 100644 CCMS/x64/Debug/cyberCafeManagementSystemMain.log
create mode 100644 CCMS/x64/Debug/cyberCafeManagementSystemMain.obj
create mode 100644 CCMS/x64/Debug/Helper.obj
create mode 100644 CCMS/x64/Debug/Owner.obj
create mode 100644 CCMS/x64/Debug/User.obj
create mode 100644 CCMS/x64/Debug/shell.obj
create mode 100644 CCMS/x64/Debug/sqlite3.obj
create mode 100644 CCMS/x64/Debug/vc143.idb
create mode 100644 CCMS/x64/Debug/vc143.pdb
create mode 100644 CCMS/x64/Release/Computer.obj
create mode 100644 CCMS/x64/Release/Customer.obj
create mode 100644 CCMS/x64/Release/CyberCaf.7e957643.tlog/CL.command.1.tlog
create mode 100644 CCMS/x64/Release/CyberCaf.7e957643.tlog/CL.read.1.tlog
create mode 100644 CCMS/x64/Release/CyberCaf.7e957643.tlog/CL.write.1.tlog
create mode 100644 CCMS/x64/Release/CyberCaf.7e957643.tlog/CyberCafeManagementSystemMain.lastbuildstate
create mode 100644 CCMS/x64/Release/CyberCaf.7e957643.tlog/Link.command.1.tlog
create mode 100644 CCMS/x64/Release/CyberCaf.7e957643.tlog/Link.read.1.tlog
create mode 100644 CCMS/x64/Release/CyberCaf.7e957643.tlog/Link.write.1.tlog
create mode 100644 CCMS/x64/Release/CyberCafeManagementSystemMain.exe.recipe
create mode 100644 CCMS/x64/Release/CyberCafeManagementSystemMain.lib
create mode 100644 CCMS/x64/Release/CyberCafeManagementSystemMain.log
create mode 100644 CCMS/x64/Release/CyberCafeManagementSystemMain.libp
create mode 100644 CCMS/x64/Release/CyberCafeManagementSystemMain.log
create mode 100644 CCMS/x64/Release/Helper.obj
create mode 100644 CCMS/x64/Release/Owner.obj
create mode 100644 CCMS/x64/Release/User.obj
create mode 100644 CCMS/x64/Release/shell.obj
create mode 100644 CCMS/x64/Release/sqlite3.obj
create mode 100644 CCMS/x64/Release/vc143.pdb
create mode 100644 CCMS/x64/Release/vc143.pdb

kadir_sheikh@NGL001300 MINGW64 /d/Cyber_Cafe (master)
$ git push -u origin master
Enumerating objects: 80, done.
Counting objects: 100% (80/80), done.
Delta compression using up to 8 threads
Compressing objects: 100% (76/76), done.
Writing objects: 100% (79/79), 11.89 MiB | 110.00 KiB/s, done.
Total 79 (delta 23), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (23/23), completed with 1 local object.
To https://github.com/KadirSheikh/Cyber_Cafe_Management_System-Team_Sharda-.git
 d441d9c..d257c4d master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
kadir_sheikh@NGL001300 MINGW64 /d/Cyber_Cafe (master)
$
```

## Future Scope

1. Monthly income of cyber cafe.
2. Security on data can be implemented. As well as it can be converted into GUI platform
3. Customer can view his/her usage history.