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

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

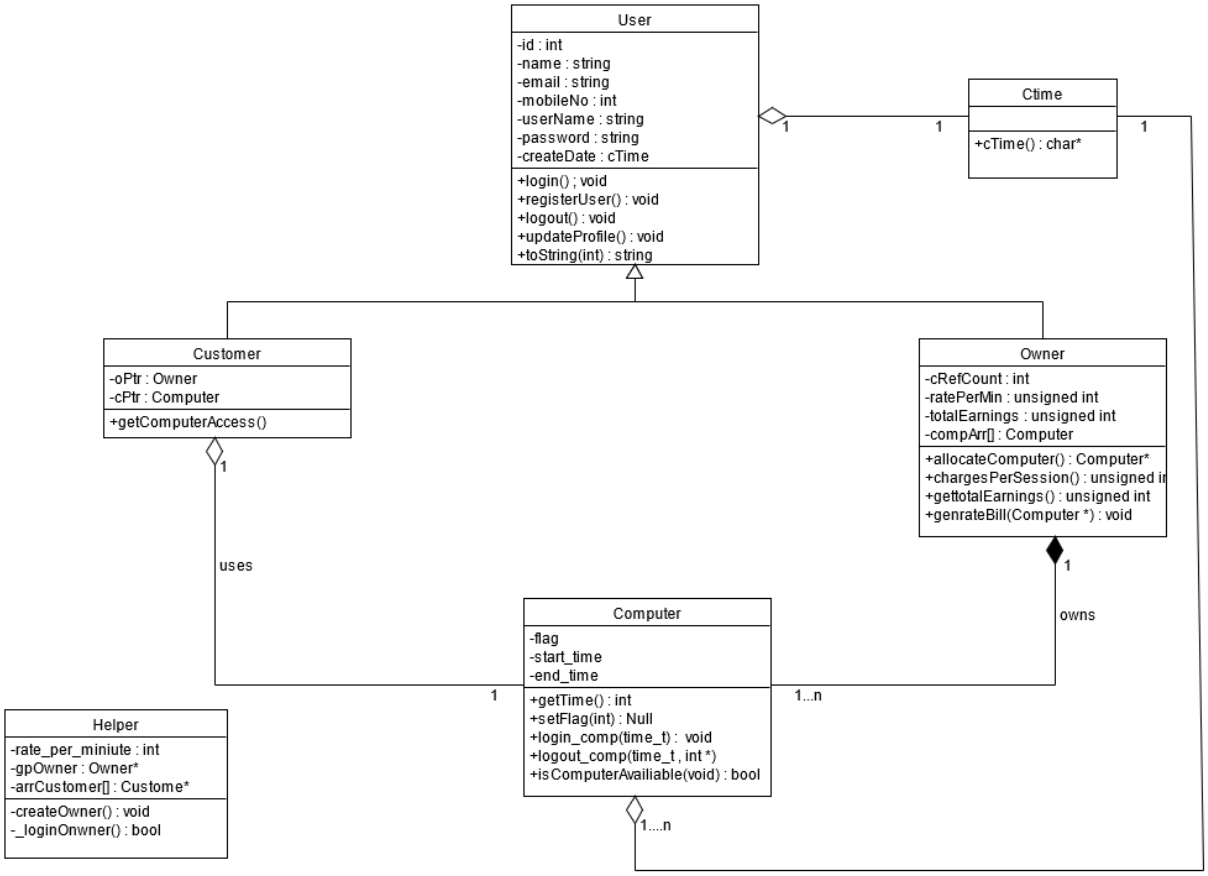
3) Hrushikesh Ahire

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.

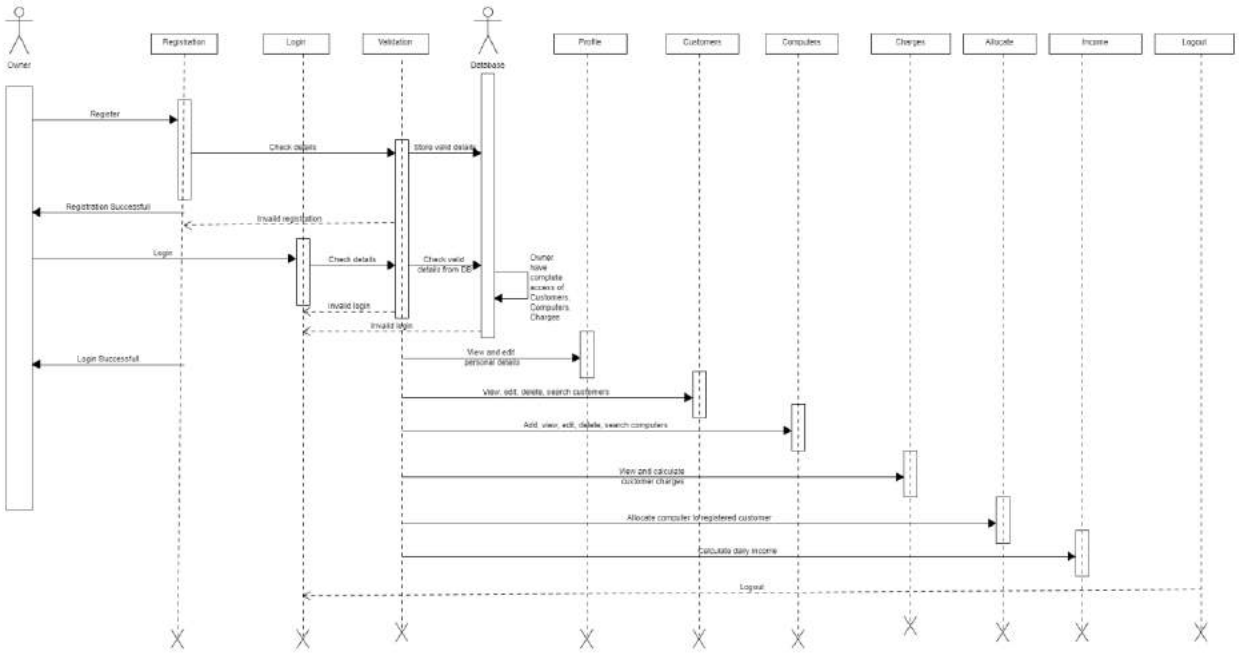
The relationship between Customer and Computer class is Aggregation. It's a “part of” kind of relationship.

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

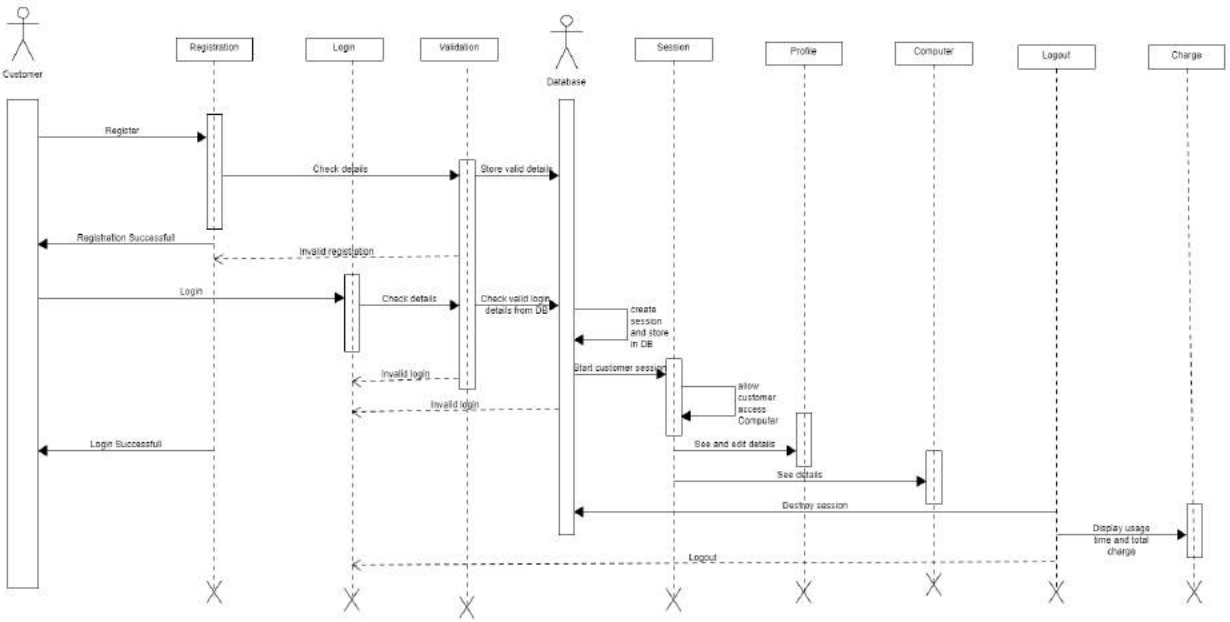
Sequence Diagrams

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

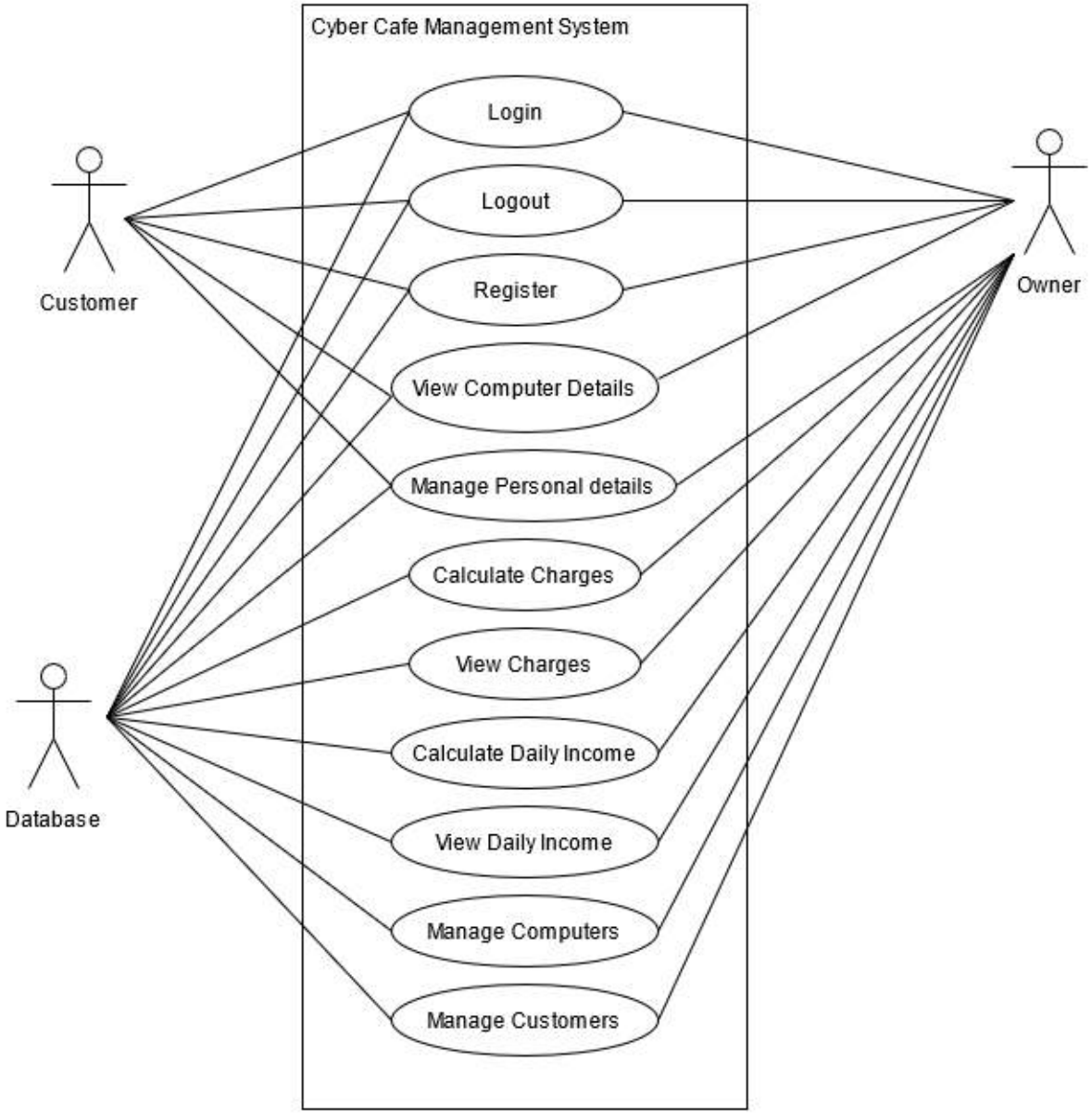


2. Customer Sequence Diagram



Use Case Diagram

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



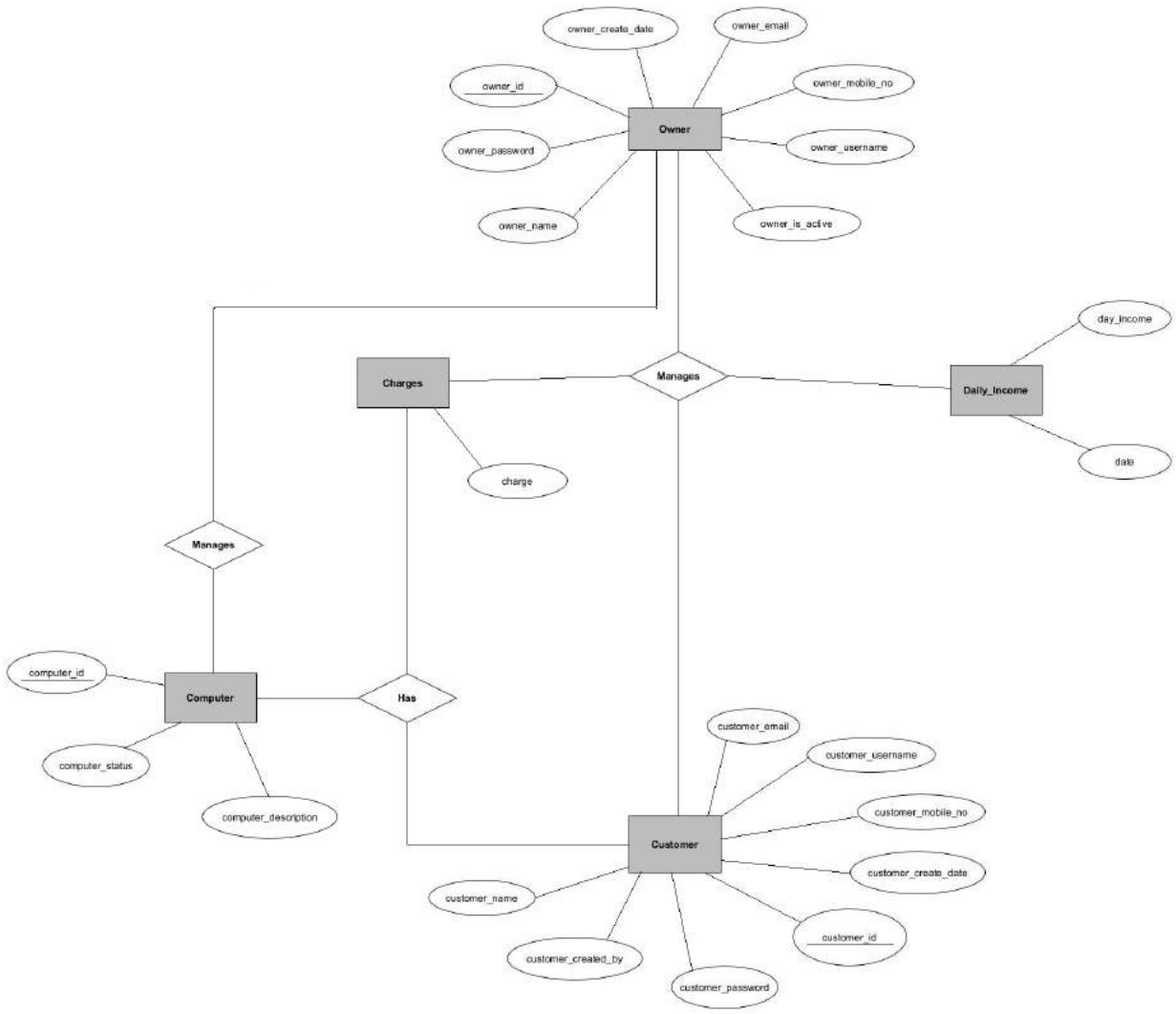
SR NO	Use Case	Description
1	Login	This Use case allows the Customer as well as the owner to login into the system.
2	Logout	Logs Out the Customer as well as the Owner. Ends the session
3	Register	Registers the user by taking personal information
4	View computer Details	Allows the logged in user to check its computer details
5	Manage Personal details	Allows the user to update any personal details from from the update profile. User 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.

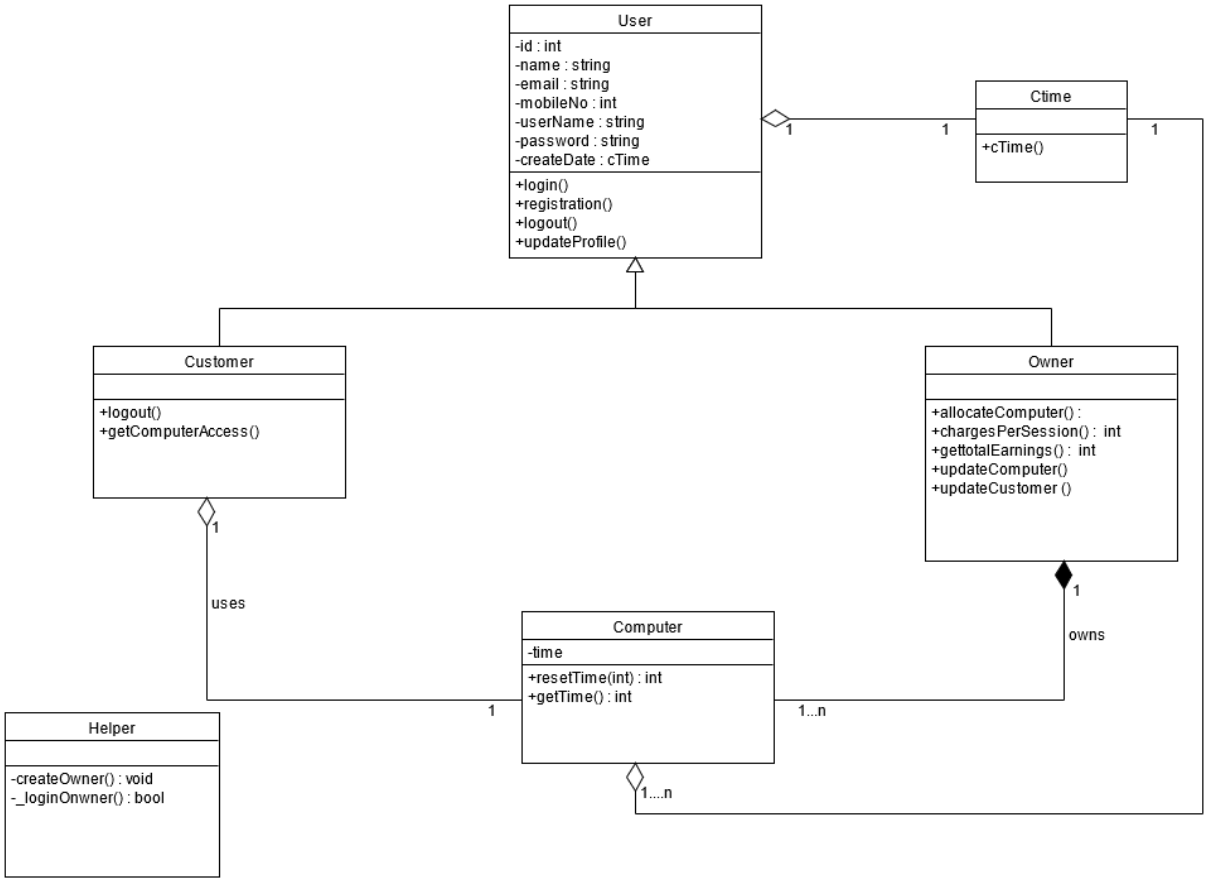
The entity Owner manages all the other entities like Computer, Customer, Charges and Daily Income.

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.

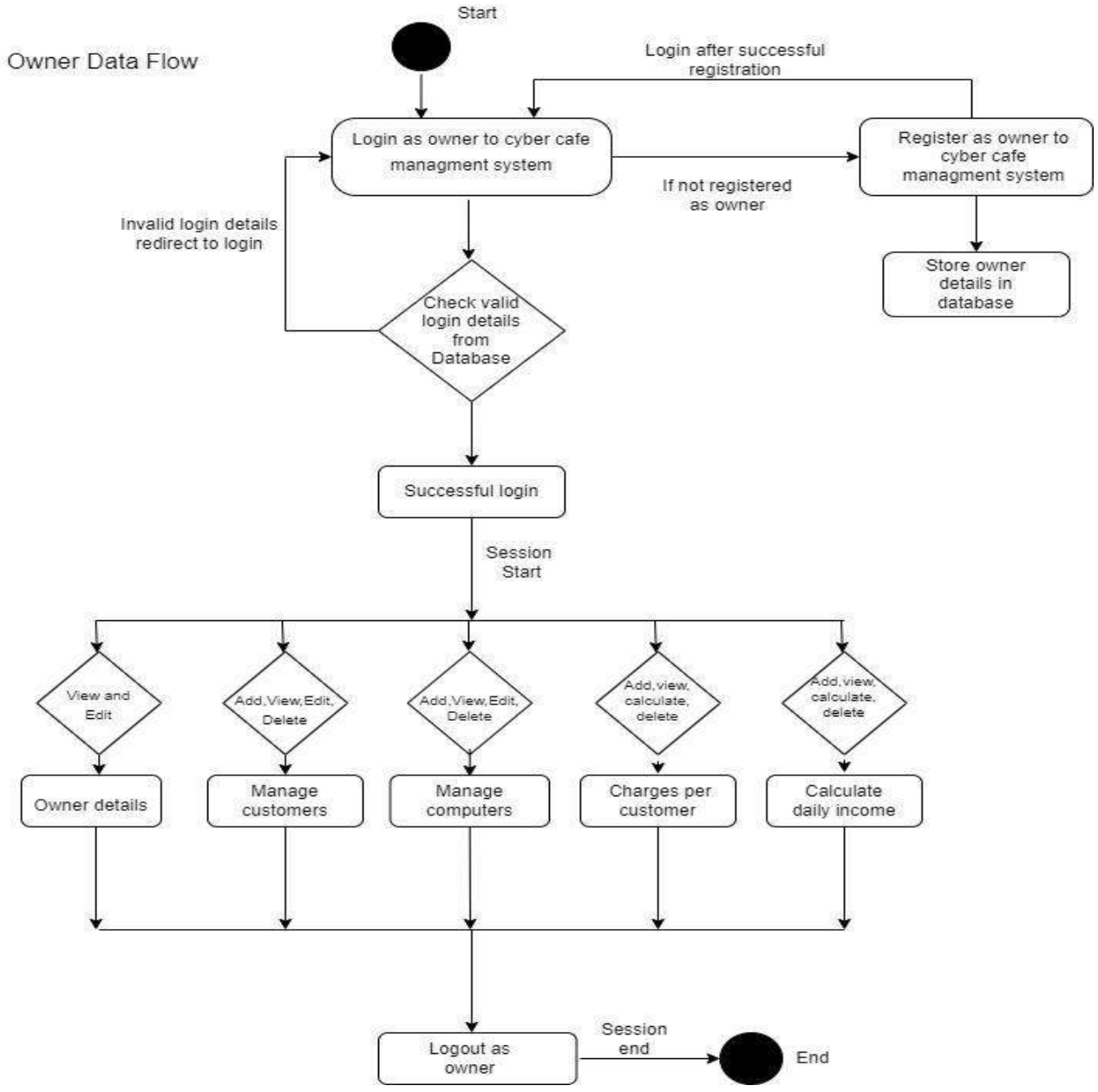


Data Flow Diagrams

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

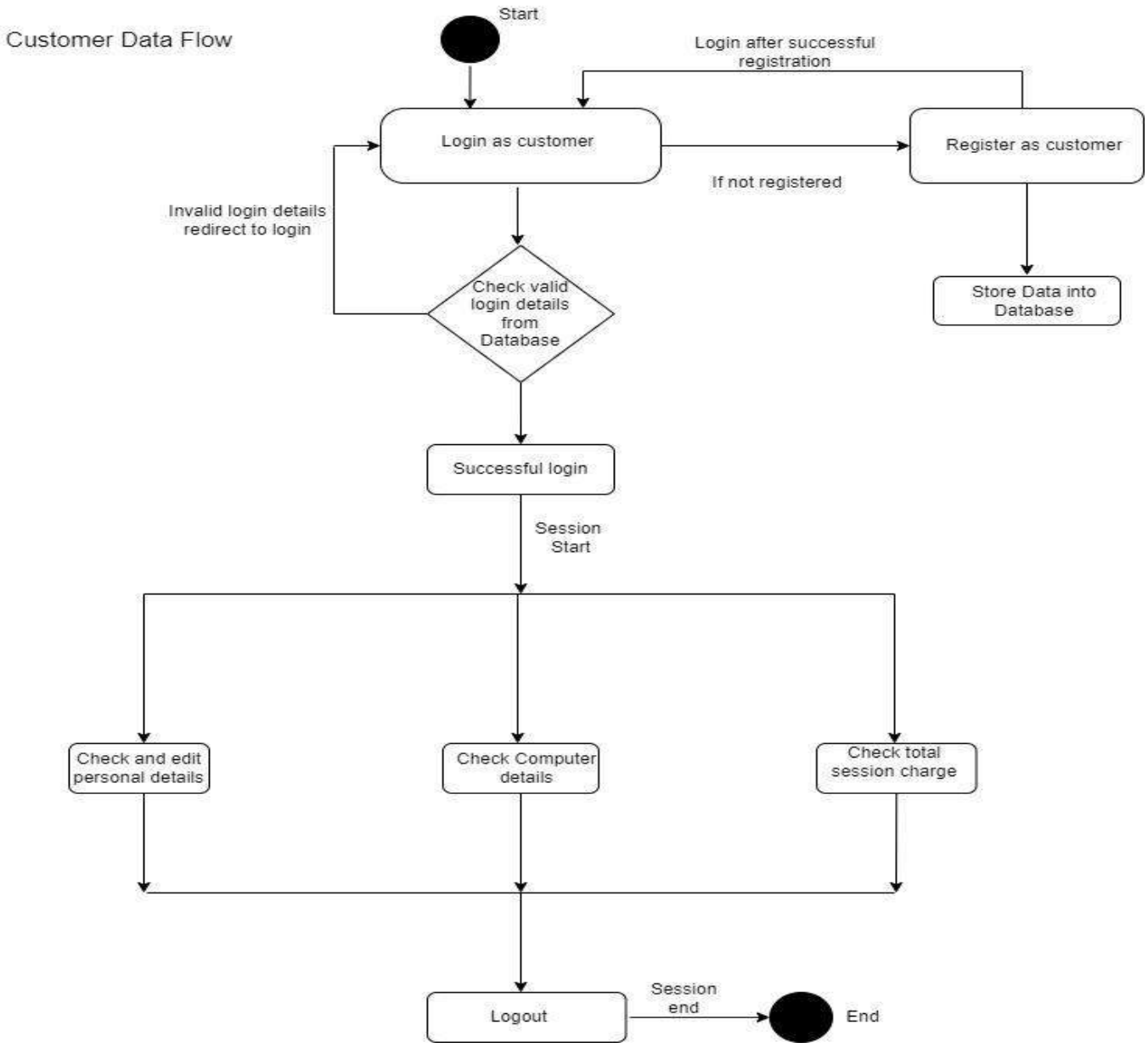
DFD modules a system by using external entities from which data flows to a process, which transforms the data and creates output data flows which go to other process or external entities or files.

1. Owner Data Flow Diagram



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

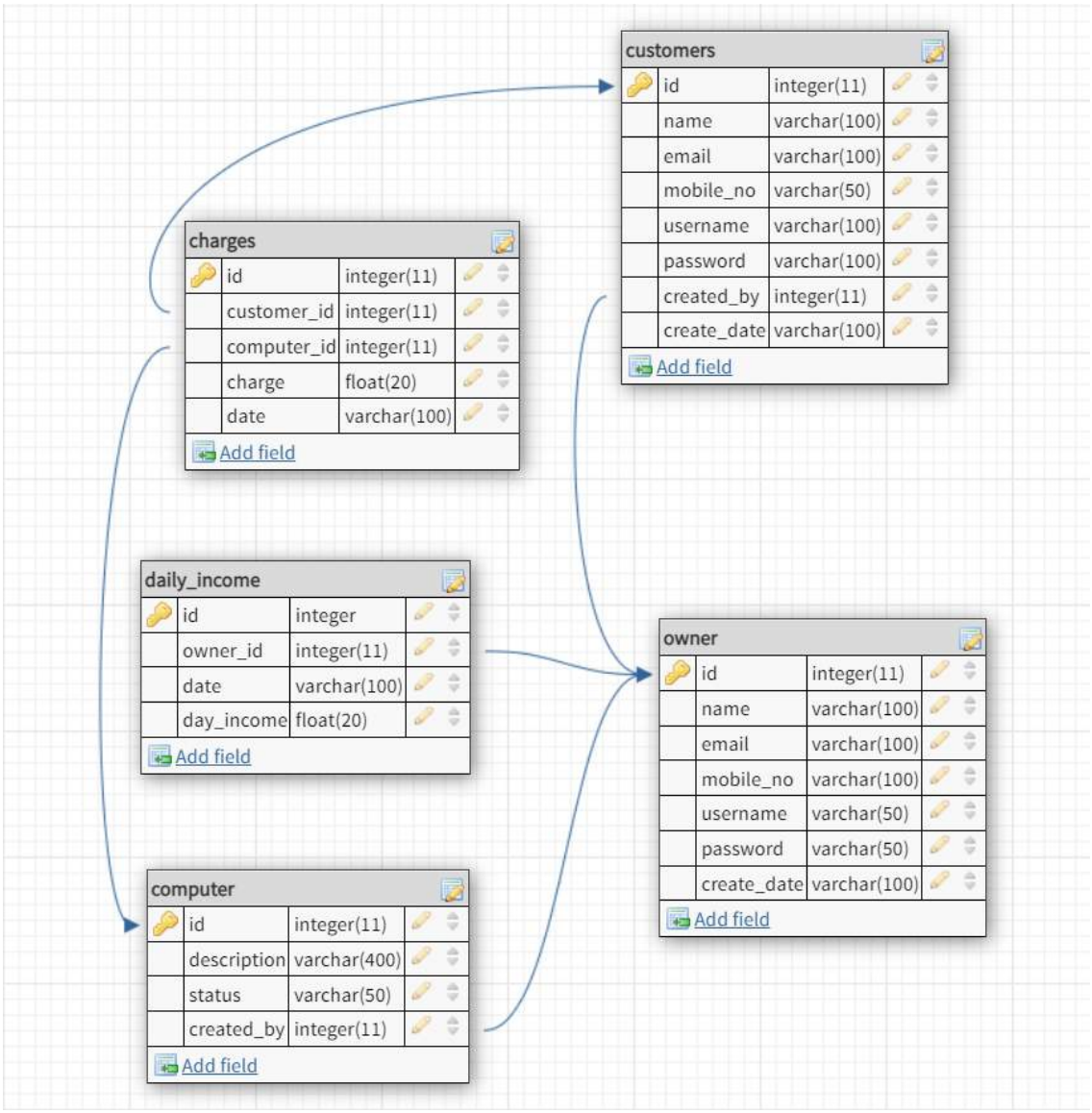
2. Customer Data Flow Diagram.



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

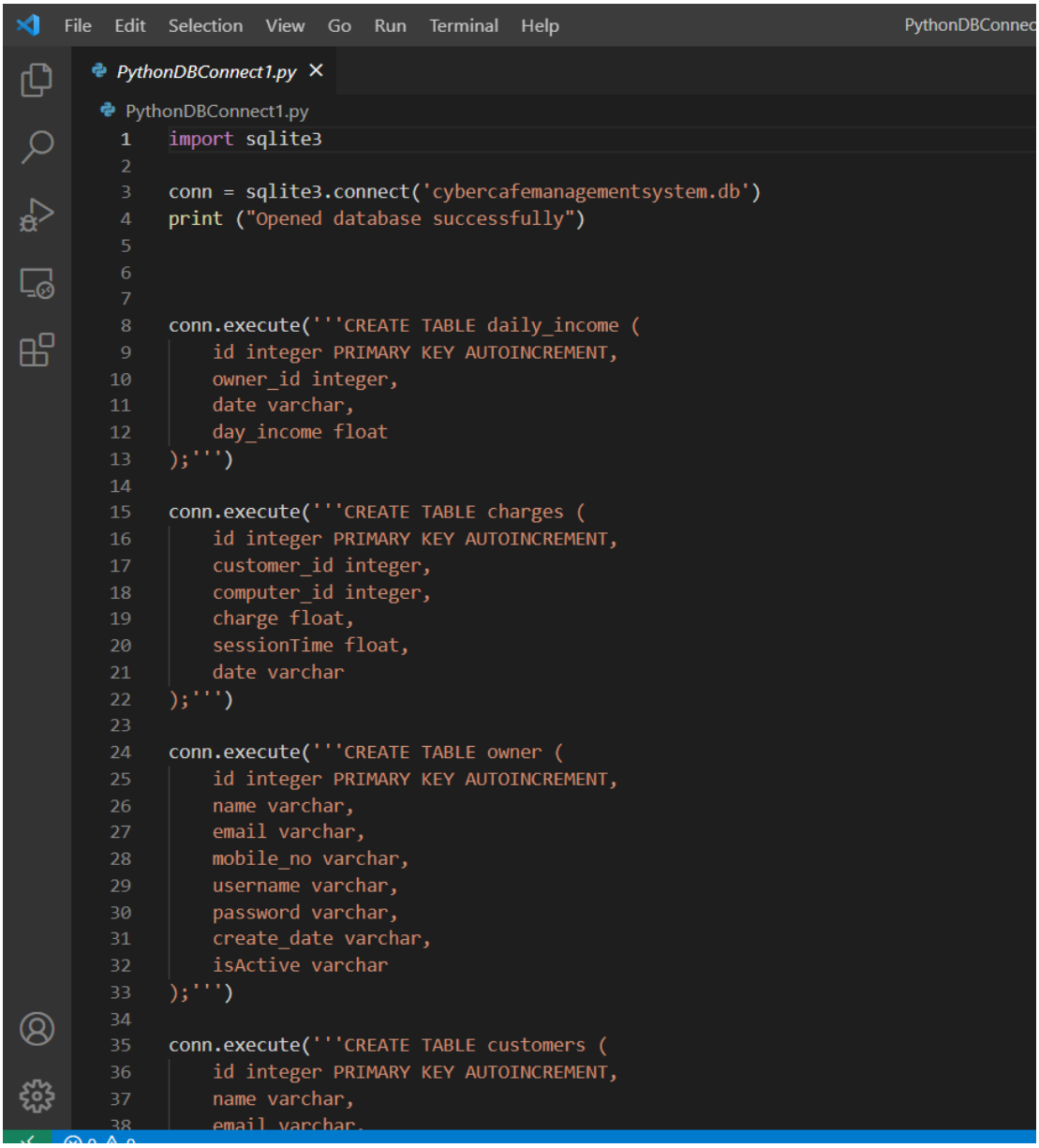
Database Schema

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

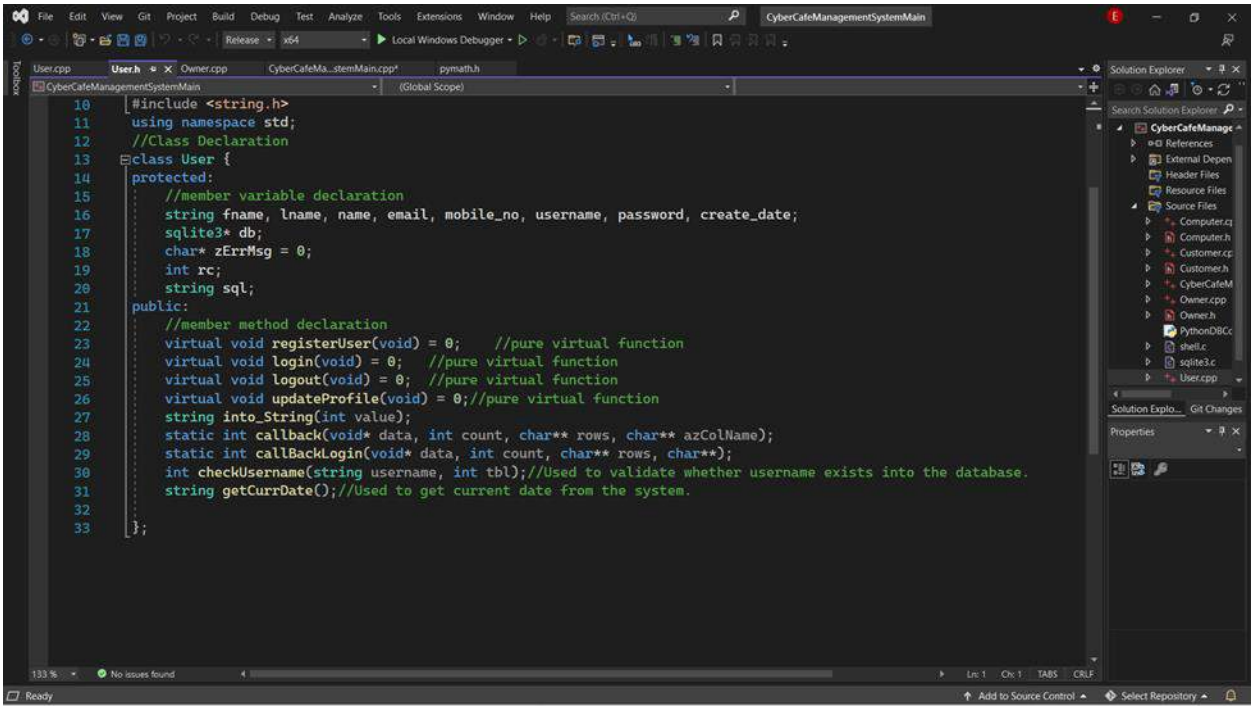
Database



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

User Class

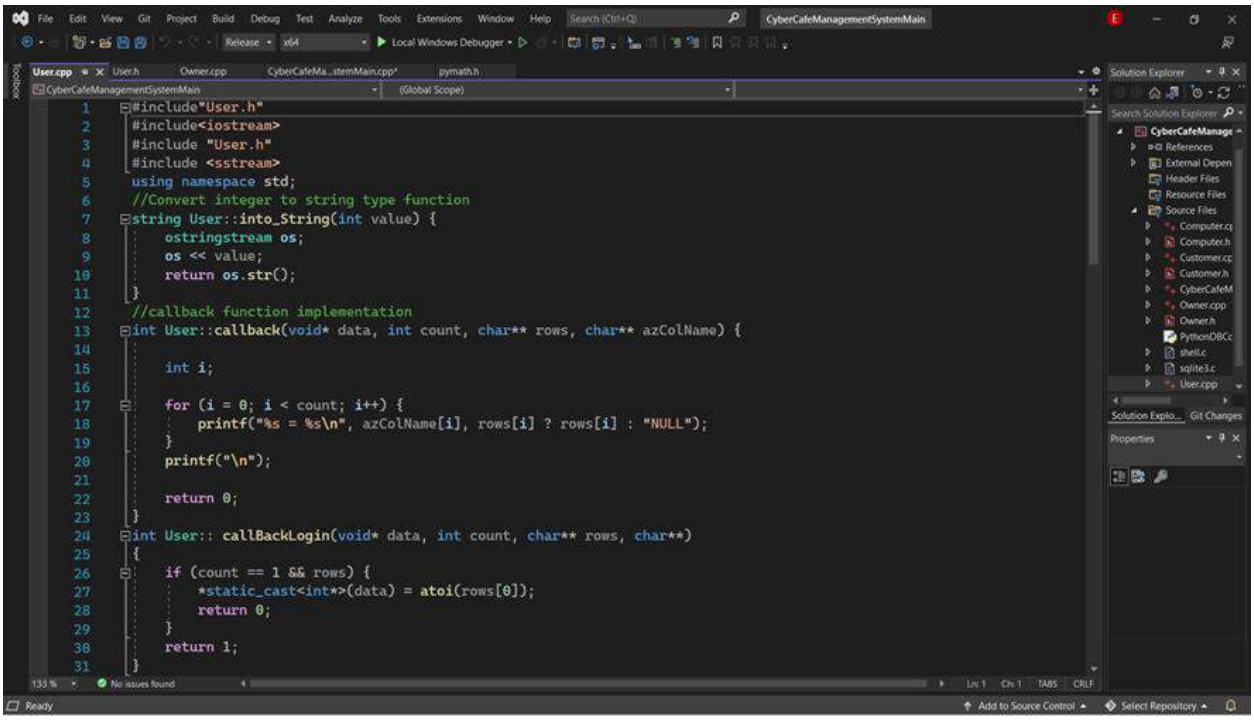
Class Declaration



```
#include <string.h>
using namespace std;
//Class Declaration
class User {
protected:
    //member variable declaration
    string fname, lname, name, email, mobile_no, username, password, create_date;
    sqlite3* db;
    char* zErrMsg = 0;
    int rc;
    string sql;
public:
    //member method declaration
    virtual void registerUser(void) = 0;    //pure virtual function
    virtual void login(void) = 0;   //pure virtual function
    virtual void logout(void) = 0;  //pure virtual function
    virtual void updateProfile(void) = 0;//pure virtual function
    string into_String(int value);
    static int callback(void* data, int count, char** rows, char** azColName);
    static int callBackLogin(void* data, int count, char** rows, char**);
    int checkUsername(string username, int tbl);//Used to validate whether username exists into the database.
    string getCurrDate();//Used to get current date from the system.

};
```

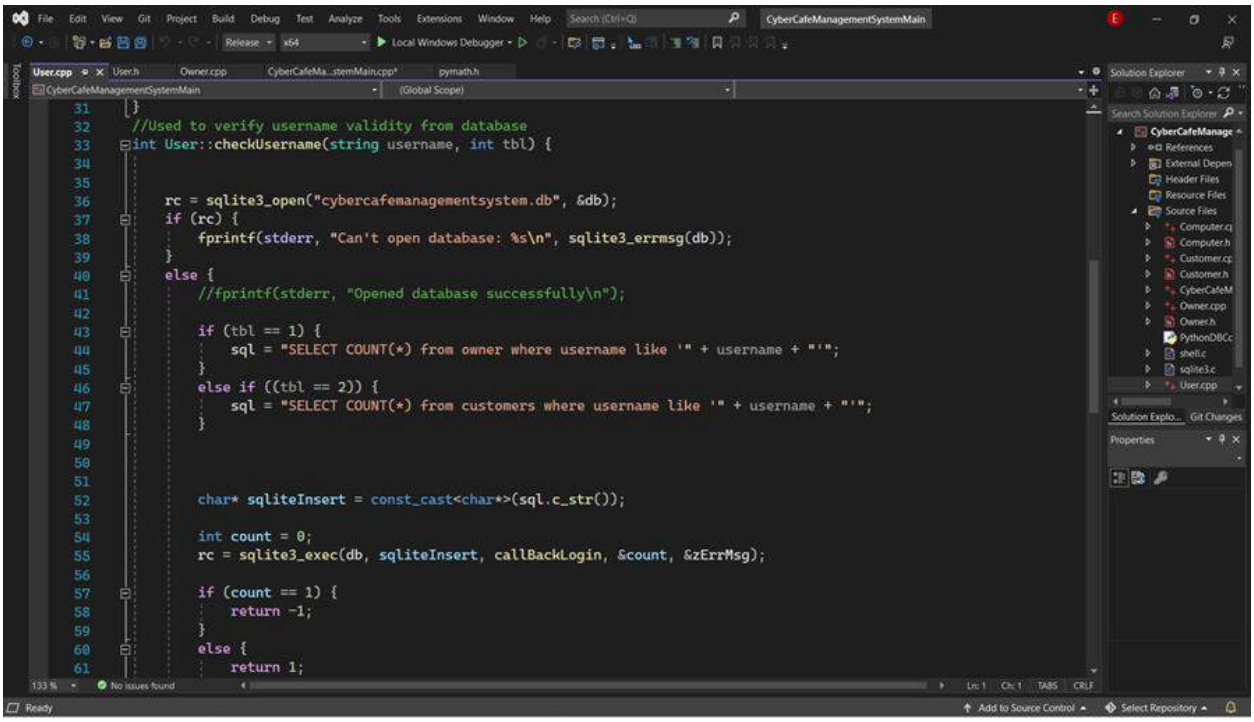
User class contains virtual methods for registration, login, logout, update profile for both owner and customer which will get override after inheritance.

Class definition



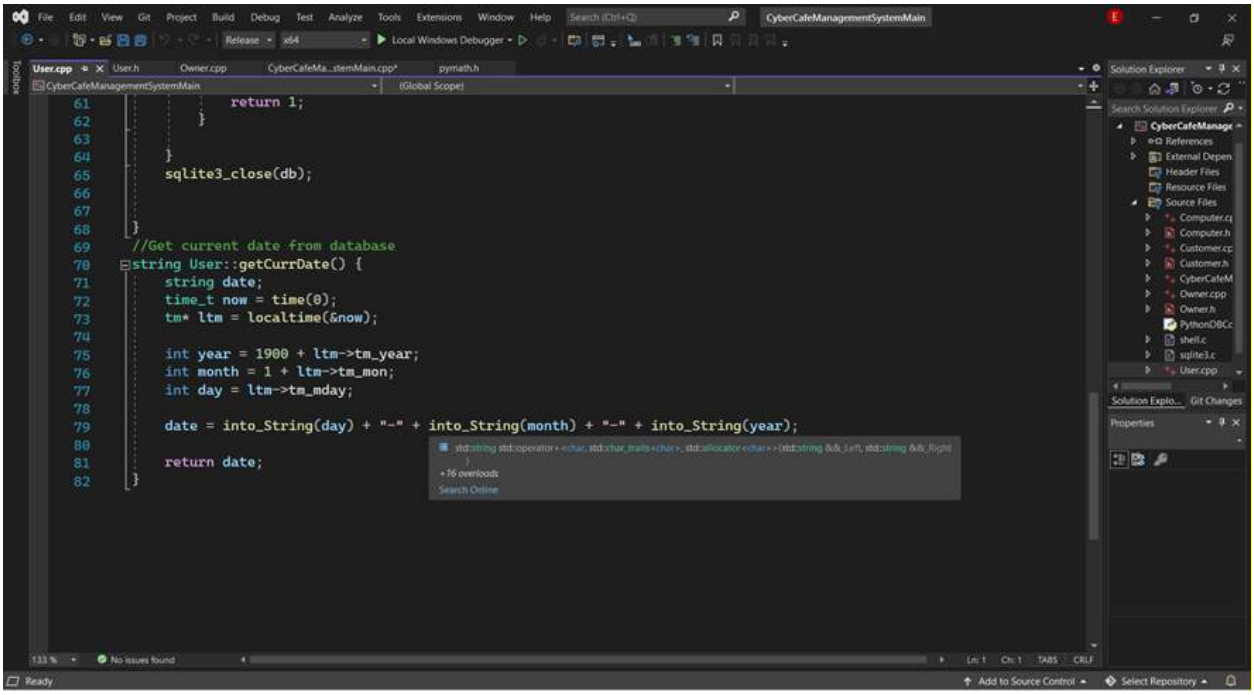
```cpp
#include"User.h"
#include<iostream>
#include "User.h"
#include <sstream>
using namespace std;
//Convert integer to string type function
string User::into_String(int value) {
    ostringstream os;
    os << value;
    return os.str();
}
//callback function implementation
int User::callback(void* data, int count, char** rows, char** azColName) {

    int i;

    for (i = 0; i < count; i++) {
        printf("%s = %s\n", azColName[i], rows[i] ? rows[i] : "NULL");
    }
    printf("\n");

    return 0;
}
int User:: callBackLogin(void* data, int count, char** rows, char**)
{
    if (count == 1 && rows) {
        *static_cast<int*>(data) = atoi(rows[0]);
        return 0;
    }
    return 1;
}
```

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



```
}
//Used to verify username validity from database
int User::checkUsername(string username, int tbl) {


    rc = sqlite3_open("cybercafemanagementsystem.db", &db);
    if (rc) {
        fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
    }
    else {
        //fprintf(stderr, "Opened database successfully\n");

        if (tbl == 1) {
            sql = "SELECT COUNT(*) from owner where username like '" + username + "'";
        }
        else if ((tbl == 2)) {
            sql = "SELECT COUNT(*) from customers where username like '" + username + "'";
        }



        char* sqliteInsert = const_cast<char*>(sql.c_str());

        int count = 0;
        rc = sqlite3_exec(db, sqliteInsert, callBackLogin, &count, &zErrMsg);

        if (count == 1) {
            return -1;
        }
        else {
            return 1;
```

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

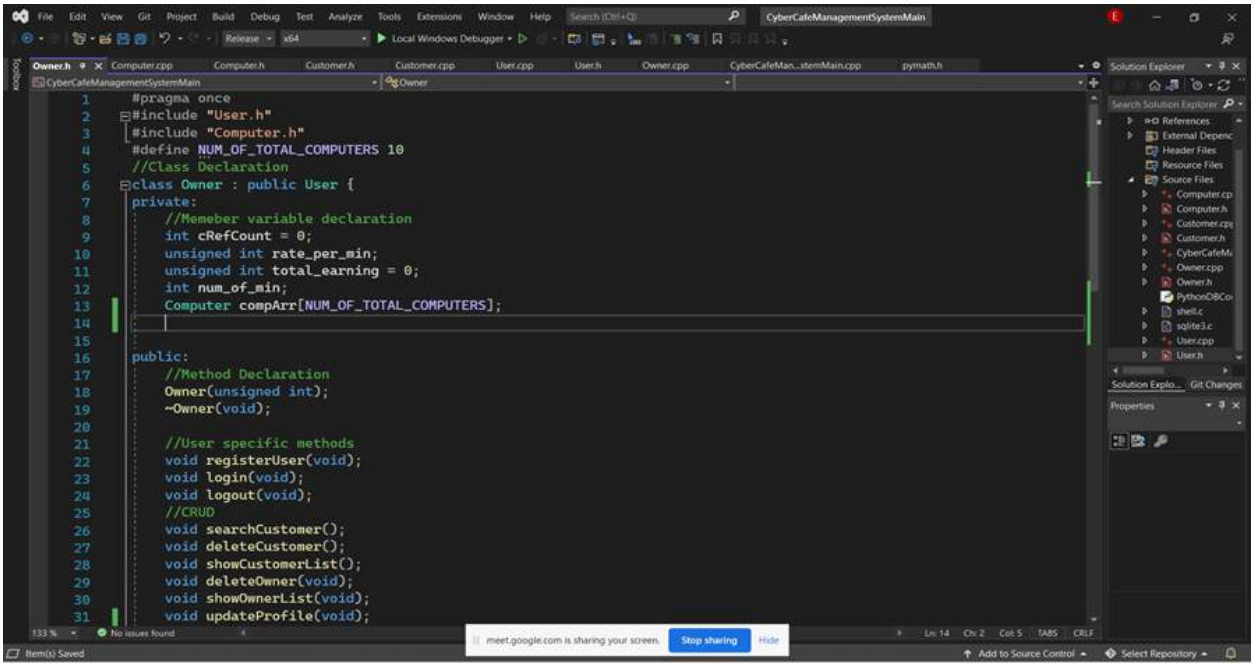


```
            return 1;
        }

    }
    sqlite3_close(db);


}
//Get current date from database
string User::getCurrDate() {
    string date;
    time_t now = time(0);
    tm* ltm = localtime(&now);

    int year = 1900 + ltm->tm_year;
    int month = 1 + ltm->tm_mon;
    int day = ltm->tm_mday;

    date = into_String(day) + "-" + into_String(month) + "-" + into_String(year);

    return date;
}
```

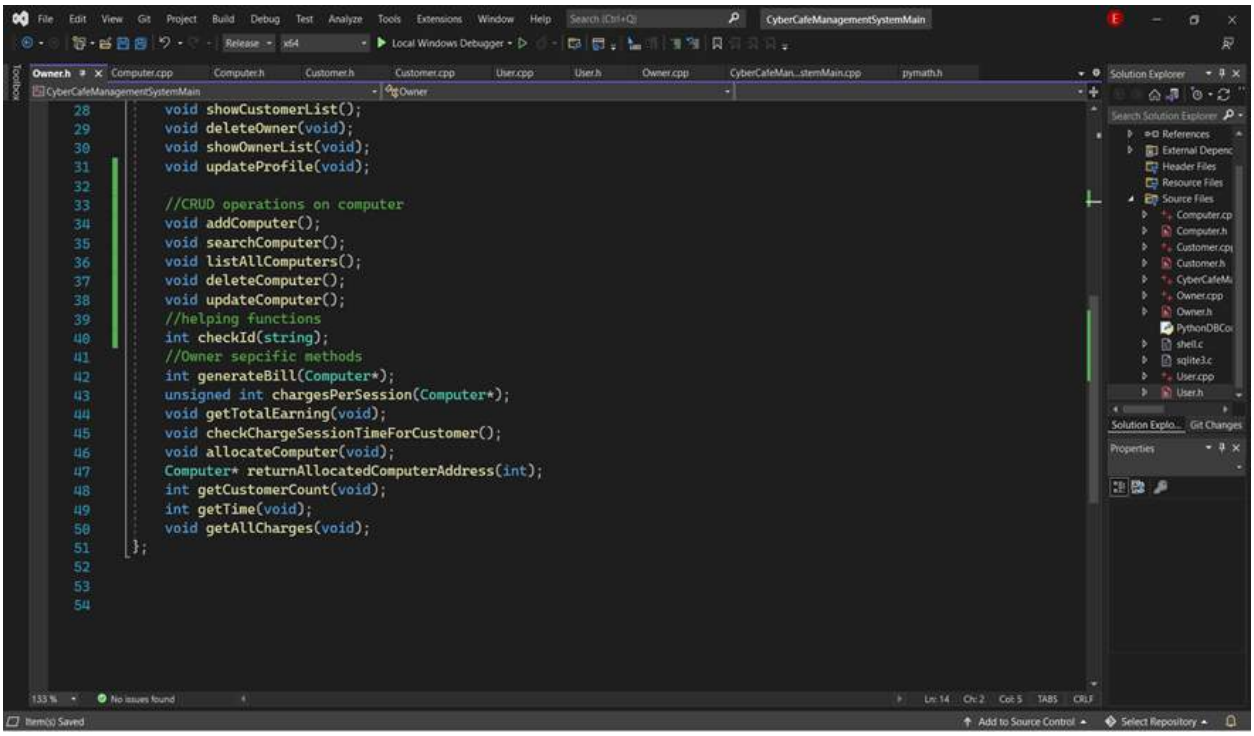
getCurrDate returns today's date in string format.

Owner class

Class Declaration

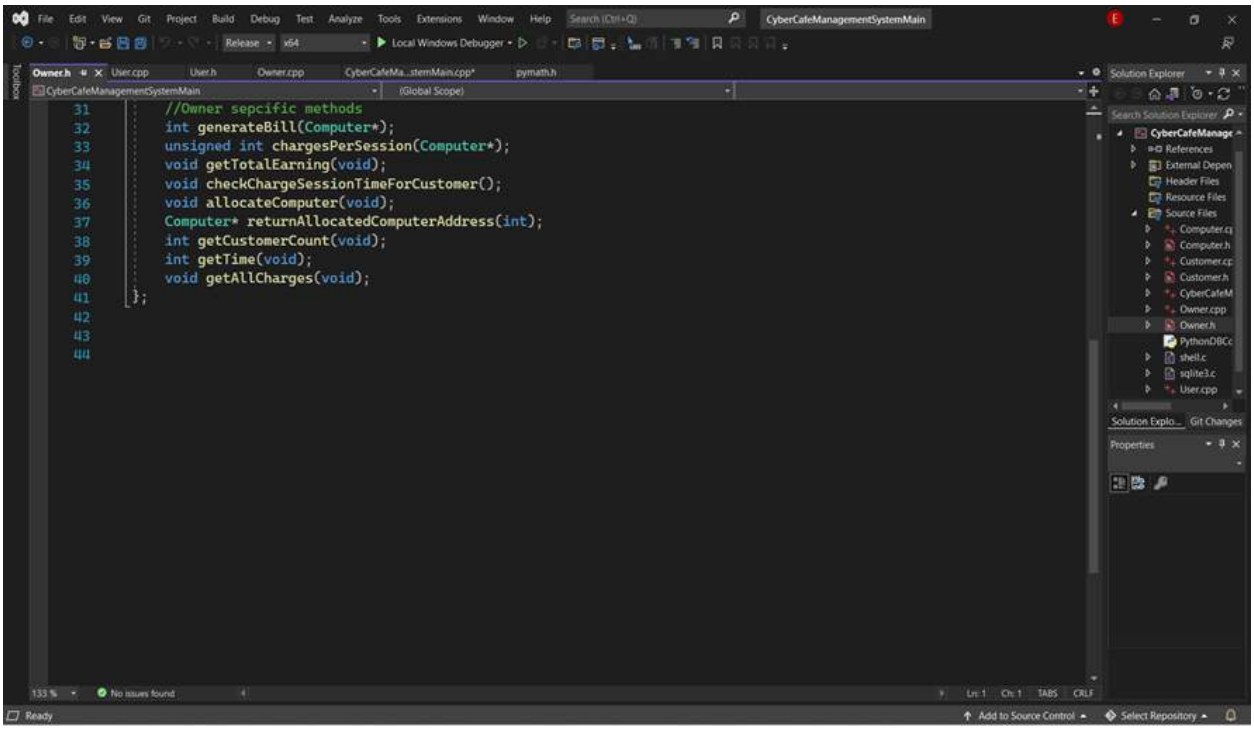


```
#pragma once
#include "User.h"
#include "Computer.h"
#define NUM_OF_TOTAL_COMPUTERS 10
//Class Declaration
class Owner : public User {
private:
    //Memeber variable declaration
    int cRefCount = 0;
    unsigned int rate_per_min;
    unsigned int total_earning = 0;
    int num_of_min;
    Computer compArr[NUM_OF_TOTAL_COMPUTERS];

public:
    //Method Declaration
    Owner(unsigned int);
    ~Owner(void);

    //User specific methods
    void registerUser(void);
    void login(void);
    void logout(void);
    //CRUD
    void searchCustomer();
    void deleteCustomer();
    void showCustomerList();
    void deleteOwner(void);
    void showOwnerList(void);
    void updateProfile(void);
```



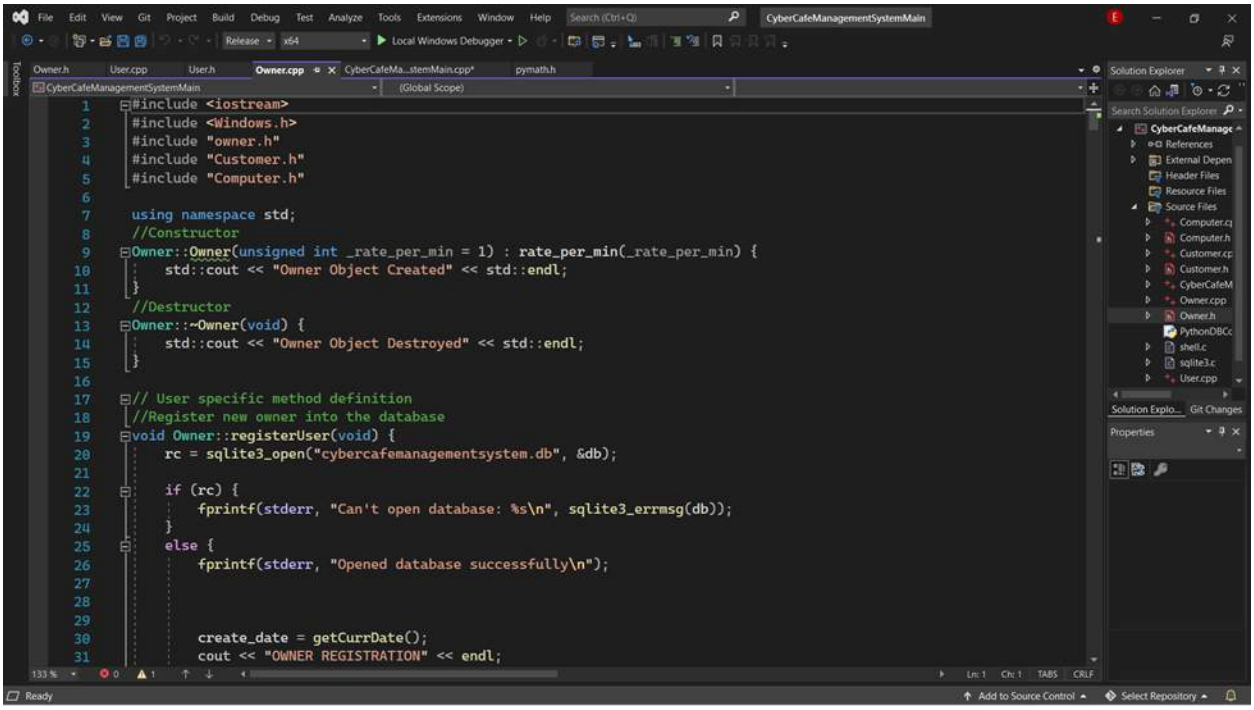
```
    void showCustomerList();
    void deleteOwner(void);
    void showOwnerList(void);
    void updateProfile(void);

    //CRUD operations on computer
    void addComputer();
    void searchComputer();
    void listAllComputers();
    void deleteComputer();
    void updateComputer();
    //helping functions
    int checkId(string);
    //Owner sepcific methods
    int generateBill(Computer*);
    unsigned int chargesPerSession(Computer*);
    void getTotalEarning(void);
    void checkChargeSessionTimeForCustomer();
    void allocateComputer(void);
    Computer* returnAllocatedComputerAddress(int);
    int getCustomerCount(void);
    int getTime(void);
    void getAllCharges(void);
};
```

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



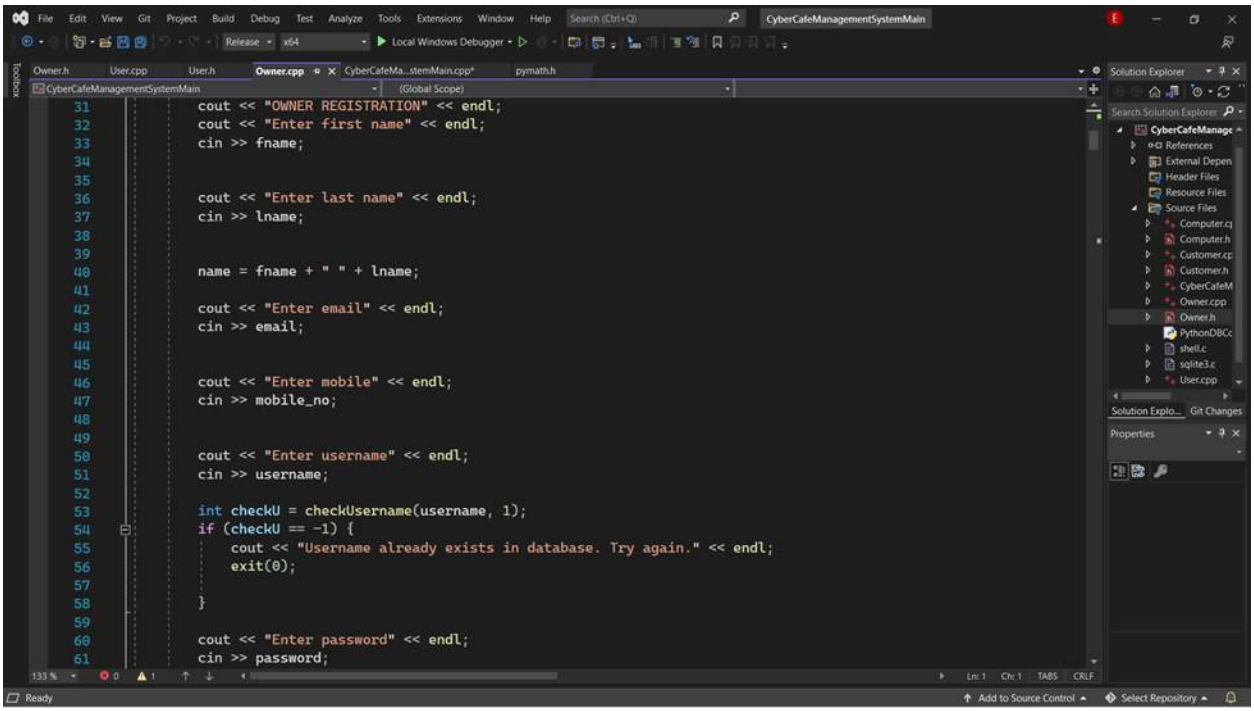
Declaration to allocate Computer, get today's total income and get the list of charges.

Class Definition

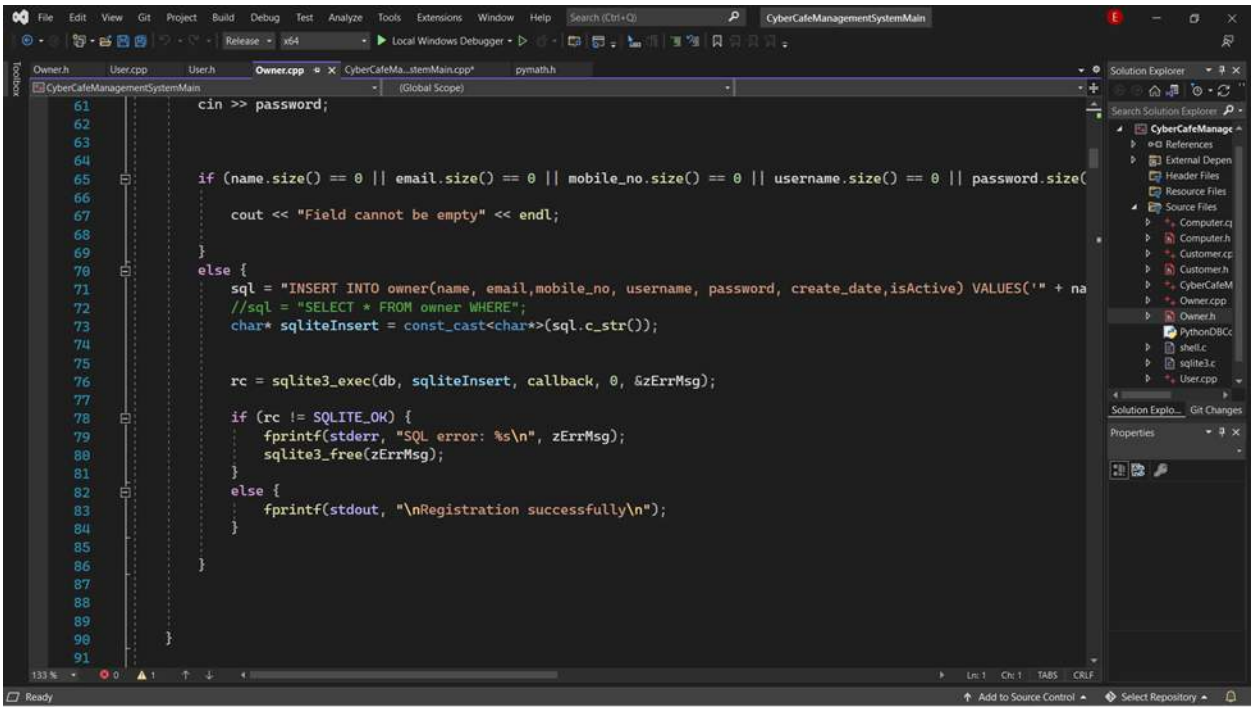


```cpp
#include <iostream>
#include <Windows.h>
#include "owner.h"
#include "Customer.h"
#include "Computer.h"

using namespace std;
//Constructor
Owner::Owner(unsigned int _rate_per_min = 1) : rate_per_min(_rate_per_min) {
    std::cout << "Owner Object Created" << std::endl;
}
//Destructor
Owner::~Owner(void) {
    std::cout << "Owner Object Destroyed" << std::endl;
}

// User specific method definition
//Register new owner into the database
void Owner::registerUser(void) {
    rc = sqlite3_open("cybercafemanagementsystem.db", &db);

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



        create_date = getCurrDate();
        cout << "OWNER REGISTRATION" << endl;
```

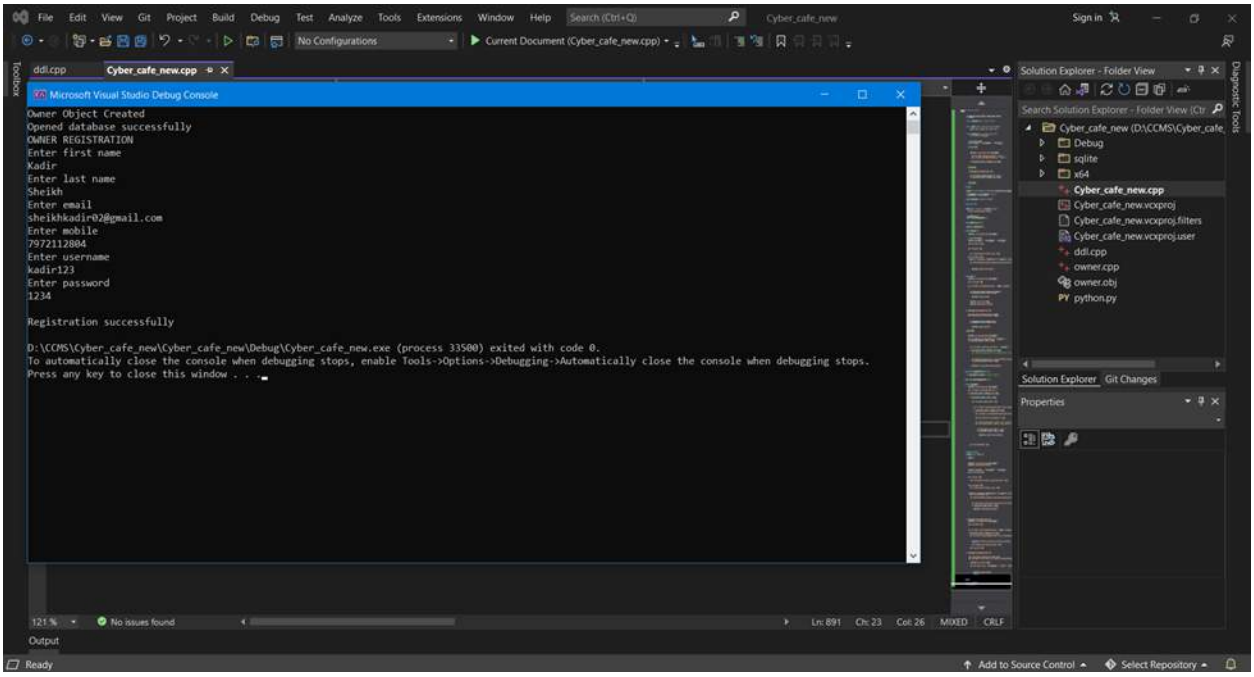
Definition of owner constructor and destructor.



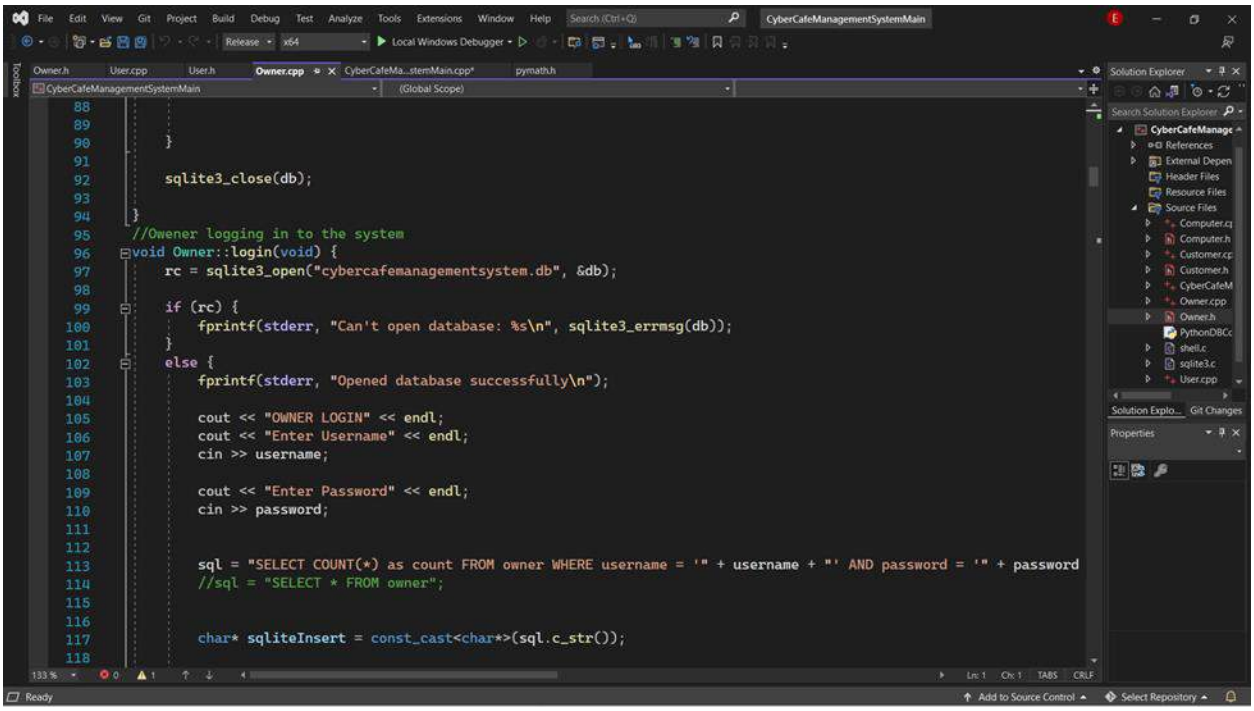
```
        cout << "OWNER REGISTRATION" << endl;
        cout << "Enter first name" << endl;
        cin >> fname;


        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;
```

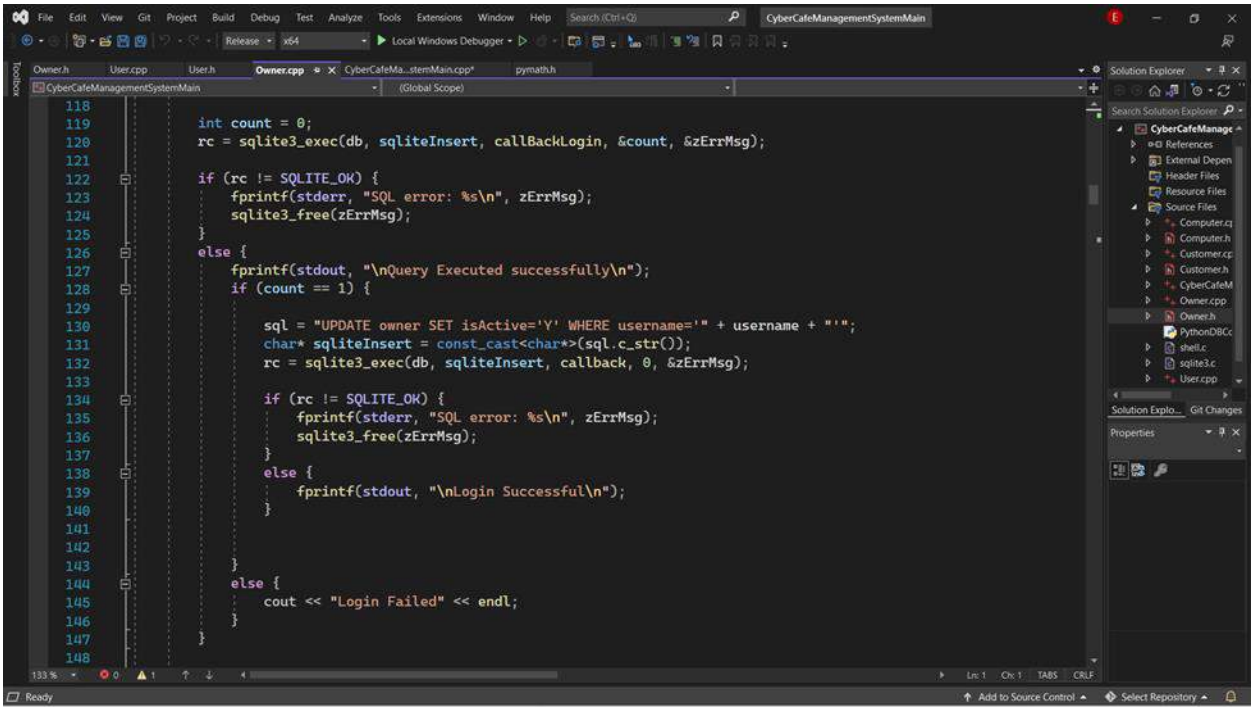


```
        cin >> password;



        if (name.size() == 0 || email.size() == 0 || mobile_no.size() == 0 || username.size() == 0 || password.size(

            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");
            }

        }


    }
```

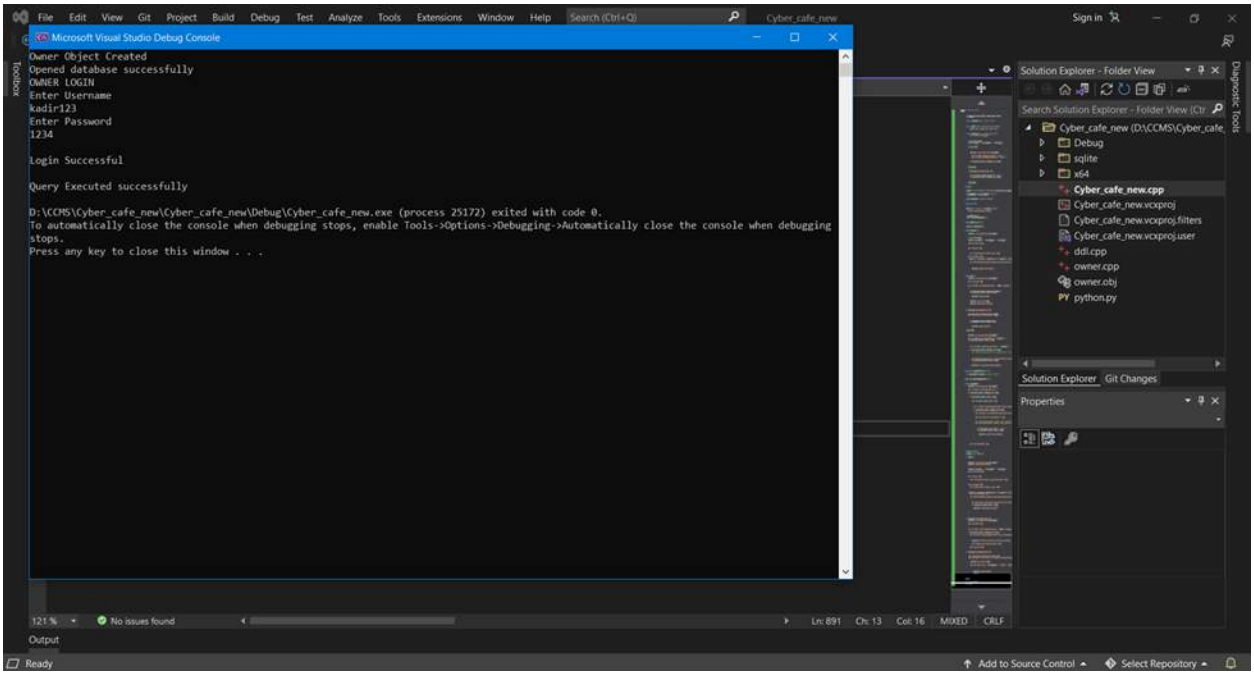


Definition of owner registration function which is overriding from User class virtual registration function.



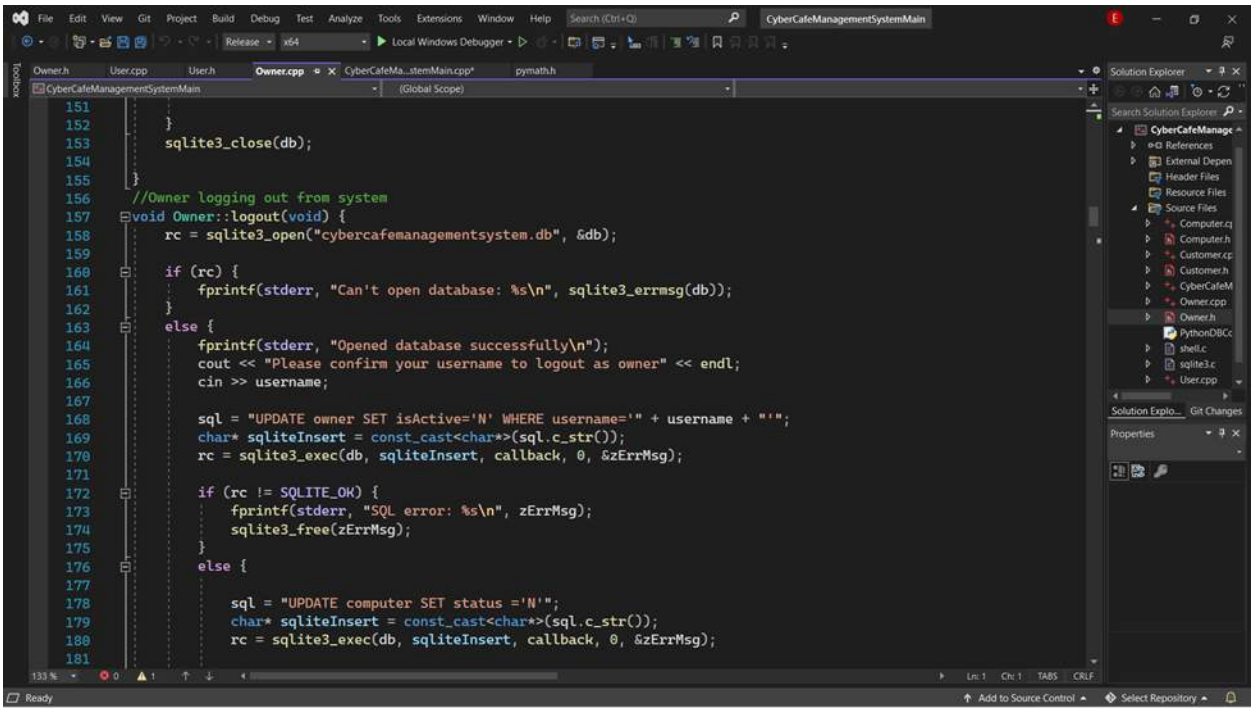


```
118
119         int count = 0;
120         rc = sqlite3_exec(db, sqliteInsert, callBackLogin, &count, &zErrMsg);
121
122         if (rc != SQLITE_OK) {
123             fprintf(stderr, "SQL error: %s\n", zErrMsg);
124             sqlite3_free(zErrMsg);
125         }
126         else {
127        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

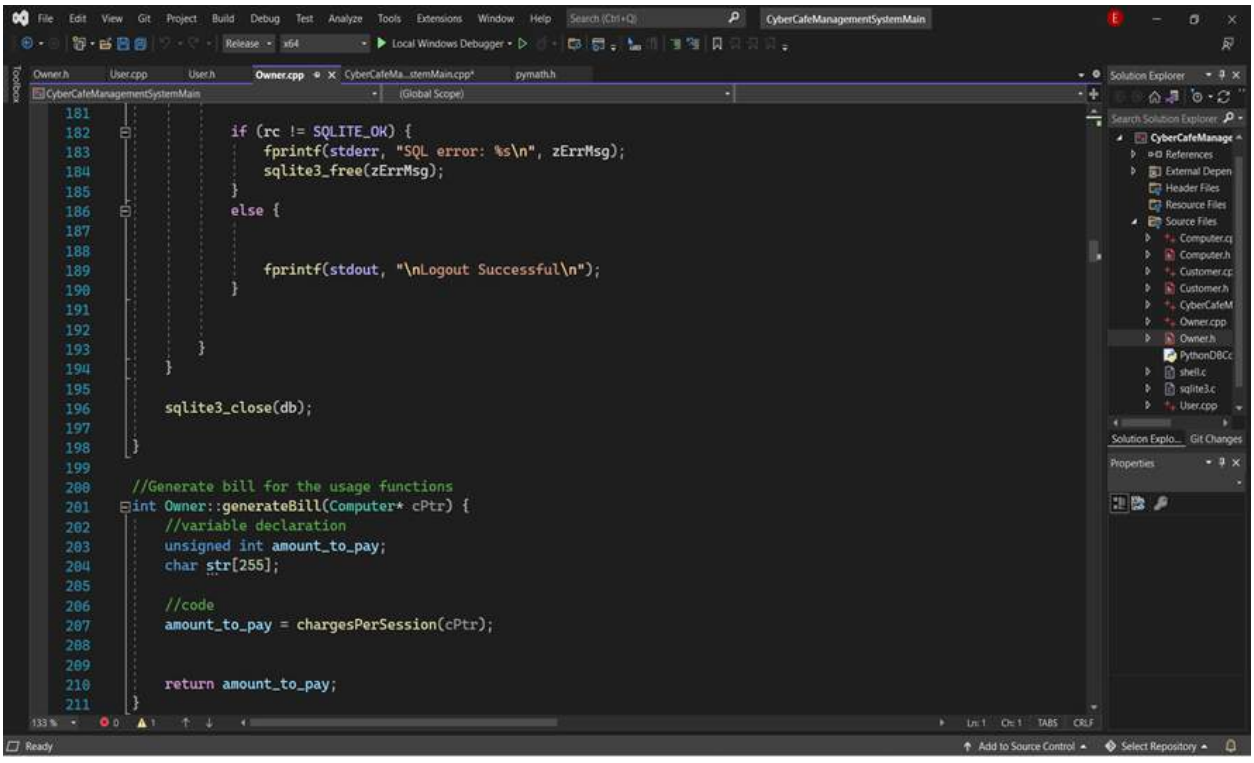


```
Microsoft Visual Studio Debug Console
Owner Object Created
Opened database successfully
OWNER LOGIN
Enter Username
kadir123
Enter Password
1234

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 . . .
```

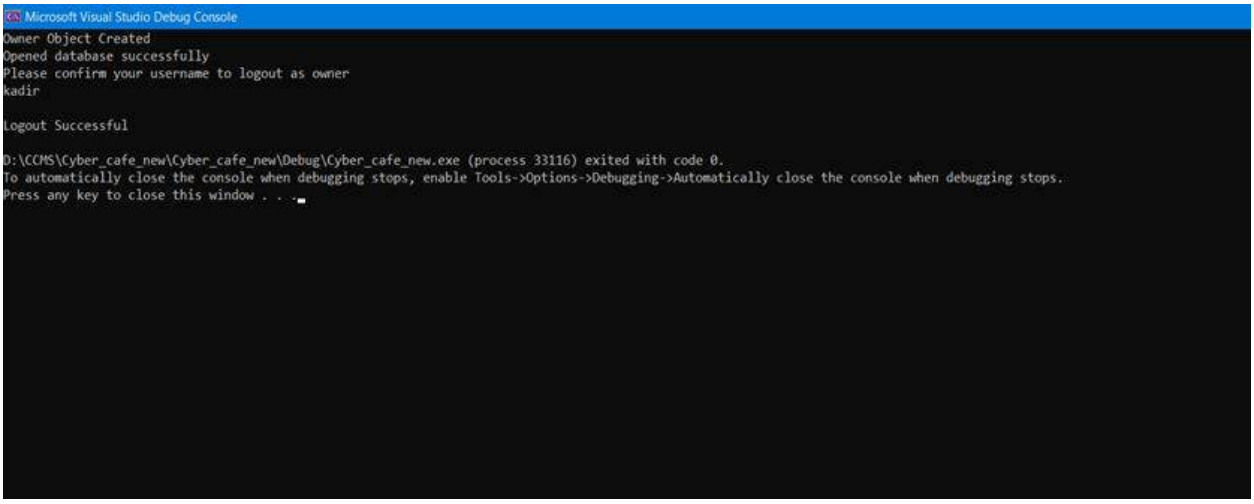
Definition of owner login function which is overriding from User class virtual login function.



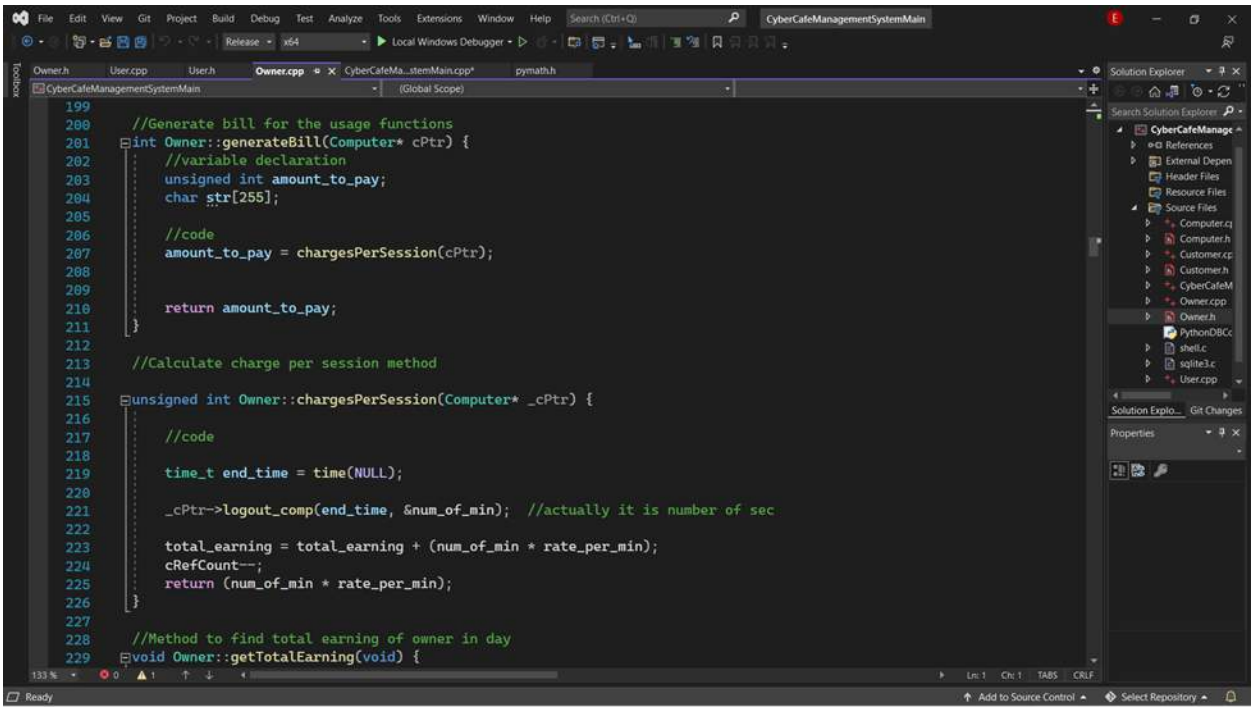
```
151
152         }
153         sqlite3_close(db);
154
155     }
156     //Owner logging out from system
157     void Owner::logout(void) {
158         rc = sqlite3_open("cybercafemanagementsystem.db", &db);
159
160         if (rc) {
161             fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
162         }
163         else {
164             fprintf(stderr, "Opened database successfully\n");
165             cout << "Please confirm your username to logout as owner" << endl;
166             cin >> username;
167
168             sql = "UPDATE owner SET isActive='N' WHERE username='" + username + "'";
169             char* sqliteInsert = const_cast<char*>(sql.c_str());
170             rc = sqlite3_exec(db, sqliteInsert, callback, 0, &zErrMsg);
171
172             if (rc != SQLITE_OK) {
173                 fprintf(stderr, "SQL error: %s\n", zErrMsg);
174                 sqlite3_free(zErrMsg);
175             }
176             else {
177
178                 sql = "UPDATE computer SET status ='N'";
179                 char* sqliteInsert = const_cast<char*>(sql.c_str());
180                 rc = sqlite3_exec(db, sqliteInsert, callback, 0, &zErrMsg);
181
```



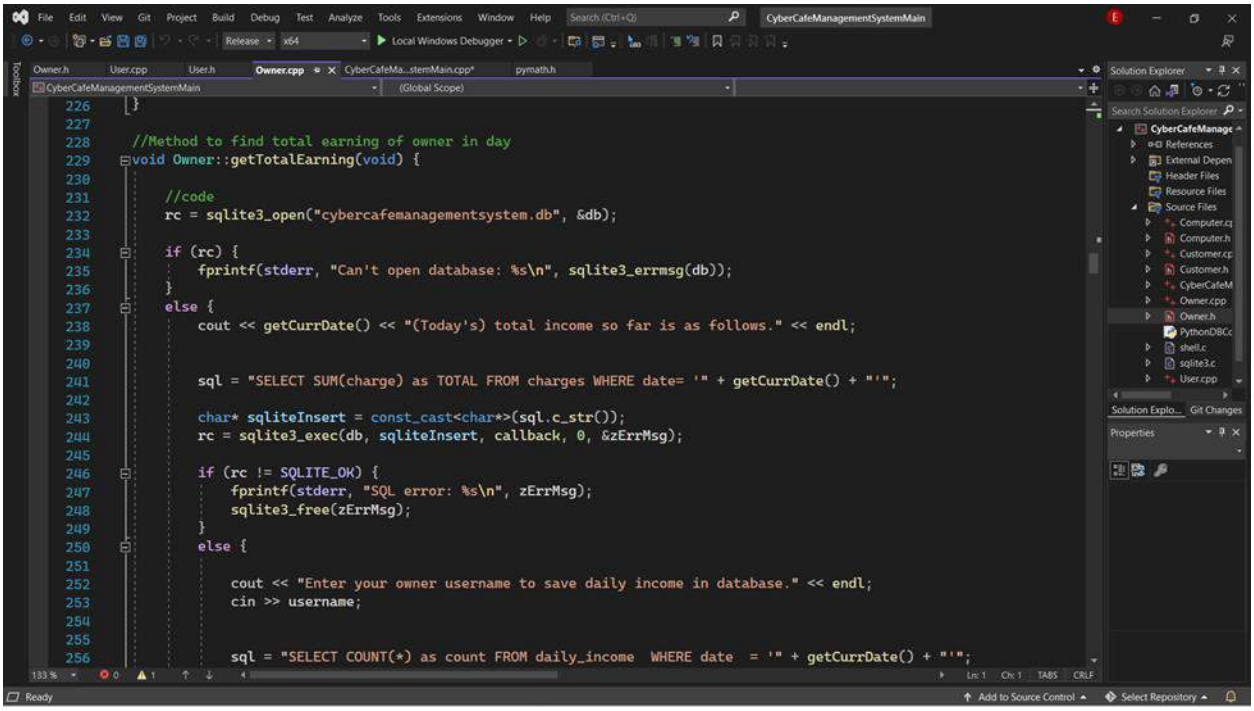
```
181
182             if (rc != SQLITE_OK) {
183                 fprintf(stderr, "SQL error: %s\n", zErrMsg);
184                 sqlite3_free(zErrMsg);
185             }
186             else {
187
188
189                 fprintf(stdout, "\nLogout Successful\n");
190             }
191
192
193             }
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     }
```



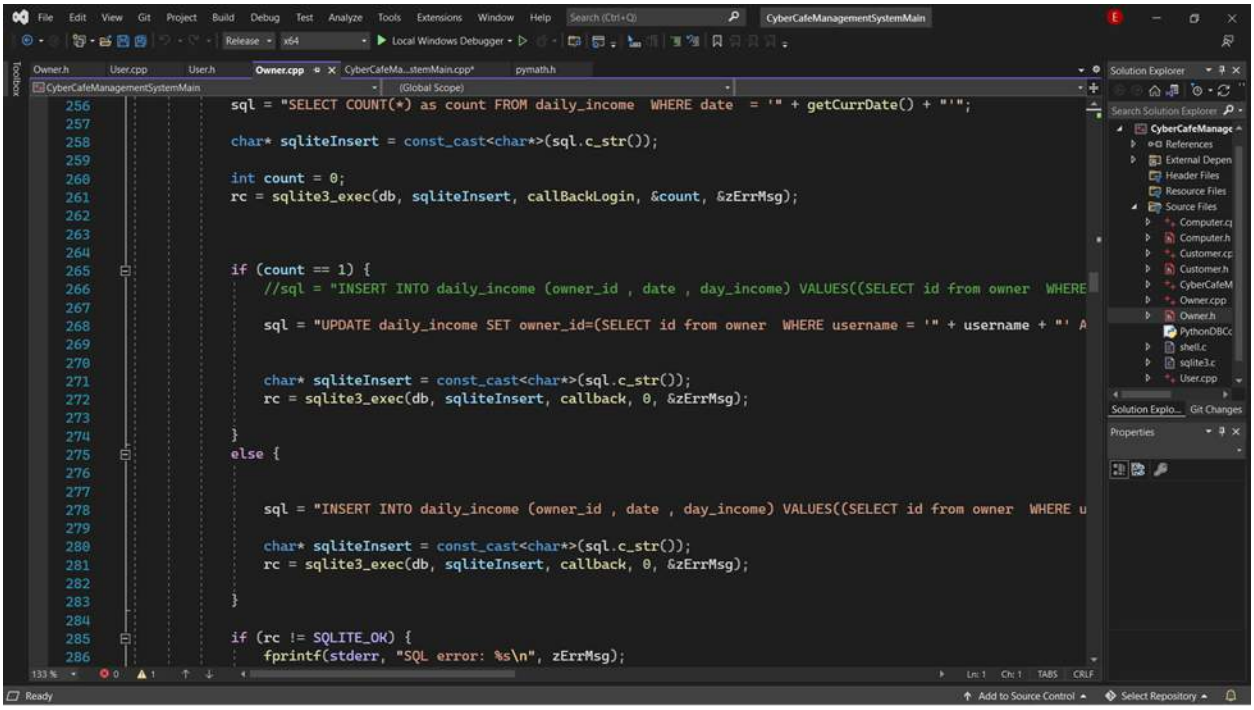
Definition of owner logout function and generate bill which returns total amount of money for the session spent by the customer on the allocated computer.



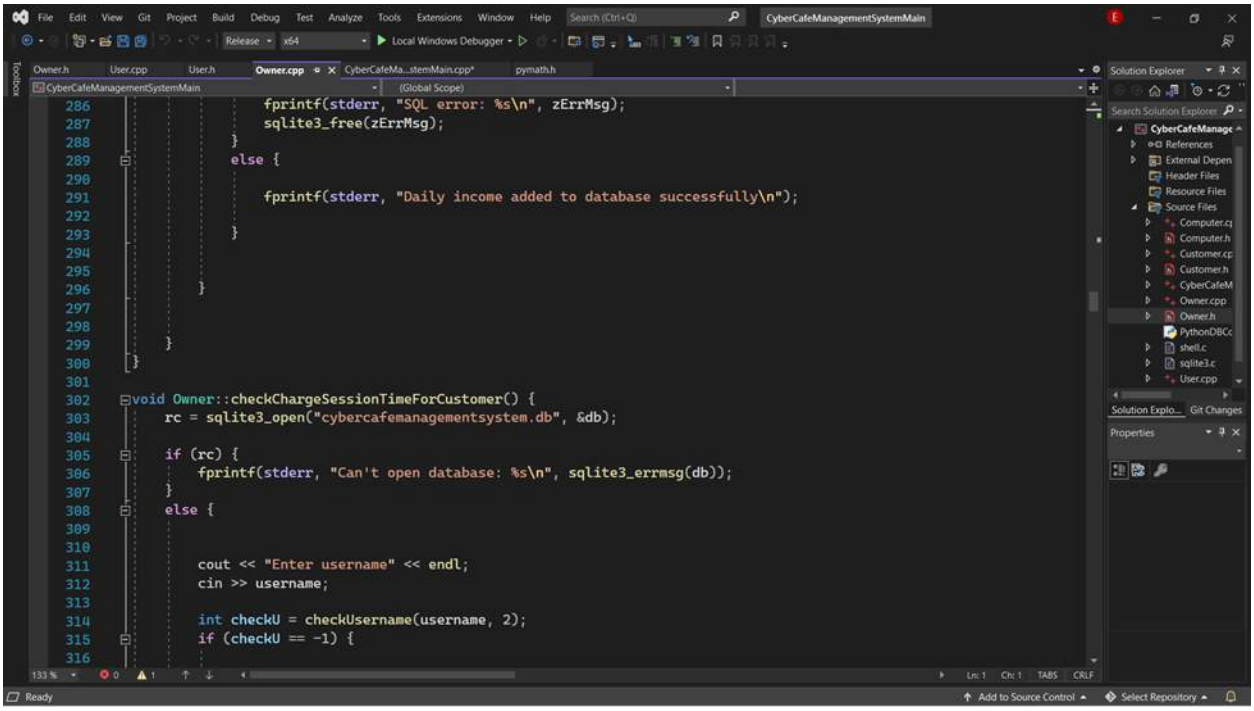
Definition of charge per session which calculate total amount of money.



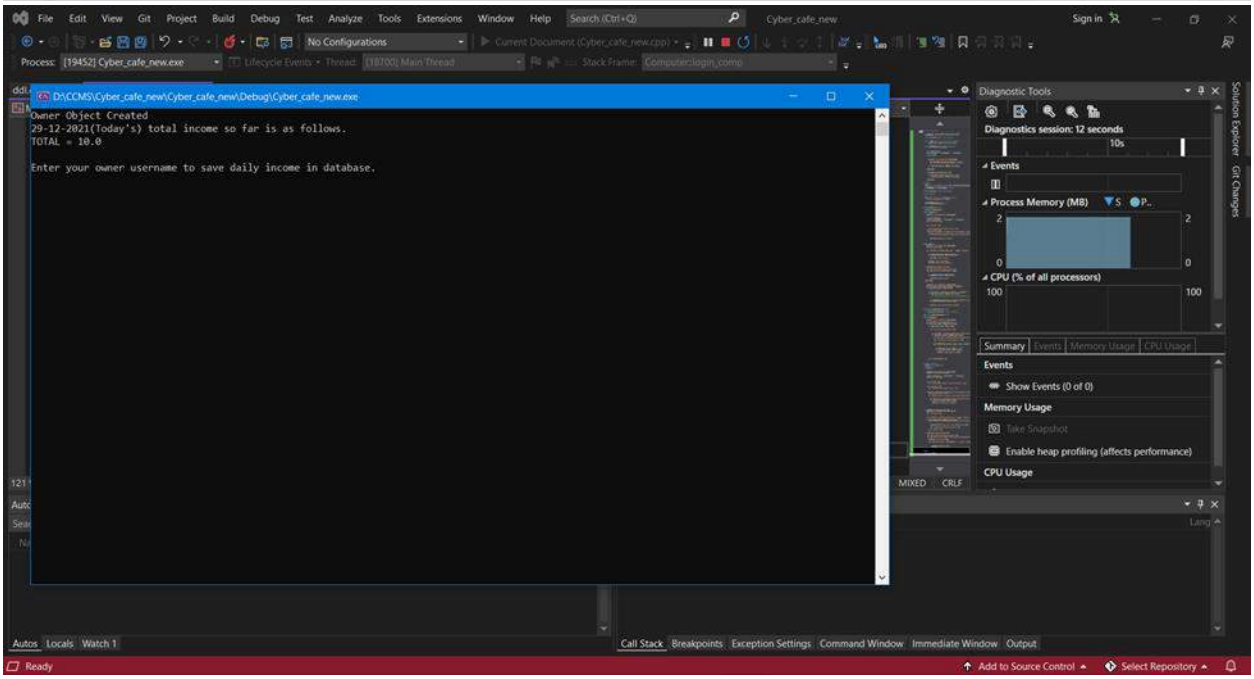
```
226     }
227
228     //Method to find total earning of owner in day
229     void Owner::getTotalEarning(void) {
230
231         //code
232         rc = sqlite3_open("cybercafemanagementsystem.db", &db);
233
234         if (rc) {
235             fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));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



```
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
268                     sql = "UPDATE daily_income SET owner_id=(SELECT id from owner  WHERE username = '" + username + "' A
269
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
278                     sql = "INSERT INTO daily_income (owner_id , date , day_income) VALUES((SELECT id from owner  WHERE u
279
280                     char* sqliteInsert = const_cast<char*>(sql.c_str());
281                     rc = sqlite3_exec(db, sqliteInsert, callback, 0, &zErrMsg);
282
283                 }
284
285                 if (rc != SQLITE_OK) {
286                     fprintf(stderr, "SQL error: %s\n", zErrMsg);
```

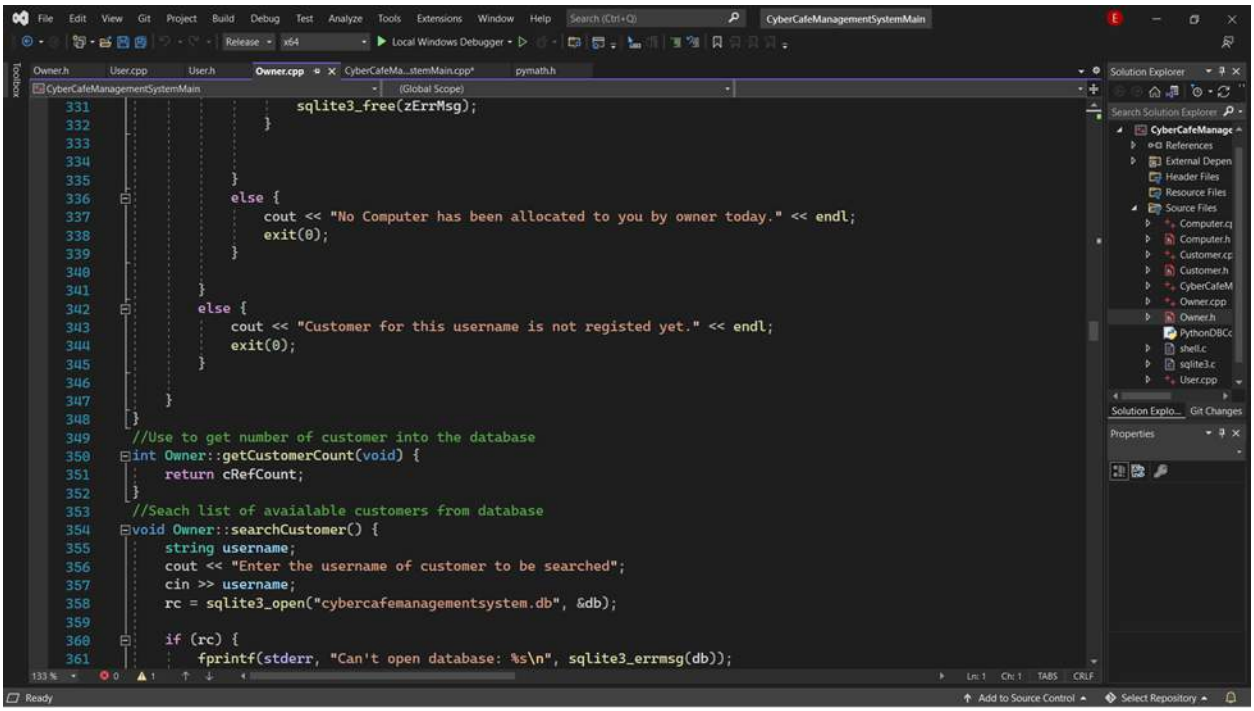


```
            fprintf(stderr, "SQL error: %s\n", zErrMsg);
            sqlite3_free(zErrMsg);
        }
        else {

            fprintf(stderr, "Daily income added to database successfully\n");

        }


    }
}

void Owner::checkChargeSessionTimeForCustomer() {
    rc = sqlite3_open("cybercafemanagementsystem.db", &db);

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


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

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

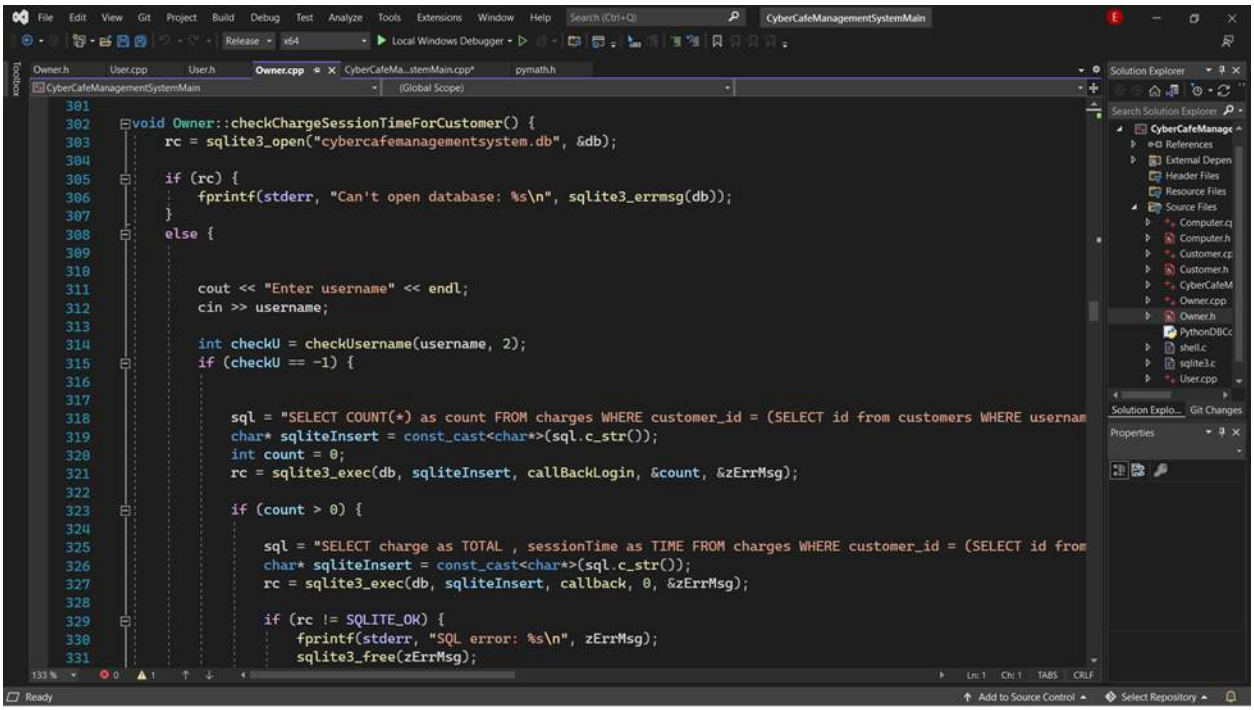


```
Owner Object Created
29-12-2021(Today's) total income so far is as follows.
TOTAL = 10.0

Enter your owner username to save daily income in database.
```

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

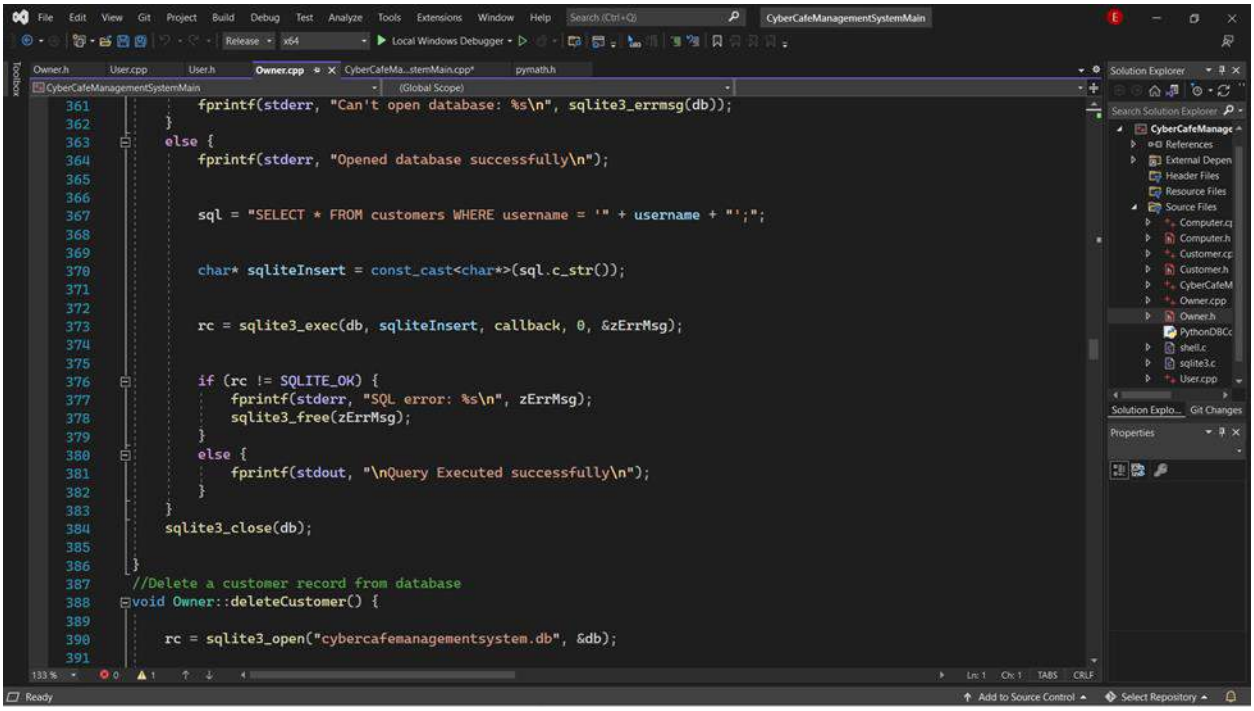


```
            sqlite3_free(zErrMsg);
        }


    }
    else {
        cout << "No Computer has been allocated to you by owner today." << endl;
        exit(0);
    }

}
else {
    cout << "Customer for this username is not registed yet." << endl;
    exit(0);
}

}
}
//Use to get number of customer into the database
int Owner::getCustomerCount(void) {
    return cRefCount;
}
//Seach list of avaialable customers from database
void Owner::searchCustomer() {
    string username;
    cout << "Enter the username of customer to be searched";
    cin >> username;
    rc = sqlite3_open("cybercafemanagementsystem.db", &db);

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

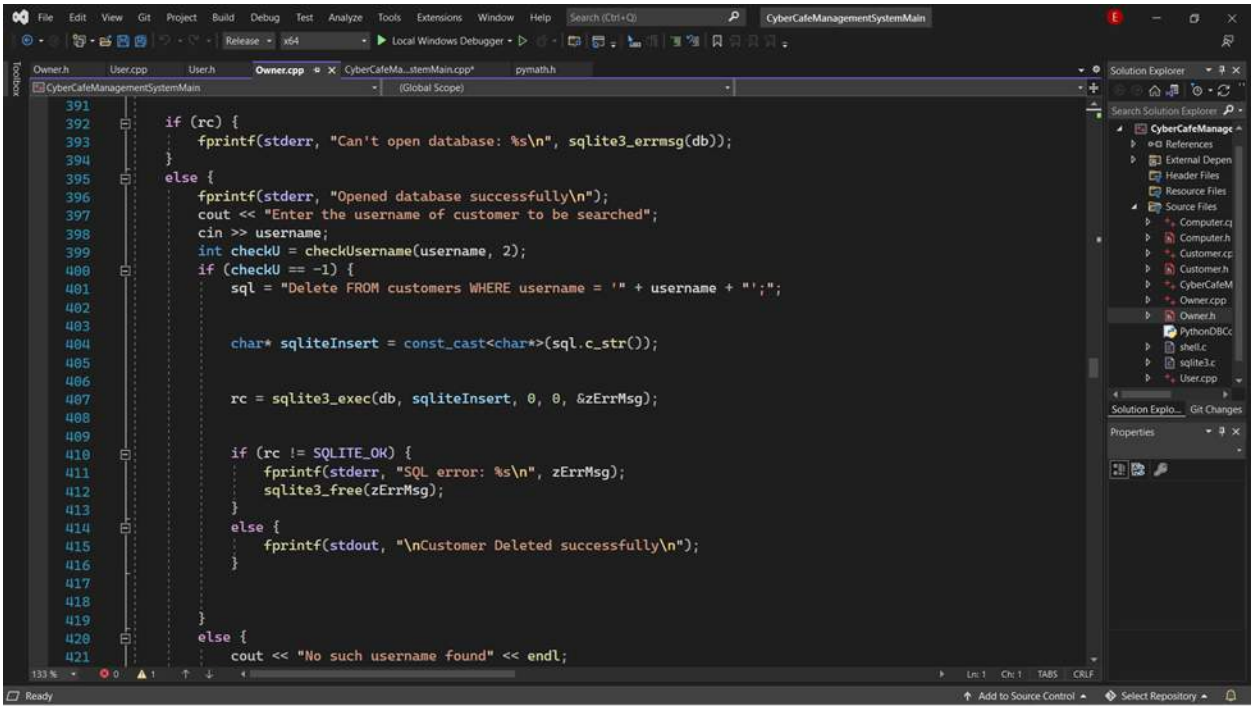


```
void Owner::checkChargeSessionTimeForCustomer() {
    rc = sqlite3_open("cybercafemanagementsystem.db", &db);

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


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

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


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

            if (count > 0) {

                sql = "SELECT charge as TOTAL , sessionTime as TIME FROM charges WHERE customer_id = (SELECT id from
                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);
```

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



```
            fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
        }
        else {
            fprintf(stderr, "Opened database successfully\n");


            sql = "SELECT * FROM customers WHERE username = '" + username + "';";


            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, "\nQuery Executed successfully\n");
            }
        }
        sqlite3_close(db);

    }
    //Delete a customer record from database
    void Owner::deleteCustomer() {

        rc = sqlite3_open("cybercafemanagementsystem.db", &db);
```



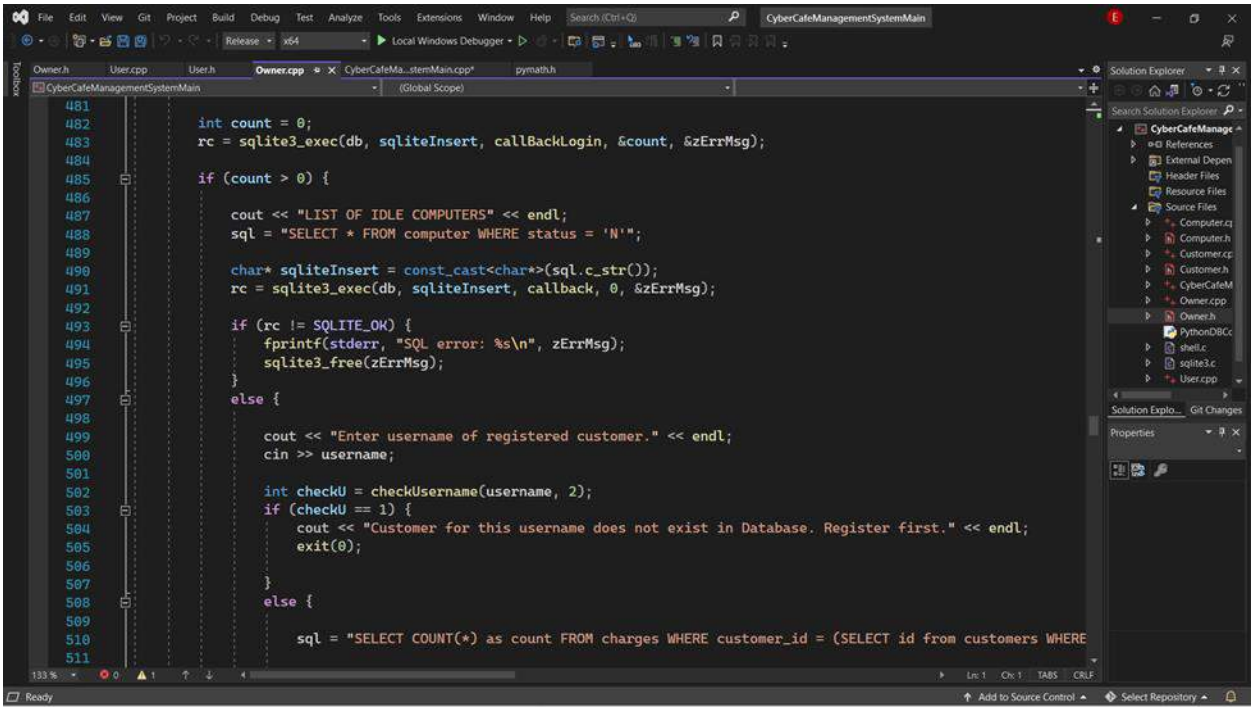
```
        if (rc) {
            fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
        }
        else {
            fprintf(stderr, "Opened database successfully\n");
            cout << "Enter the username of customer to be searched";
            cin >> username;
            int checkU = checkUsername(username, 2);
            if (checkU == -1) {
                sql = "Delete FROM customers WHERE username = '" + username + "';";


                char* sqliteInsert = const_cast<char*>(sql.c_str());


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


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


            }
            else {
                cout << "No such username found" << endl;
```

Function definition of delete customer by username from customer table.

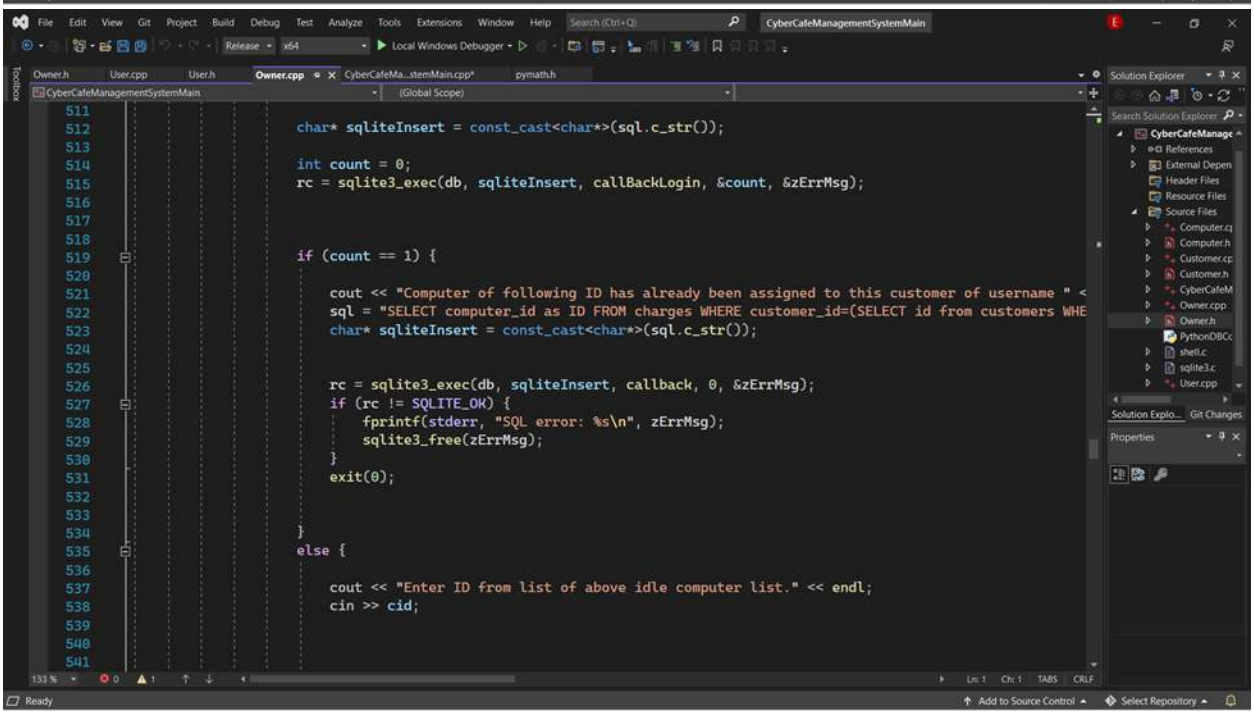
```
            cout << "No such username found" << endl;
            exit(0);


        }


    }
    sqlite3_close(db);

}
//Display whole data of customers came into the shop
void Owner::showCustomerList(void) {

    rc = sqlite3_open("cybercafemanagementsystem.db", &db);

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


        sql = "Select * FROM customers";


        char* sqliteInsert = const_cast<char*>(sql.c_str());


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


        if (rc != SQLITE_OK) {
```

Function definition of show list of all customers.

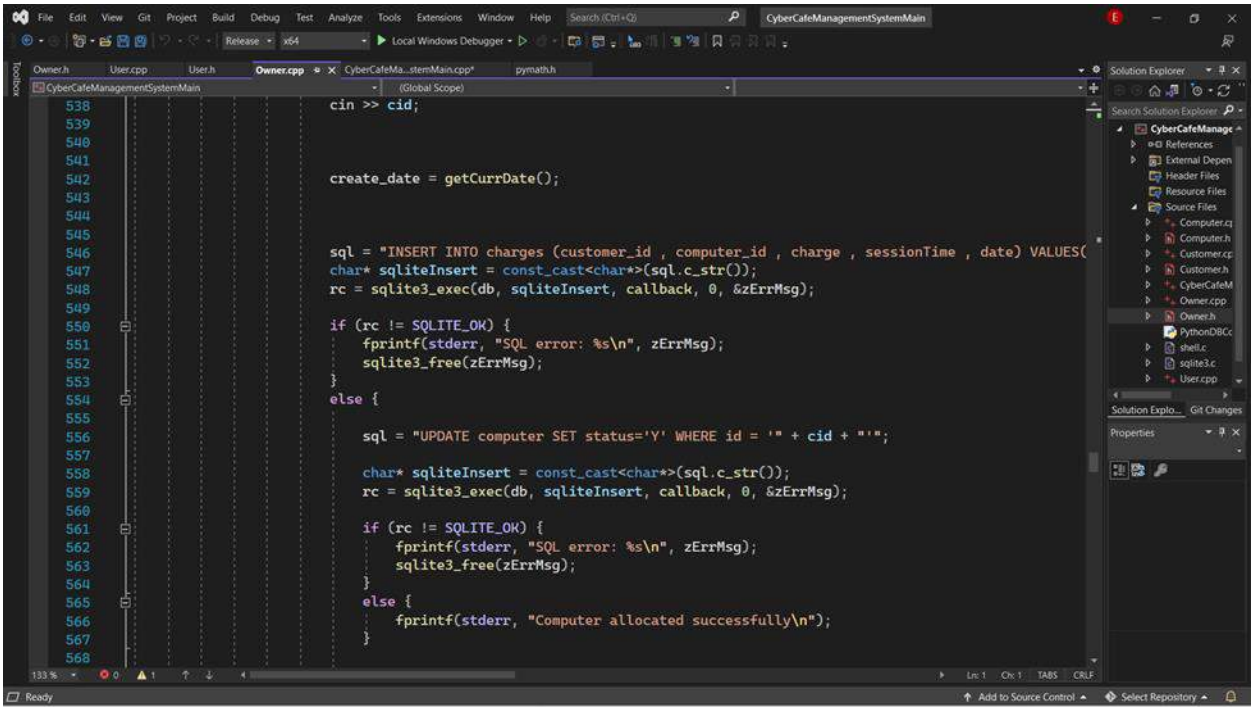
```
        if (rc != SQLITE_OK) {
            fprintf(stderr, "SQL error: %s\n", zErrMsg);
            sqlite3_free(zErrMsg);
        }
        else {
            fprintf(stdout, "\nQuery Executed successfully\n");
        }
    }
    sqlite3_close(db);

}

//Used to allocate a vacant computer to customer
void Owner::allocateComputer(void) {


    string cid;

    rc = sqlite3_open("cybercafemanagementsystem.db", &db);

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

        cout << "COMPUTER ALLOCATION" << endl;

        sql = "SELECT COUNT(*) as count FROM computer WHERE status = 'N'";

        char* sqliteInsert = const_cast<char*>(sql.c_str());
```



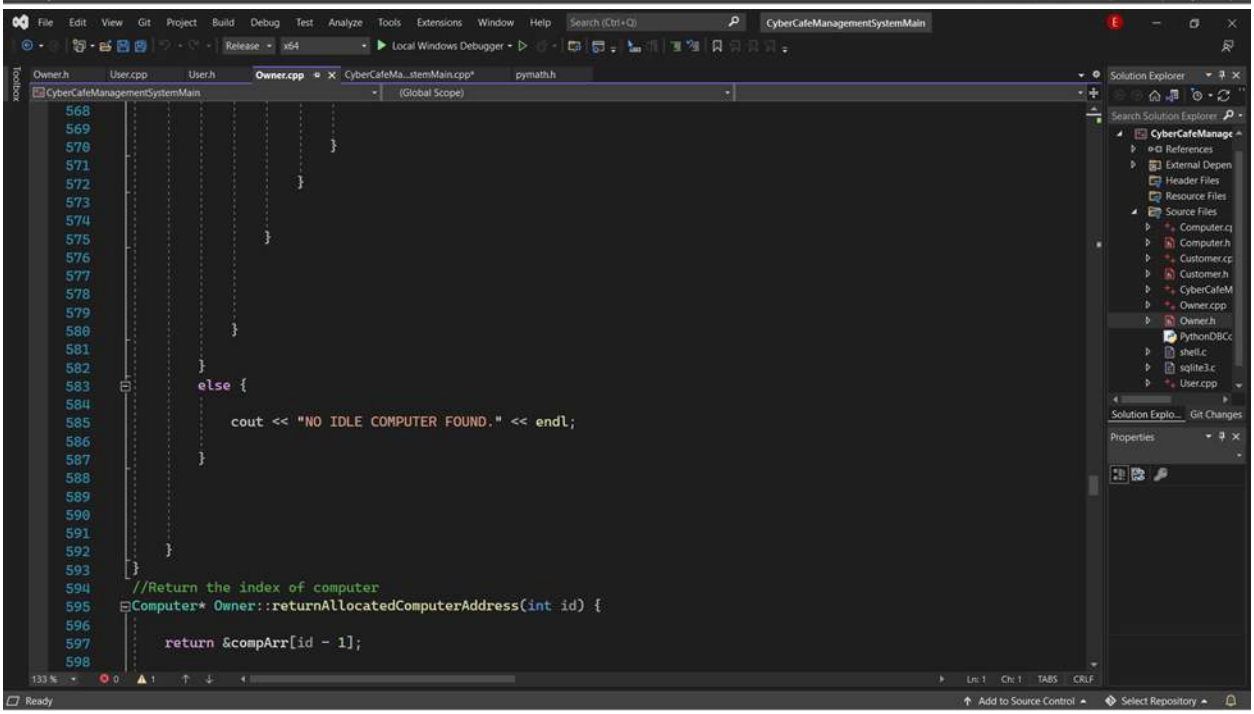
```
481
482         int count = 0;
483         rc = sqlite3_exec(db, sqliteInsert, callBackLogin, &count, &zErrMsg);
484
485         if (count > 0) {
486
487             cout << "LIST OF IDLE COMPUTERS" << endl;
488             sql = "SELECT * FROM computer WHERE status = 'N'";
489
490             char* sqliteInsert = const_cast<char*>(sql.c_str());
491             rc = sqlite3_exec(db, sqliteInsert, callback, 0, &zErrMsg);
492
493             if (rc != SQLITE_OK) {
494                 fprintf(stderr, "SQL error: %s\n", zErrMsg);
495                 sqlite3_free(zErrMsg);
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                 else {
509
510                     sql = "SELECT COUNT(*) as count FROM charges WHERE customer_id = (SELECT id from customers WHERE
511
```



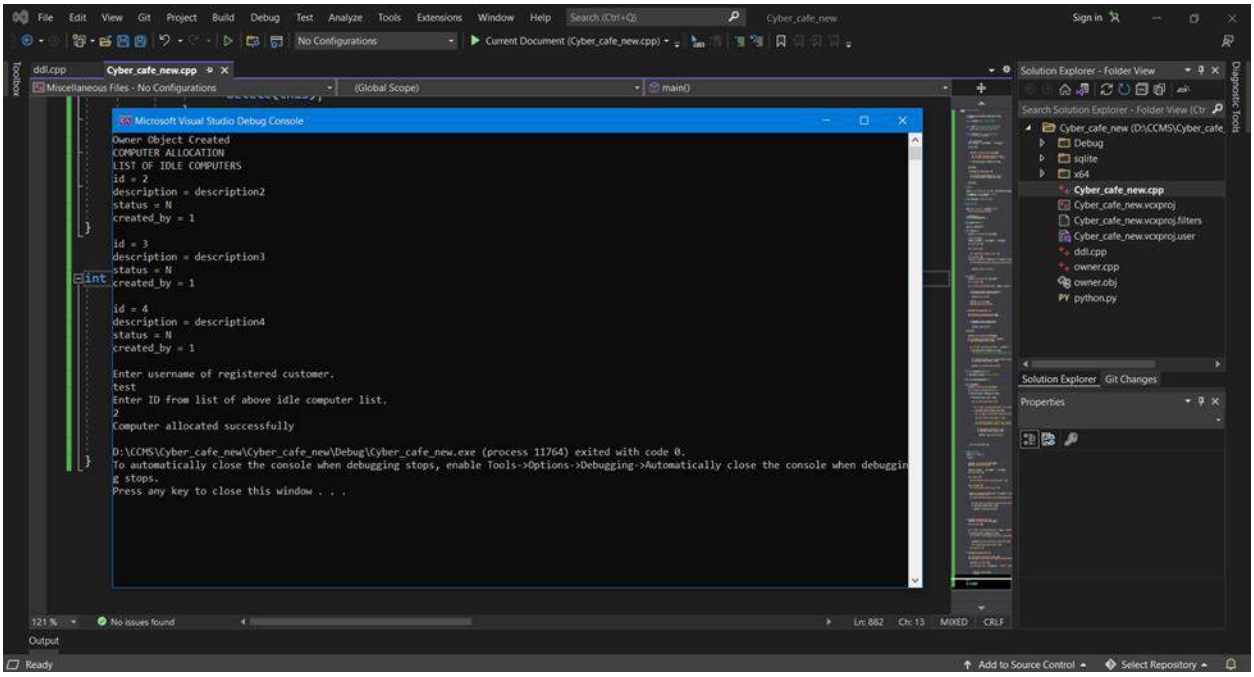
```
511
512                 char* sqliteInsert = const_cast<char*>(sql.c_str());
513
514                 int count = 0;
515                 rc = sqlite3_exec(db, sqliteInsert, callBackLogin, &count, &zErrMsg);
516
517
518
519                 if (count == 1) {
520
521                     cout << "Computer 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
```



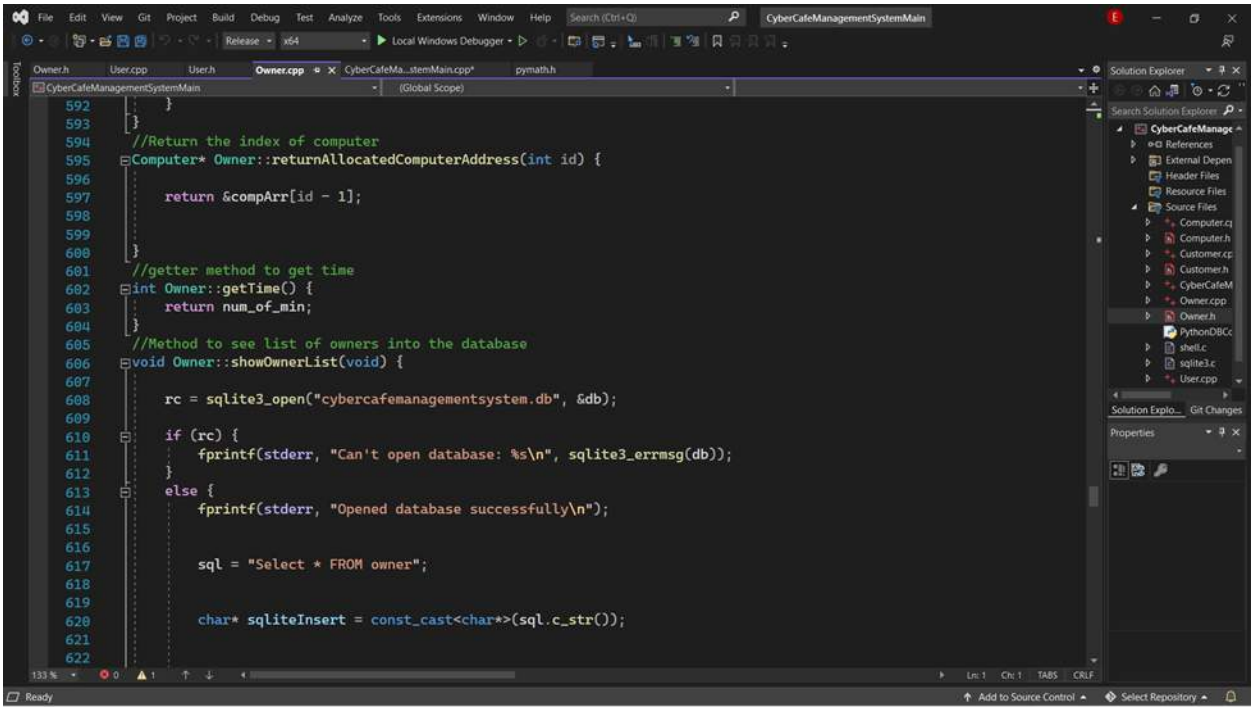
```
538                     cin >> cid;
539
540
541
542                     create_date = getCurrDate();
543
544
545
546                     sql = "INSERT INTO charges (customer_id , computer_id , charge , sessionTime , date) VALUES(
547                     char* sqliteInsert = const_cast<char*>(sql.c_str());
548                     rc = sqlite3_exec(db, sqliteInsert, callback, 0, &zErrMsg);
549
550                     if (rc != SQLITE_OK) {
551                         fprintf(stderr, "SQL error: %s\n", zErrMsg);
552                         sqlite3_free(zErrMsg);
553                     }
554                     else {
555
556                         sql = "UPDATE computer SET status='Y' WHERE id = '" + cid + "'";
557
558                         char* sqliteInsert = const_cast<char*>(sql.c_str());
559                         rc = sqlite3_exec(db, sqliteInsert, callback, 0, &zErrMsg);
560
561                         if (rc != SQLITE_OK) {
562                             fprintf(stderr, "SQL error: %s\n", zErrMsg);
563                             sqlite3_free(zErrMsg);
564                         }
565                         else {
566                             fprintf(stderr, "Computer allocated successfully\n");
567                         }
568
```



```
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
```



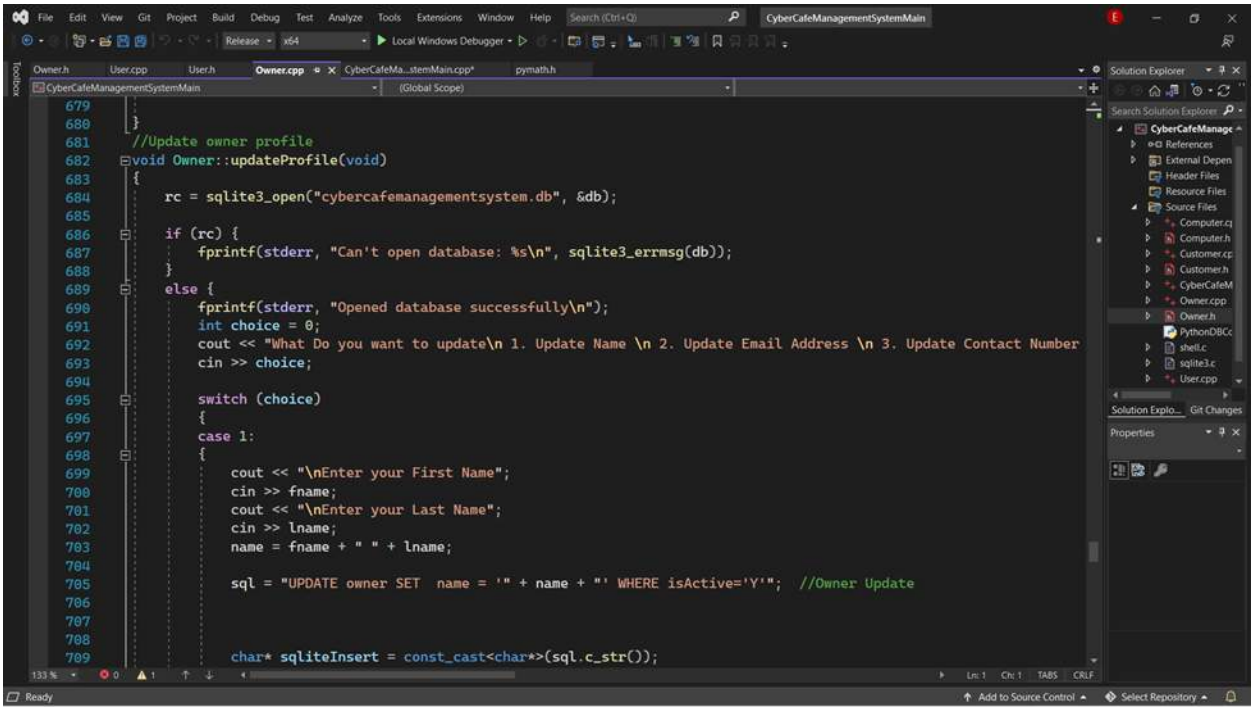
Function definition to allocate computers to registered customer from idle computer list.



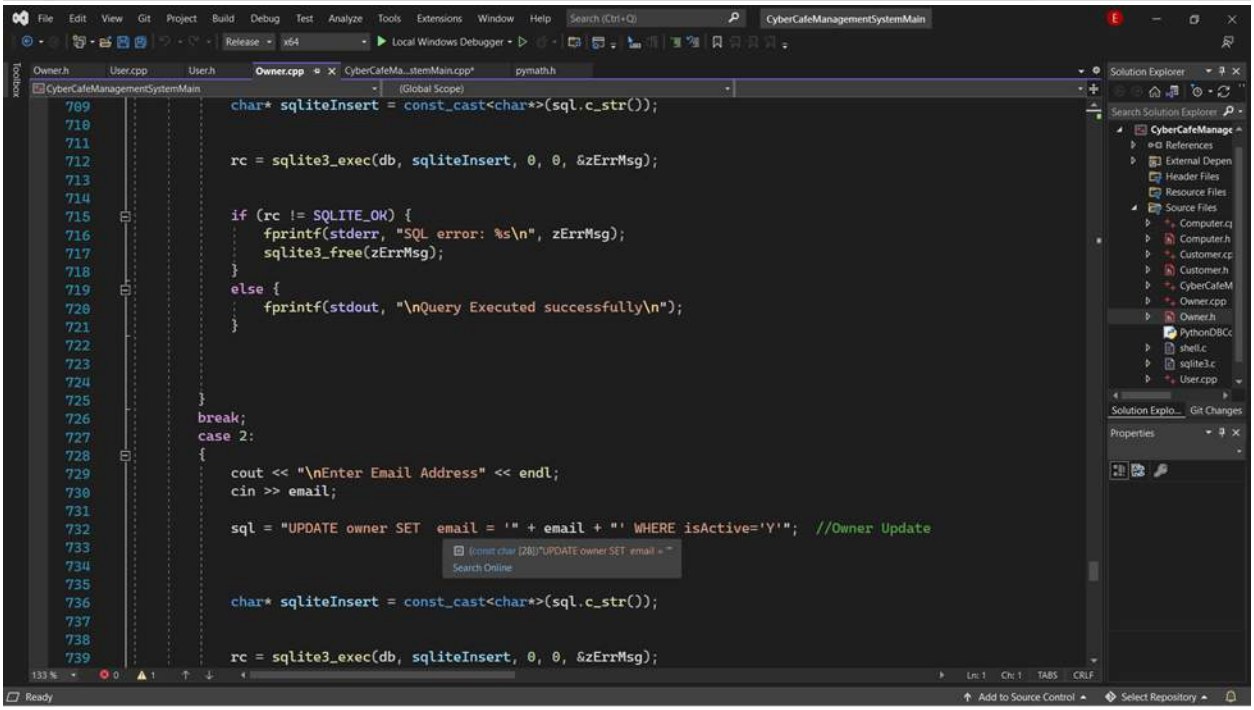
```
            char* sqliteInsert = const_cast<char*>(sql.c_str());


            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, "\nQuery Executed successfully\n");
            }
        }
        sqlite3_close(db);

    }
    //Delete owner from database

void Owner::deleteOwner() {

    rc = sqlite3_open("cybercafemanagementsystem.db", &db);

    if (rc) {
        fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
    }
    else {
        fprintf(stderr, "Opened database successfully\n");
        cout << "Enter the username of owner to be deleted";
        cin >> username;
```

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

```
        cin >> username;
        int checkU = checkUsername(username, 1);
        if (checkU == -1) {
            sql = "Delete FROM owner WHERE username = '" + username + "';";


            char* sqliteInsert = const_cast<char*>(sql.c_str());


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


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


        }
        else {
            cout << "No such username found" << endl;
            exit(0);

        }

    }
    sqlite3_close(db);
```



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

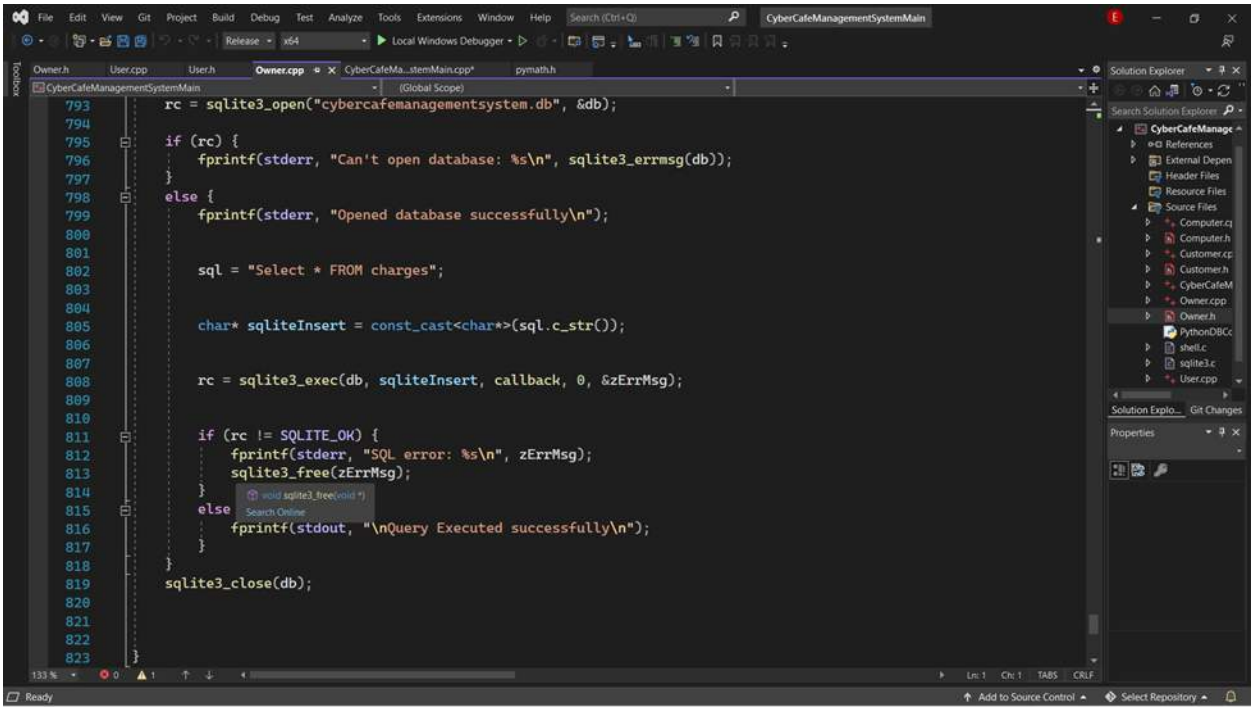


```
709                 char* sqliteInsert = const_cast<char*>(sql.c_str());
710
711
712                 rc = sqlite3_exec(db, sqliteInsert, 0, 0, &zErrMsg);
713
714
715                 if (rc != SQLITE_OK) {
716                     fprintf(stderr, "SQL error: %s\n", zErrMsg);
717                   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

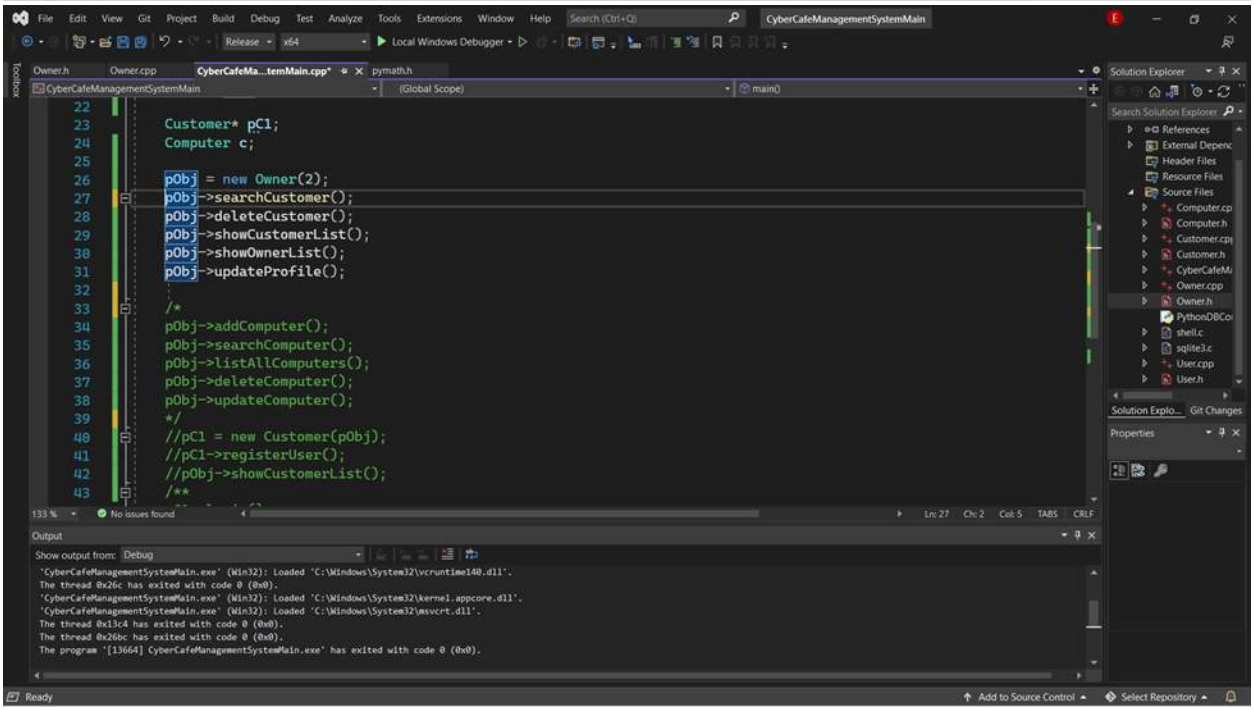
```
            rc = sqlite3_exec(db, sqliteInsert, 0, 0, &zErrMsg);


            if (rc != SQLITE_OK) {
                fprintf(stderr, "SQL error: %s\n", zErrMsg);
                sqlite3_free(zErrMsg);
            }
            else {
                fprintf(stdout, "\nQuery Executed successfully\n");
            }
        }
        break;
        case 3:
        {
            cout << "\nEnter your Contact Number";
            cin >> mobile_no;

            sql = "UPDATE owner SET  mobile_no = '" + mobile_no + "' WHERE isActive='Y'";  //Owner Update


            char* sqliteInsert = const_cast<char*>(sql.c_str());


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


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

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

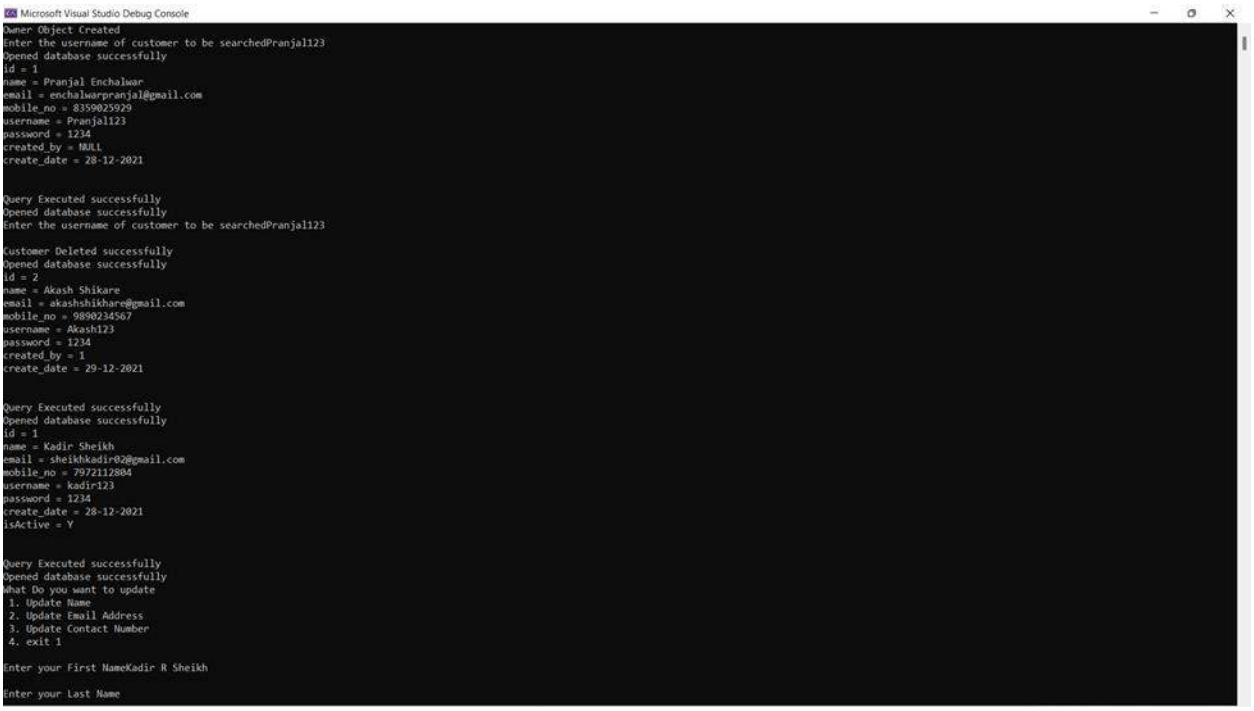
```
                sqlite3_free(zErrMsg);
            }
            else {
                fprintf(stdout, "\nQuery Executed successfully\n");
            }
        }
        break;
        case 4:
        {

        }




        sqlite3_close(db);

        }


    }
}
//Get all the charges paid by customers
void Owner::getAllCharges(void) {
    rc = sqlite3_open("cybercafemanagementsystem.db", &db);

    if (rc) {
        fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
    }
    else {
        fprintf(stderr, "Opened database successfully\n");
```



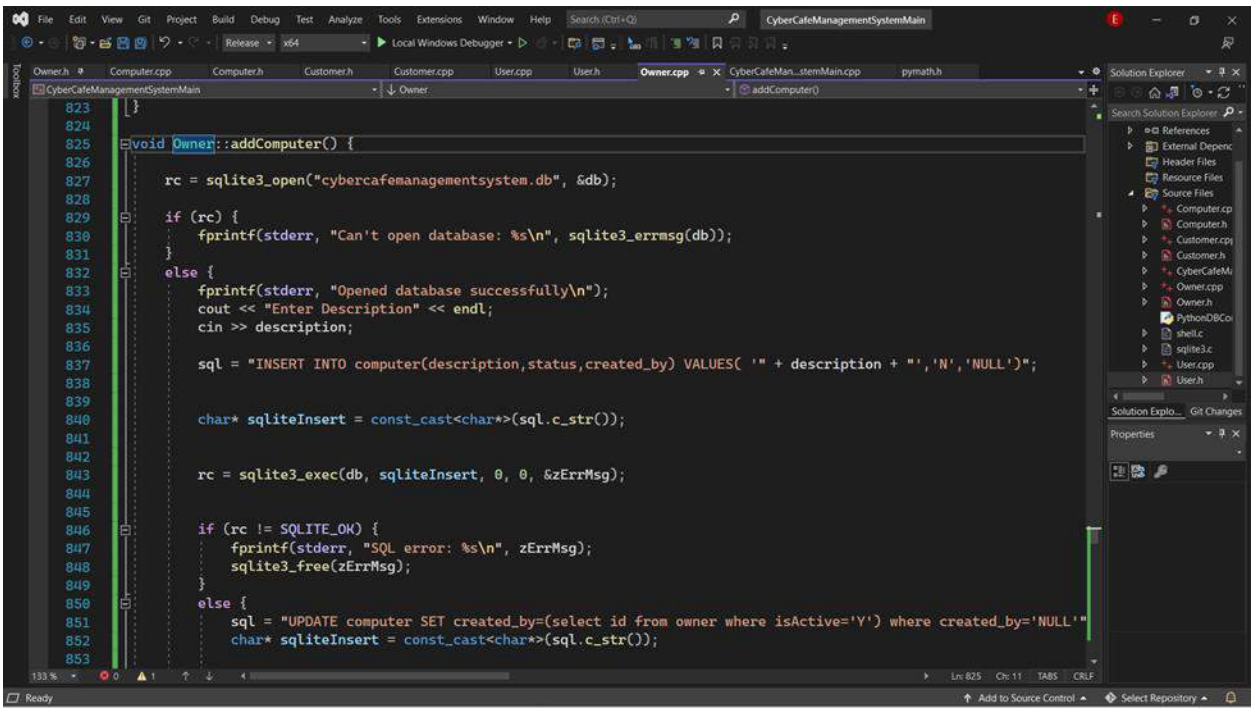
```
    rc = sqlite3_open("cybercafemanagementsystem.db", &db);

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


        sql = "Select * FROM charges";


        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, "\nQuery Executed successfully\n");
        }
    }
    sqlite3_close(db);
}
```

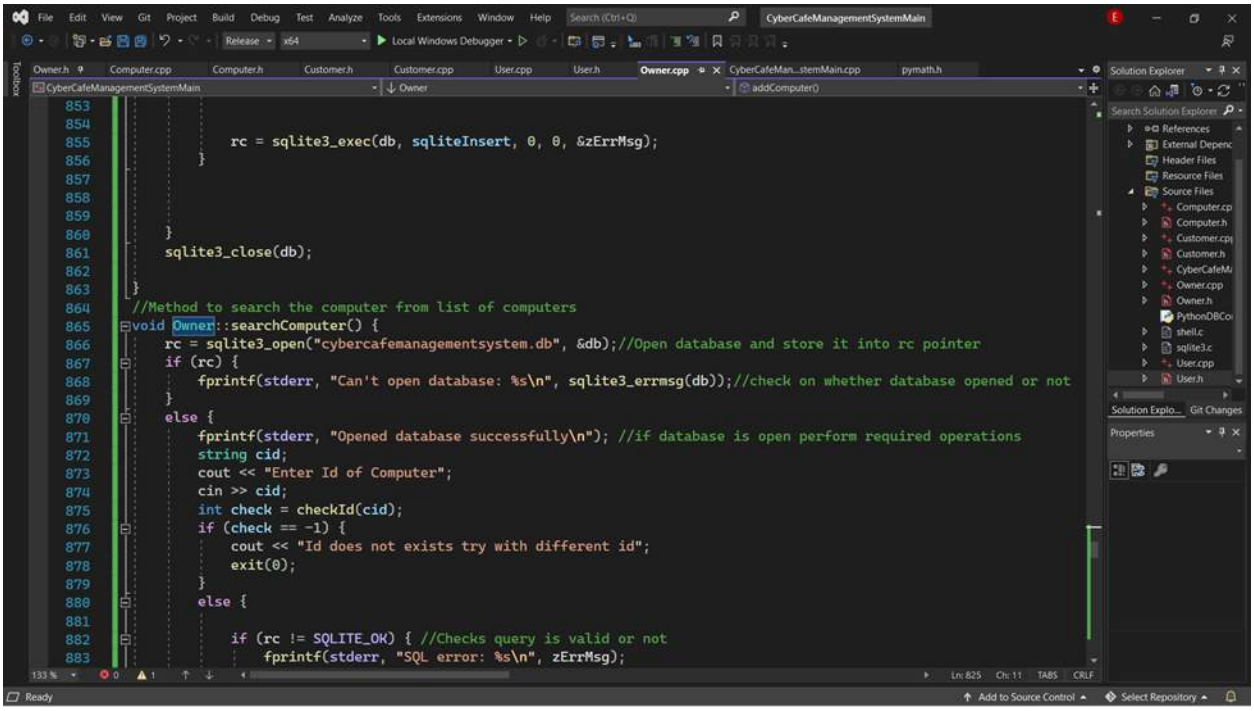


```
    Customer* pC1;
    Computer c;

    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();
    /**
```

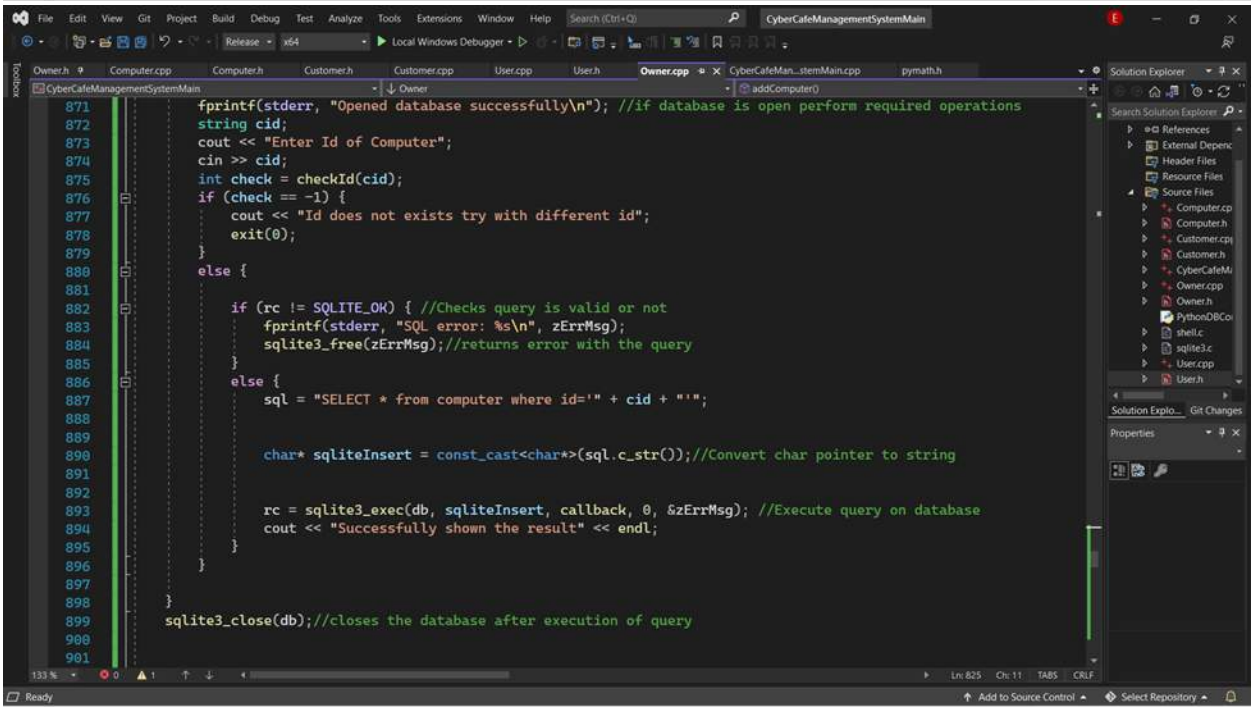


Function definition to get all charges which prints a charges table that shows entry of session time, charge for that session and the computer and customer id.

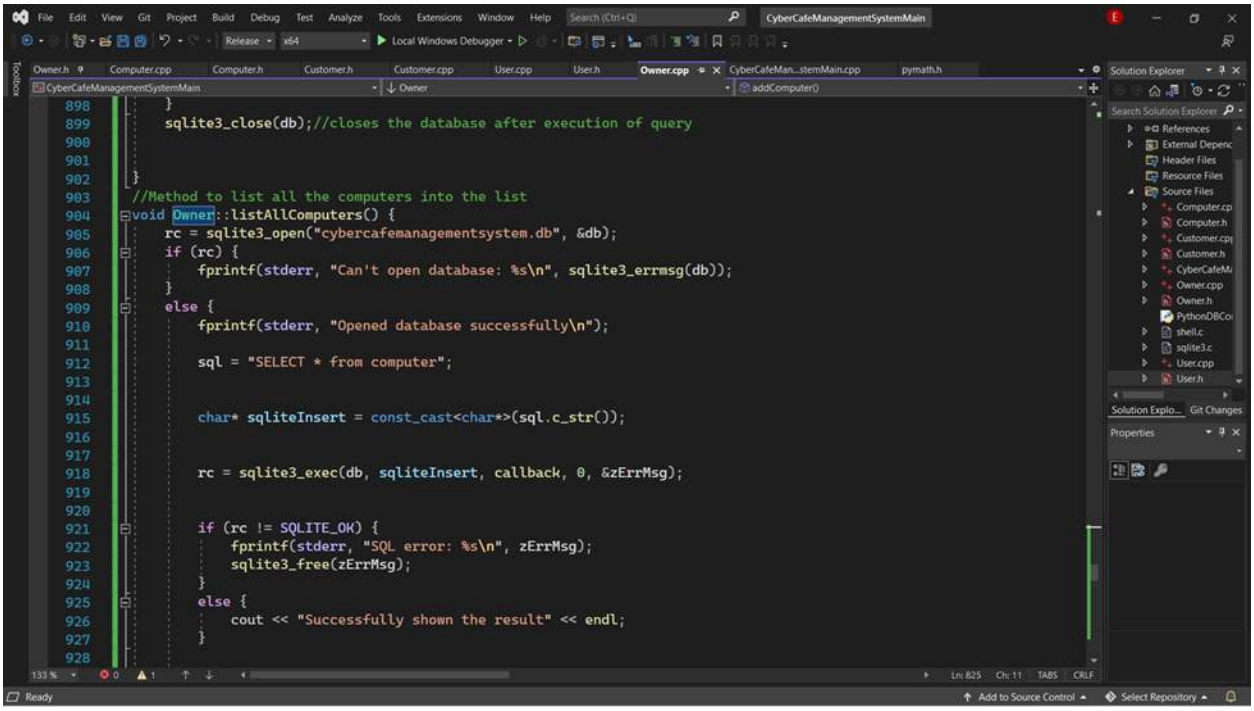




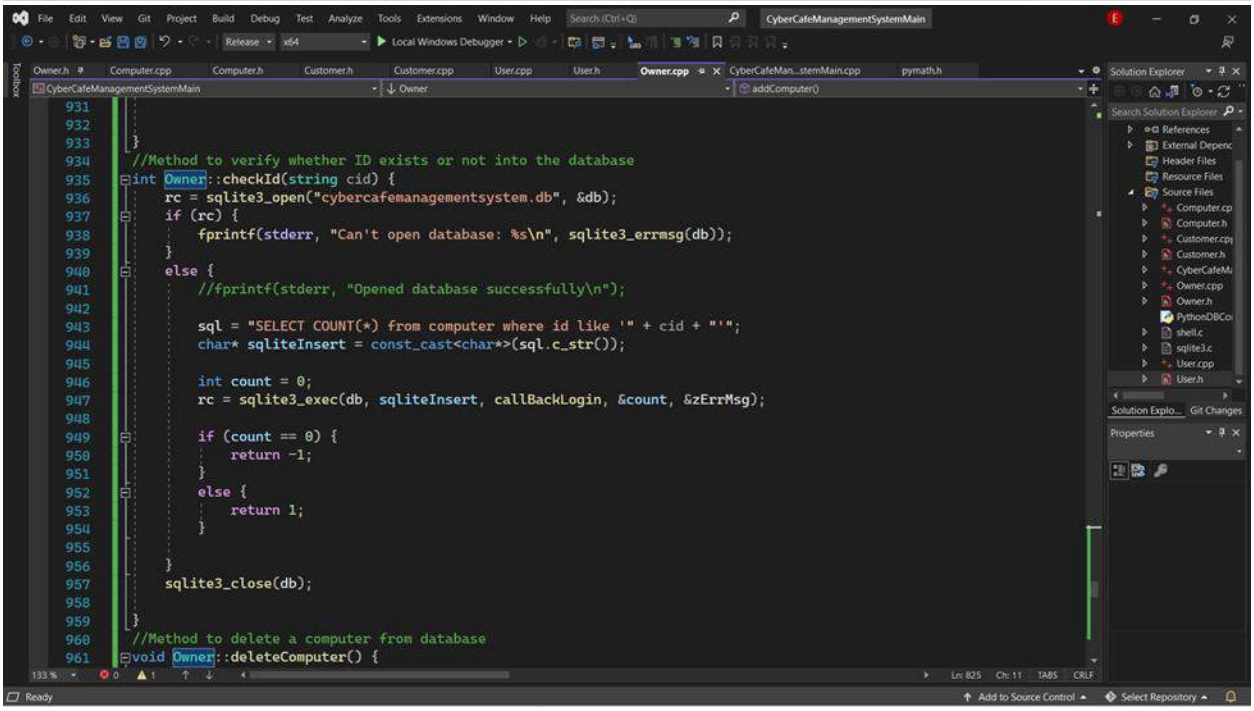
```
            rc = sqlite3_exec(db, sqliteInsert, 0, 0, &zErrMsg);
        }



    }
    sqlite3_close(db);

}
//Method to search the computer from list of computers
void Owner::searchComputer() {
    rc = sqlite3_open("cybercafemanagementsystem.db", &db);//Open database and store it into rc pointer
    if (rc) {
        fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));//check on whether database opened or not
    }
    else {
        fprintf(stderr, "Opened database successfully\n"); //if database is open perform required operations
        string cid;
        cout << "Enter Id of Computer";
        cin >> cid;
        int check = checkId(cid);
        if (check == -1) {
            cout << "Id does not exists try with different id";
            exit(0);
        }
        else {

            if (rc != SQLITE_OK) { //Checks query is valid or not
                fprintf(stderr, "SQL error: %s\n", zErrMsg);
```



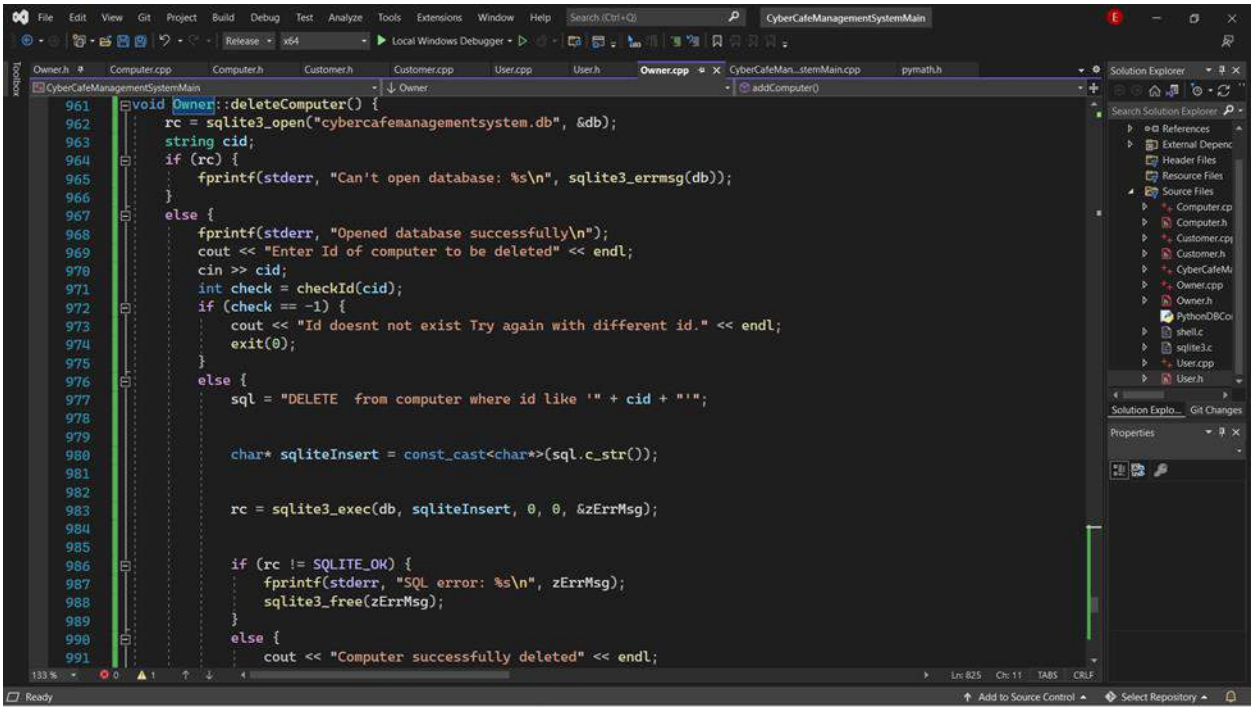
```
        fprintf(stderr, "Opened database successfully\n"); //if database is open perform required operations
        string cid;
        cout << "Enter Id of Computer";
        cin >> cid;
        int check = checkId(cid);
        if (check == -1) {
            cout << "Id does not exists try with different id";
            exit(0);
        }
        else {

            if (rc != SQLITE_OK) { //Checks query is valid or not
                fprintf(stderr, "SQL error: %s\n", zErrMsg);
                sqlite3_free(zErrMsg);//returns error with the query
            }
            else {
                sql = "SELECT * from computer where id='" + cid + "'";


                char* sqliteInsert = const_cast<char*>(sql.c_str());//Convert char pointer to string


                rc = sqlite3_exec(db, sqliteInsert, callback, 0, &zErrMsg); //Execute query on database
                cout << "Successfully shown the result" << endl;
            }
        }

    }
    sqlite3_close(db);//closes the database after execution of query
```



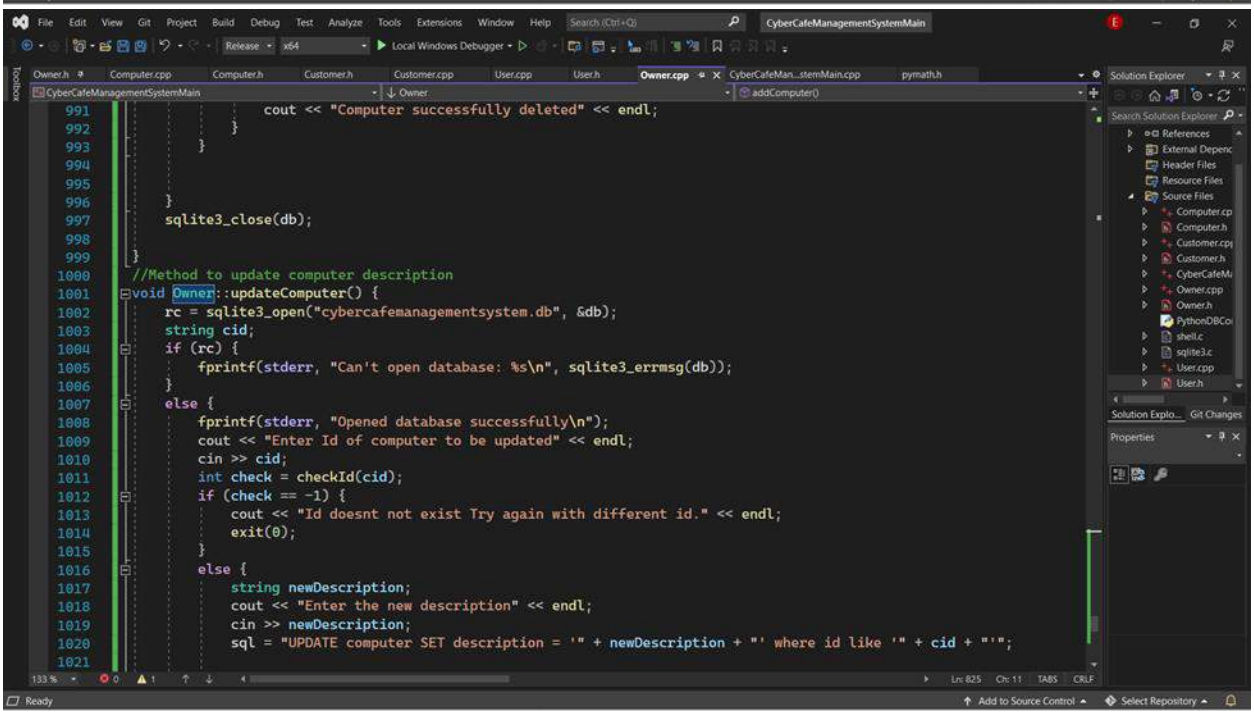
```
    }
    sqlite3_close(db);//closes the database after execution of query


}
//Method to list all the computers into the list
void Owner::listAllComputers() {
    rc = sqlite3_open("cybercafemanagementsystem.db", &db);
    if (rc) {
        fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
    }
    else {
        fprintf(stderr, "Opened database successfully\n");

        sql = "SELECT * from computer";


        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 {
            cout << "Successfully shown the result" << endl;
        }
```



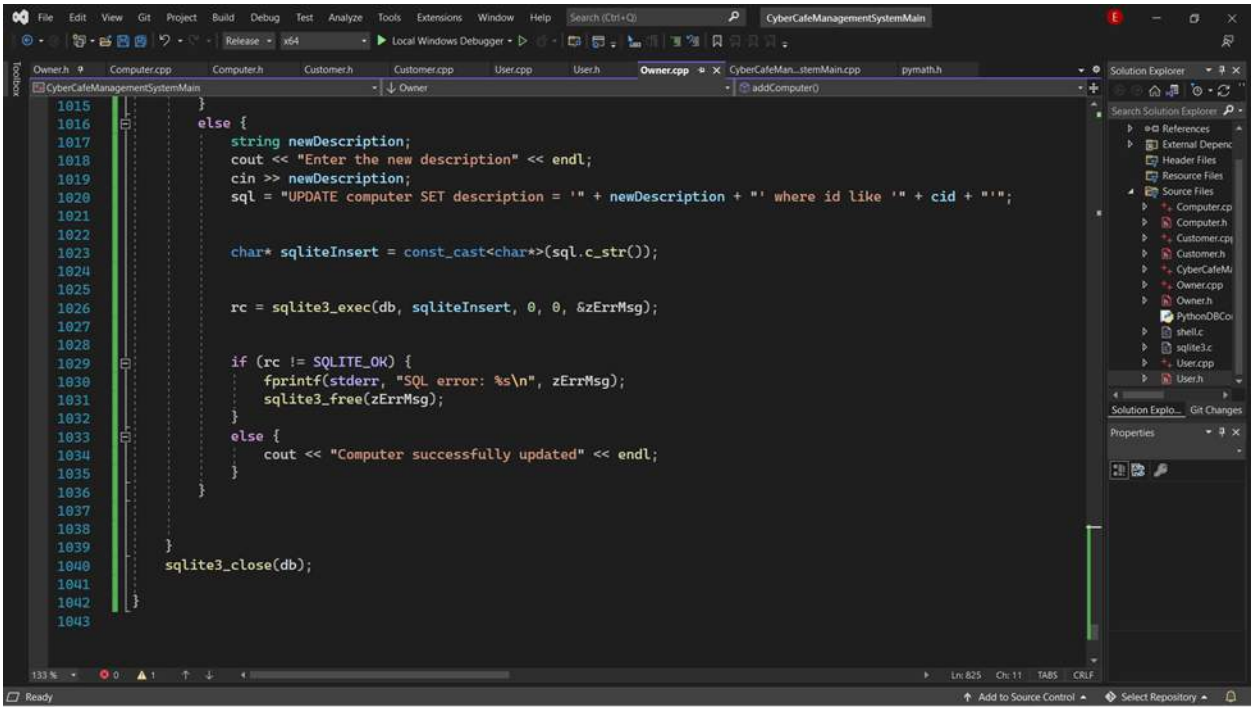
```


}
//Method to verify whether ID exists or not into the database
int Owner::checkId(string cid) {
    rc = sqlite3_open("cybercafemanagementsystem.db", &db);
    if (rc) {
        fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
    }
    else {
        //fprintf(stderr, "Opened database successfully\n");

        sql = "SELECT COUNT(*) from computer where id like '" + cid + "'";
        char* sqliteInsert = const_cast<char*>(sql.c_str());

        int count = 0;
        rc = sqlite3_exec(db, sqliteInsert, callBackLogin, &count, &zErrMsg);

        if (count == 0) {
            return -1;
        }
        else {
            return 1;
        }


    }
    sqlite3_close(db);


}
//Method to delete a computer from database
void Owner::deleteComputer() {
```



```
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 doesnt not exist Try again with different id." << endl;
974              exit(0);
975          }
976          else {
977              sql = "DELETE  from computer where id like '" + cid + "'";
978
979
980              char* sqliteInsert = const_cast<char*>(sql.c_str());
981
982
983              rc = sqlite3_exec(db, sqliteInsert, 0, 0, &zErrMsg);
984
985
986              if (rc != SQLITE_OK) {
987                  fprintf(stderr, "SQL error: %s\n", zErrMsg);
988                  sqlite3_free(zErrMsg);
989              }
990              else {
991                  cout << "Computer successfully deleted" << endl;
```

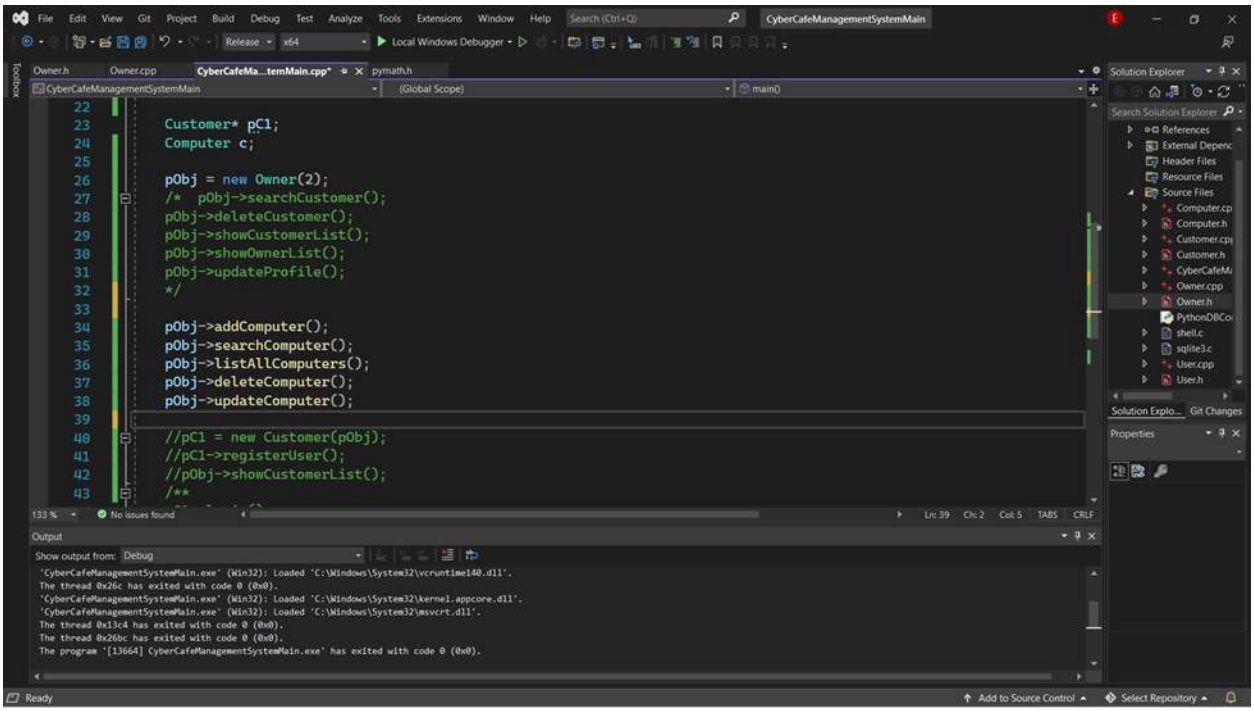


```
991                  cout << "Computer successfully deleted" << endl;
992              }
993          }
994
995
996      }
997      sqlite3_close(db);
998
999  }
1000 //Method to update computer description
1001 void Owner::updateComputer() {
1002     rc = sqlite3_open("cybercafemanagementsystem.db", &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 doesnt not 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
```



```
        }
        else {
            string newDescription;
            cout << "Enter the new description" << endl;
            cin >> newDescription;
            sql = "UPDATE computer SET description = '" + newDescription + "' where id like '" + cid + "'";


            char* sqliteInsert = const_cast<char*>(sql.c_str());


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


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


    }
    sqlite3_close(db);

}
```

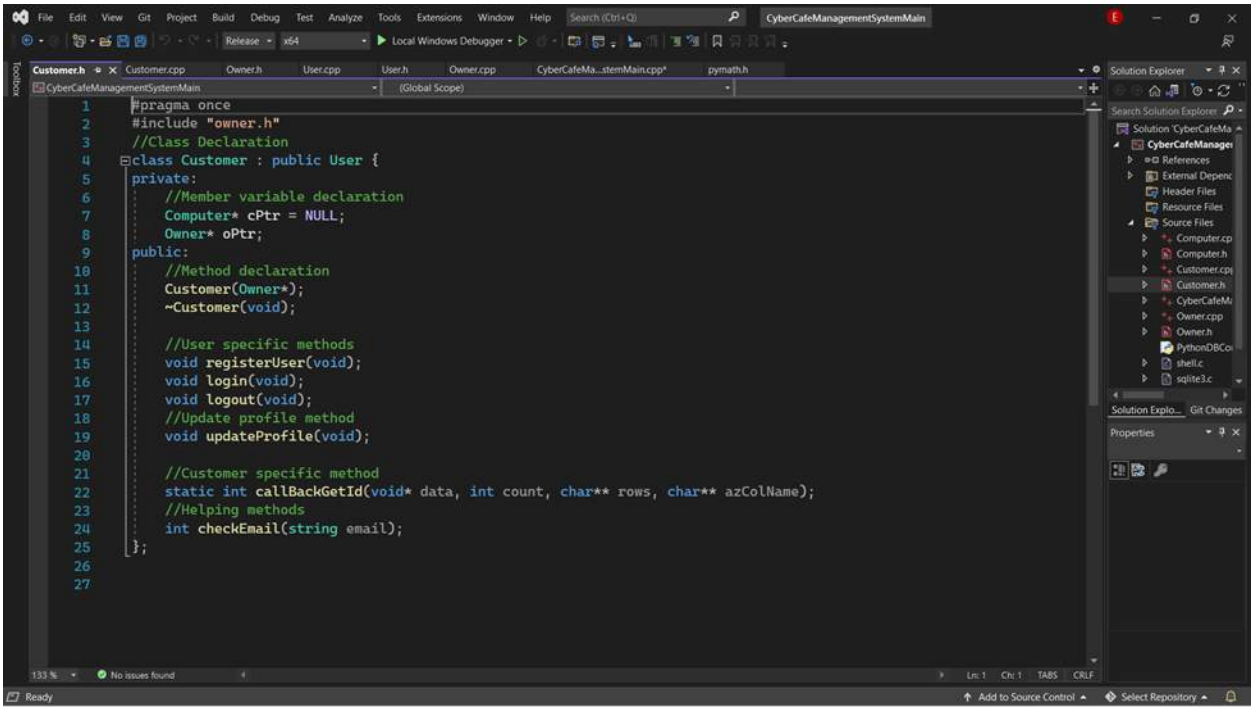
Execution


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();
/**
'CyberCafeManagementSystemMain.exe' (Win32): Loaded 'C:\Windows\System32\vcruntime140.dll'.
The thread 0x26c has exited with code 0 (0x0).
'CyberCafeManagementSystemMain.exe' (Win32): Loaded 'C:\Windows\System32\kernel.appcore.dll'.
'CyberCafeManagementSystemMain.exe' (Win32): Loaded 'C:\Windows\System32\msvcrt.dll'.
The thread 0x13c4 has exited with code 0 (0x0).
The thread 0x26bc has exited with code 0 (0x0).
The program '[13664] CyberCafeManagementSystemMain.exe' has exited with code 0 (0x0).


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

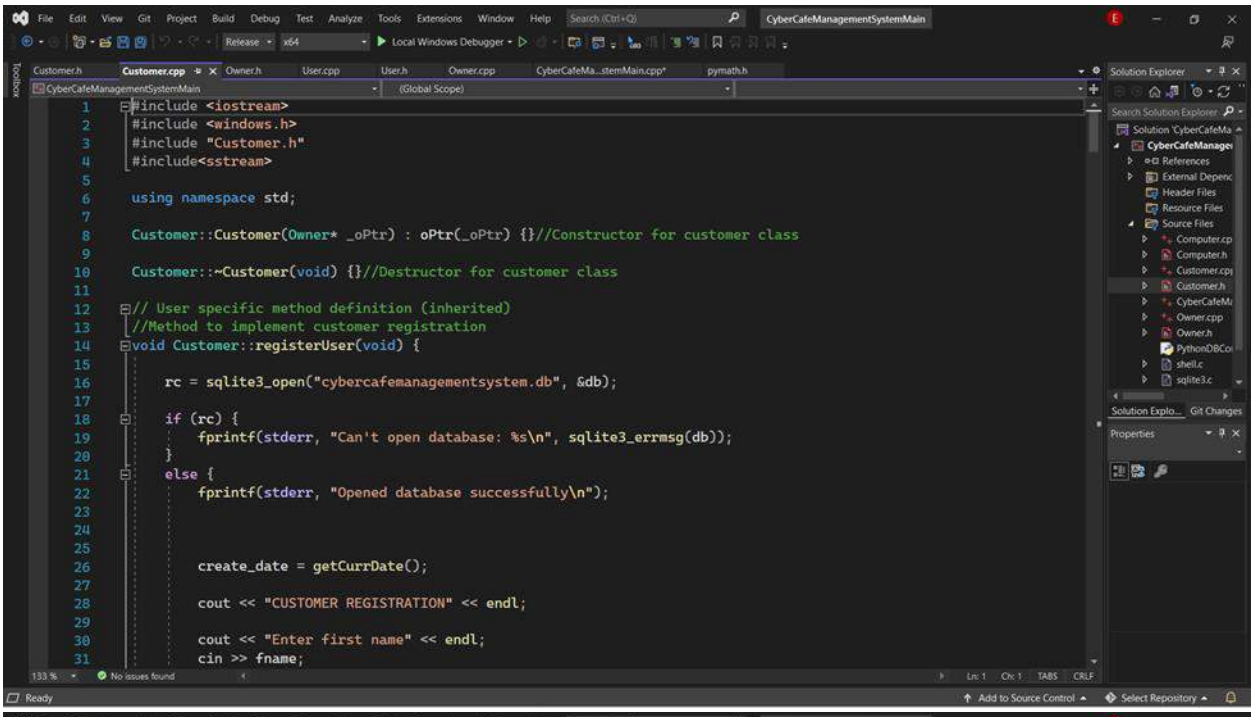
Class declaration



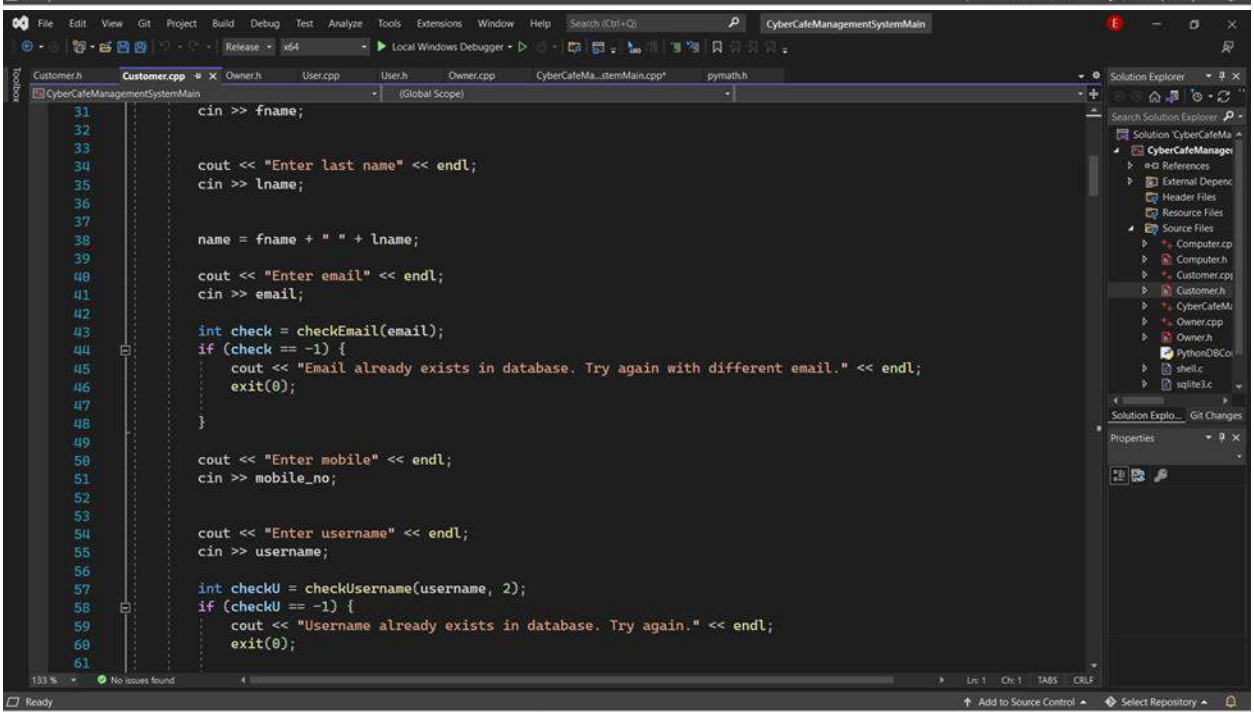
```
#pragma once
#include "owner.h"
//Class Declaration
class Customer : public User {
private:
    //Member variable declaration
    Computer* cPtr = NULL;
    Owner* oPtr;
public:
    //Method declaration
    Customer(Owner*);
    ~Customer(void);

    //User specific methods
    void registerUser(void);
    void login(void);
    void logout(void);
    //Update profile method
    void updateProfile(void);

    //Customer specific method
    static int callBackGetId(void* data, int count, char** rows, char** azColName);
    //Helping methods
    int checkEmail(string email);
};
```

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

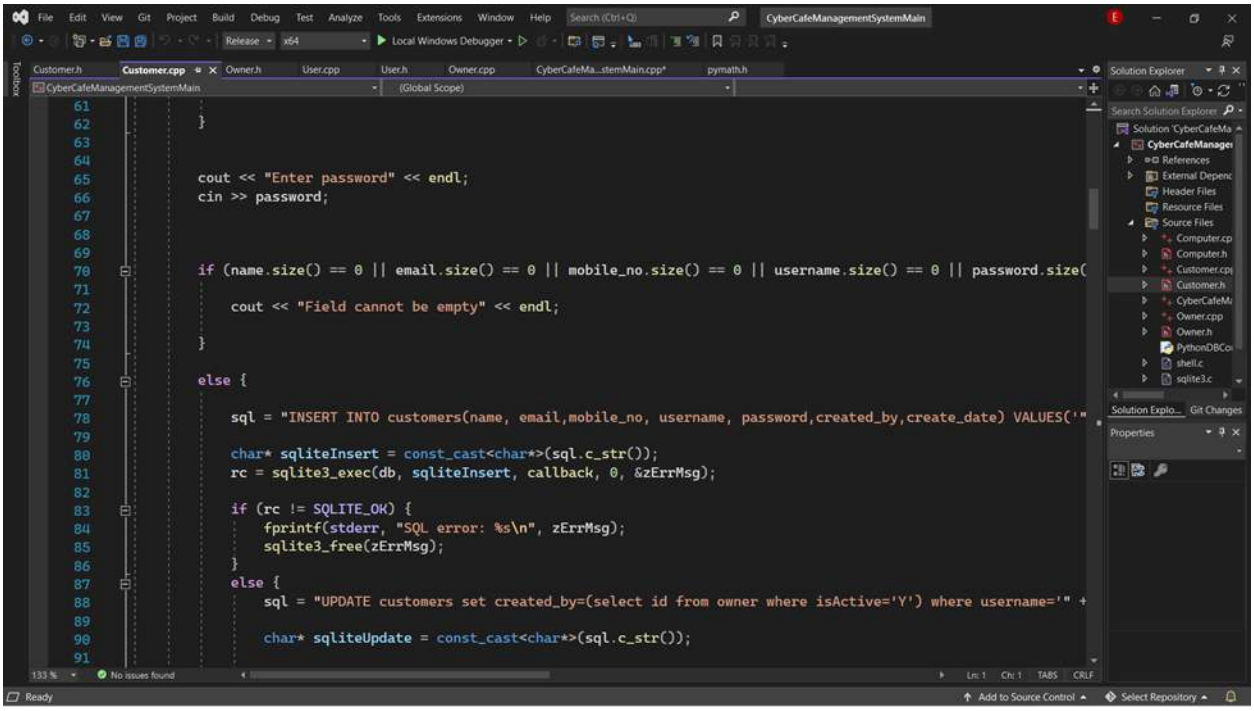
Class definition



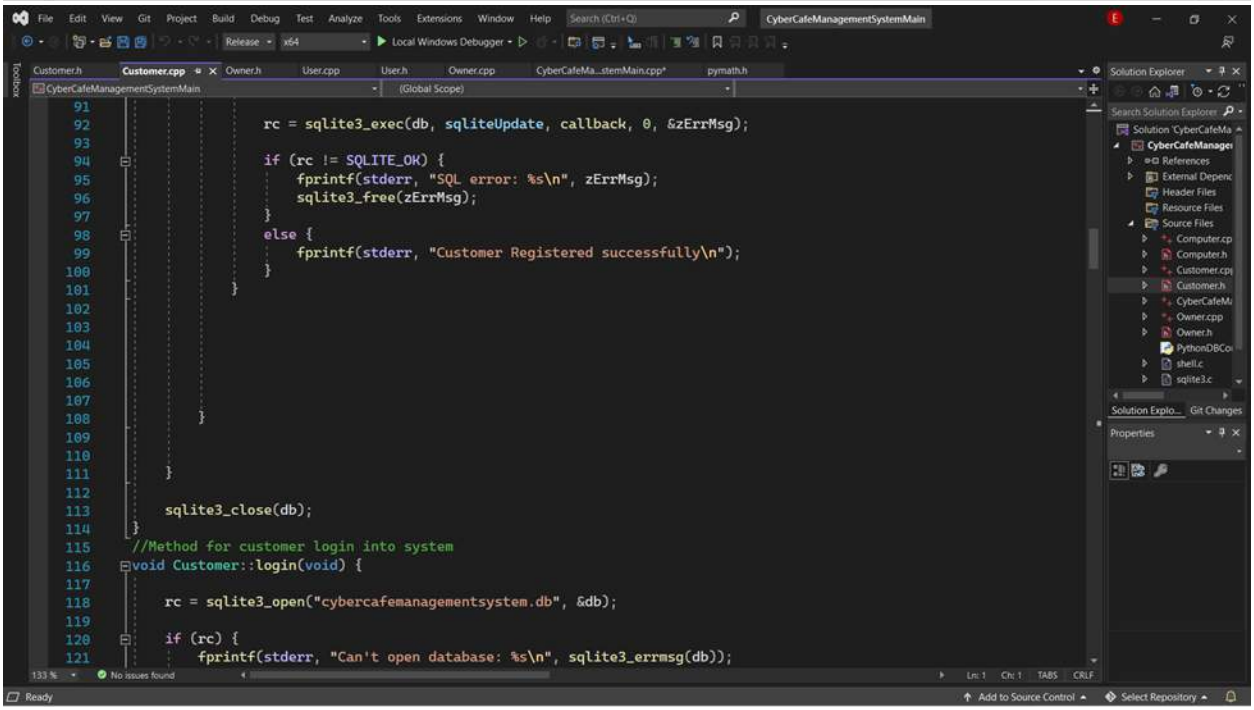
```cpp
#include <iostream>
#include <windows.h>
#include "Customer.h"
#include<sstream>

using namespace std;

Customer::Customer(Owner* _oPtr) : oPtr(_oPtr) {}//Constructor for customer class

Customer::~Customer(void) {}//Destructor for customer class

// User specific method definition (inherited)
//Method to implement customer registration
void Customer::registerUser(void) {

    rc = sqlite3_open("cybercafemanagementsystem.db", &db);

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



        create_date = getCurrDate();

        cout << "CUSTOMER REGISTRATION" << endl;

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



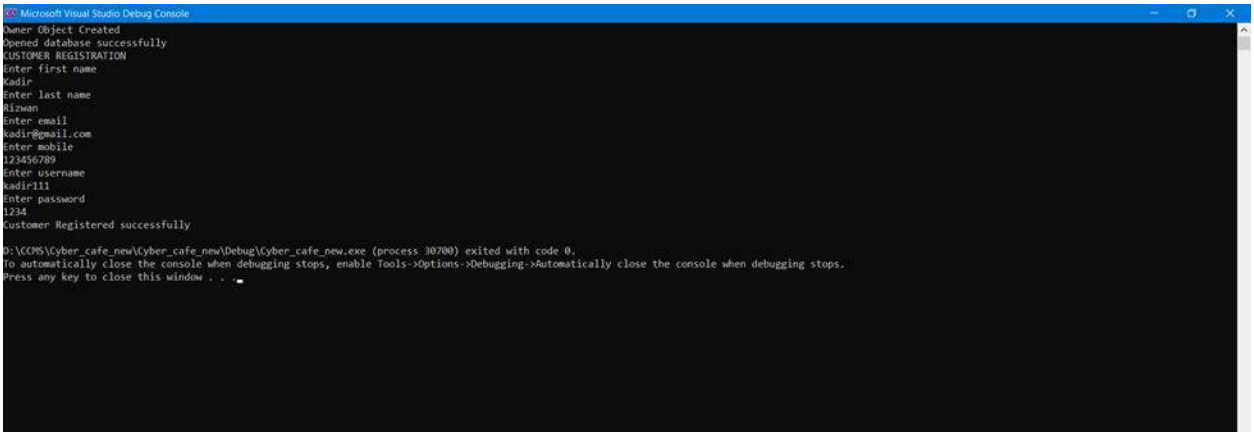
```cpp
        cin >> fname;


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


        name = fname + " " + lname;

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

        int check = checkEmail(email);
        if (check == -1) {
            cout << "Email already exists in database. Try again with different email." << endl;
            exit(0);

        }

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


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

        int checkU = checkUsername(username, 2);
        if (checkU == -1) {
            cout << "Username already exists in database. Try again." << endl;
            exit(0);
```



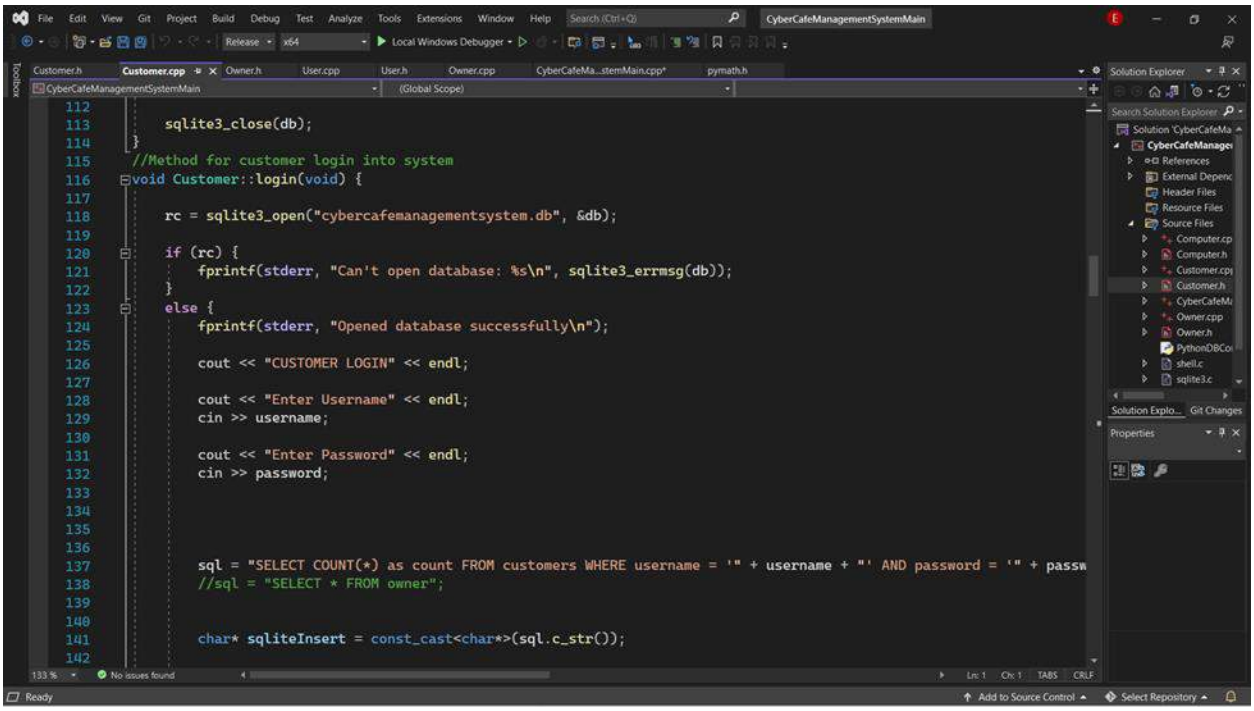
```
61          }
62
63
64          cout << "Enter password" << endl;
65          cin >> password;
66
67
68
69          if (name.size() == 0 || email.size() == 0 || mobile_no.size() == 0 || username.size() == 0 || password.size(
70
71              cout << "Field cannot be empty" << endl;
72
73          }
74
75          else {
76
77              sql = "INSERT INTO customers(name, email,mobile_no, username, password,created_by,create_date) VALUES('"
78
79              char* sqliteInsert = const_cast<char*>(sql.c_str());
80              rc = sqlite3_exec(db, sqliteInsert, callback, 0, &zErrMsg);
81
82              if (rc != SQLITE_OK) {
83                  fprintf(stderr, "SQL error: %s\n", zErrMsg);
84                  sqlite3_free(zErrMsg);
85              }
86              else {
87                  sql = "UPDATE customers set created_by=(select id from owner where isActive='Y') where username='" +
88
89                  char* sqliteUpdate = const_cast<char*>(sql.c_str());
90
91
```

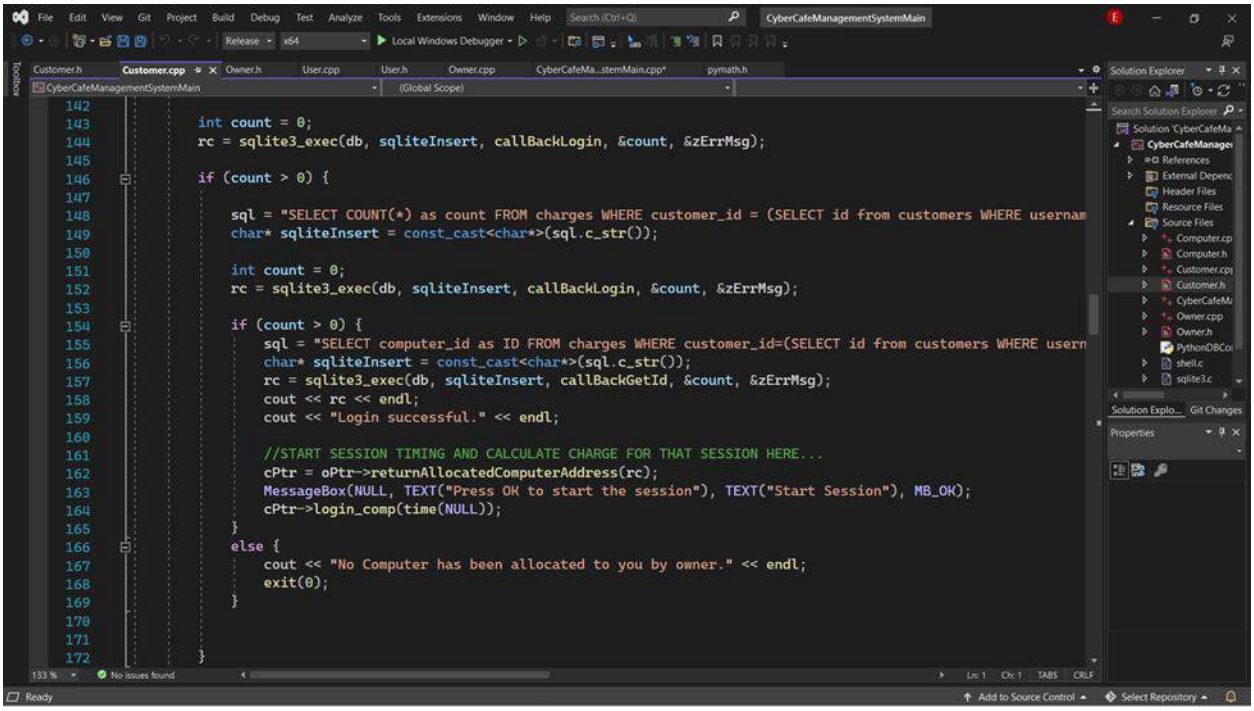


```
91
92                  rc = sqlite3_exec(db, sqliteUpdate, callback, 0, &zErrMsg);
93
94                  if (rc != SQLITE_OK) {
95                      fprintf(stderr, "SQL error: %s\n", zErrMsg);
96                      sqlite3_free(zErrMsg);
97                  }
98                  else {
99                      fprintf(stderr, "Customer Registered successfully\n");
100                 }
101             }
102
103
104
105
106
107
108         }
109
110
111     }
112
113     sqlite3_close(db);
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));
```

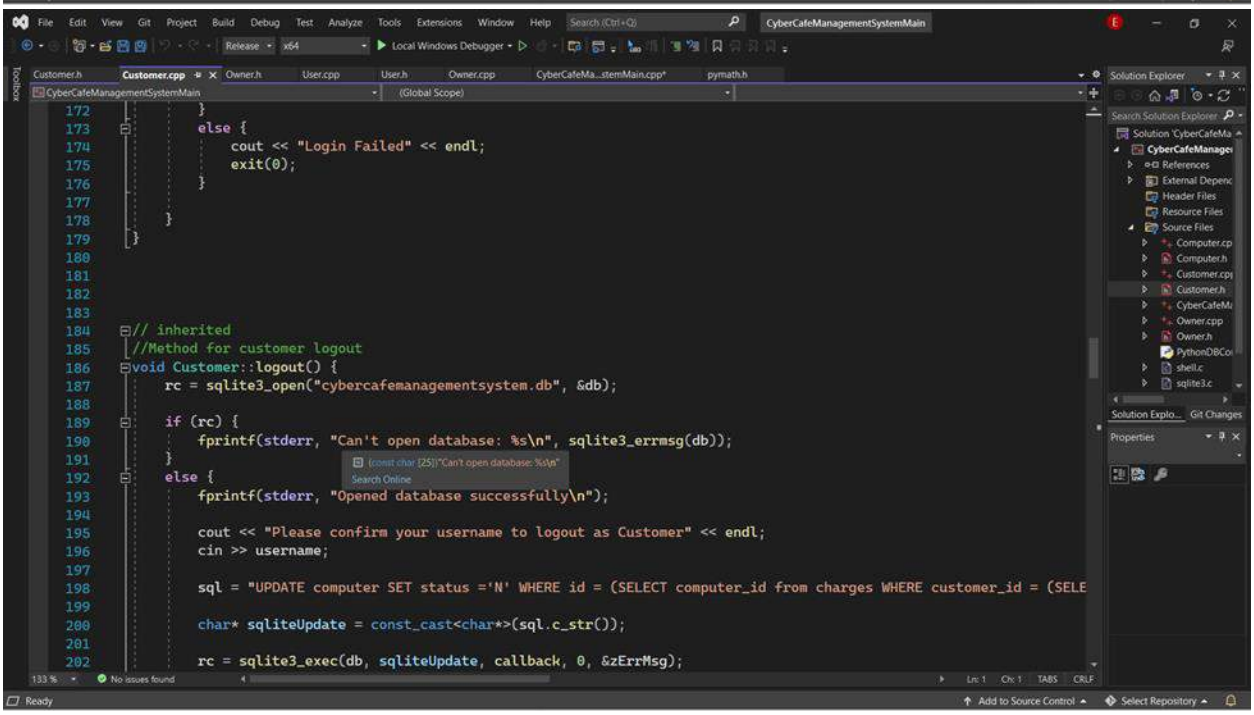


Function definition for customer registration which is overriding User's class virtual function.



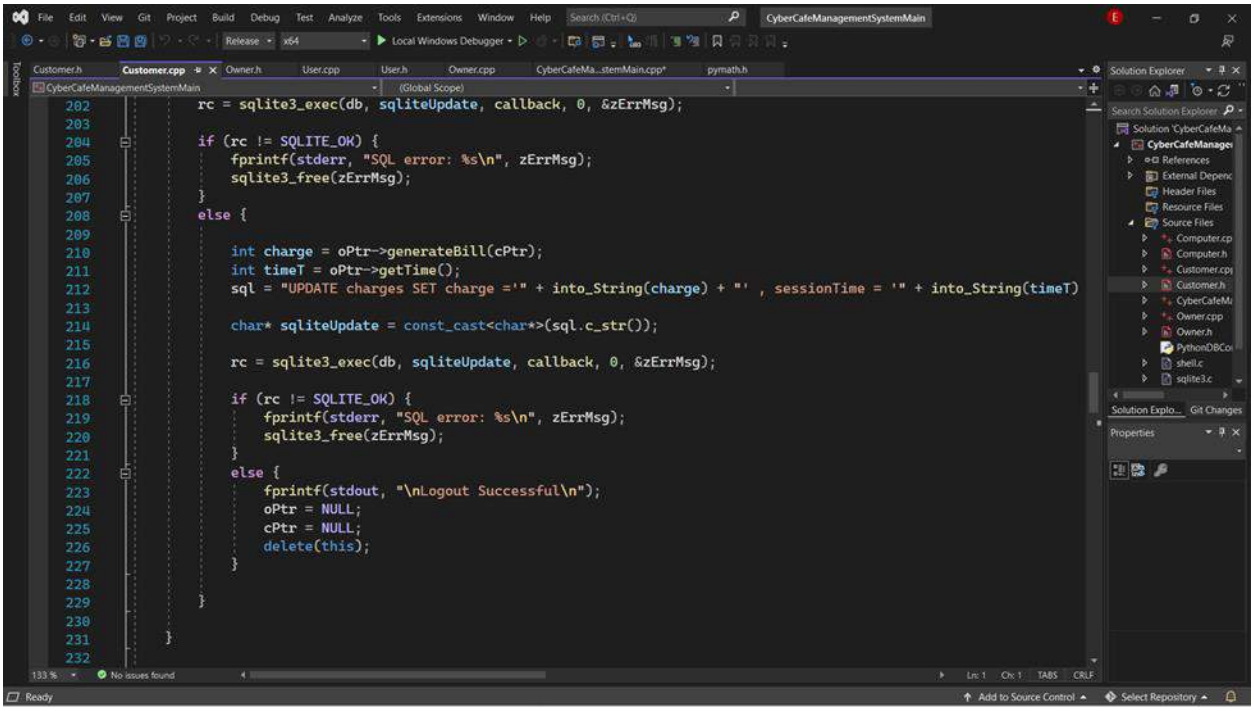


```
142
143             int count = 0;
144             rc = sqlite3_exec(db, sqliteInsert, callBackLogin, &count, &zErrMsg);
145
146             if (count > 0) {
147
148                 sql = "SELECT COUNT(*) as count FROM charges WHERE customer_id = (SELECT id from customers WHERE usernam
149                 char* sqliteInsert = const_cast<char*>(sql.c_str());
150
151                 int count = 0;
152                 rc = sqlite3_exec(db, sqliteInsert, callBackLogin, &count, &zErrMsg);
153
154                 if (count > 0) {
155                     sql = "SELECT computer_id as ID FROM charges WHERE customer_id=(SELECT id from customers WHERE usern
156                     char* sqliteInsert = const_cast<char*>(sql.c_str());
157                     rc = sqlite3_exec(db, sqliteInsert, callBackGetId, &count, &zErrMsg);
158                     cout << rc << endl;
159                     cout << "Login successful." << endl;
160
161                     //START SESSION TIMING AND CALCULATE CHARGE FOR THAT SESSION HERE...
162                     cPtr = oPtr->returnAllocatedComputerAddress(rc);
163                     MessageBox(NULL, TEXT("Press OK to start the session"), TEXT("Start Session"), MB_OK);
164                     cPtr->login_comp(time(NULL));
165                 }
166                 else {
167                     cout << "No Computer has been allocated to you by owner." << endl;
168                     exit(0);
169                 }
170
171
172             }
```

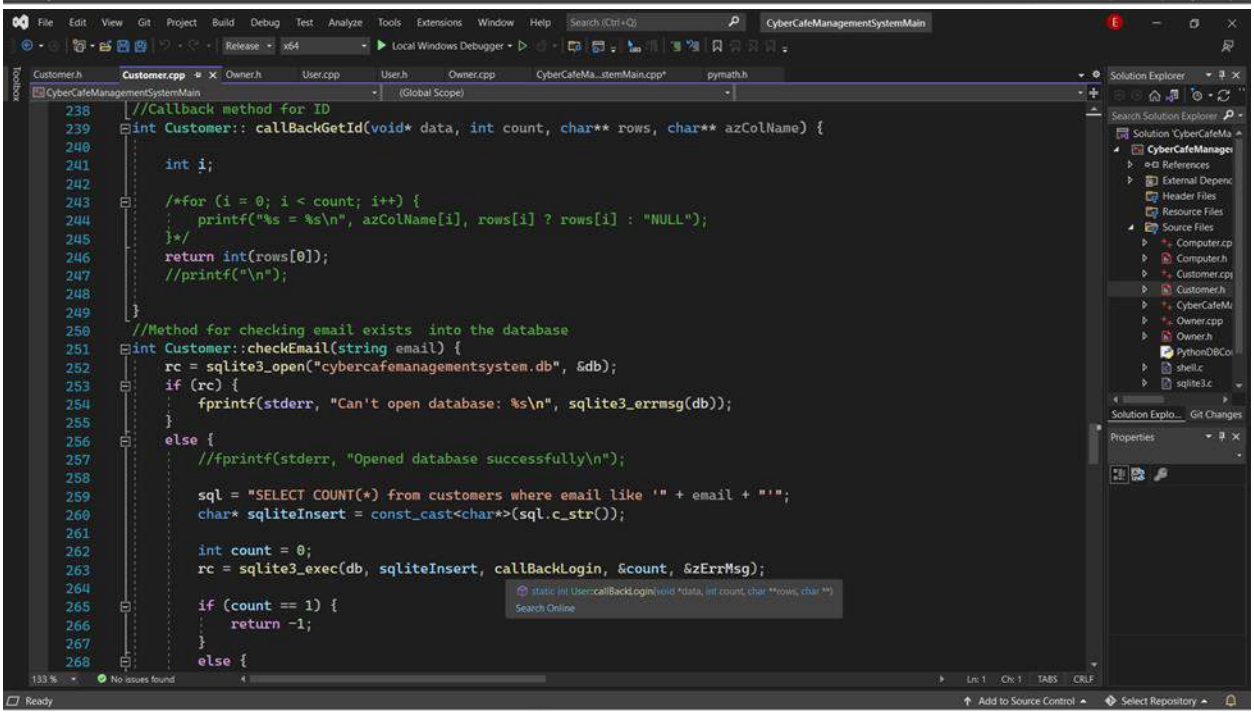


```
172             }
173             else {
174                 cout << "Login Failed" << endl;
175                 exit(0);
176             }
177
178         }
179     }
180
181
182
183
184 // inherited
185  //Method for customer logout
186 void Customer::logout() {
187     rc = sqlite3_open("cybercafemanagementsystem.db", &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 = (SELE
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.

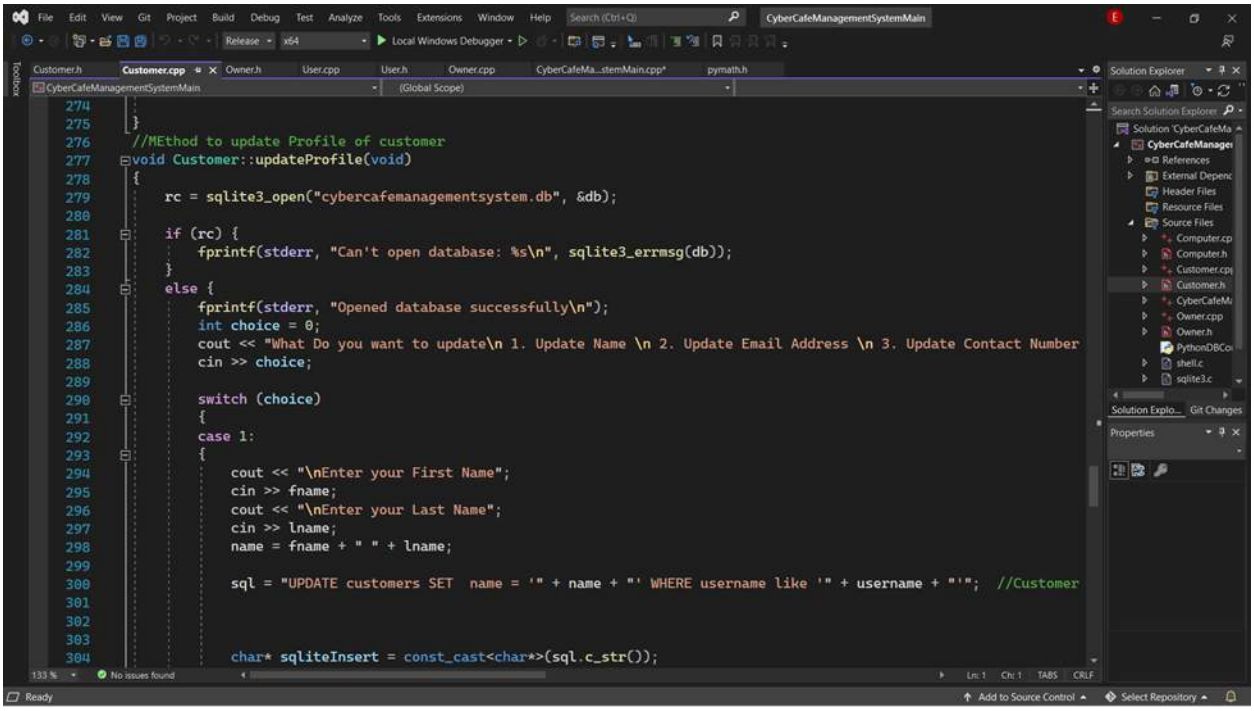


```
202             rc = sqlite3_exec(db, sqliteUpdate, callback, 0, &zErrMsg);
203
204             if (rc != SQLITE_OK) {
205                 fprintf(stderr, "SQL error: %s\n", zErrMsg);
206                 sqlite3_free(zErrMsg);
207             }
208             else {
209
210                 int charge = 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         }
232
```

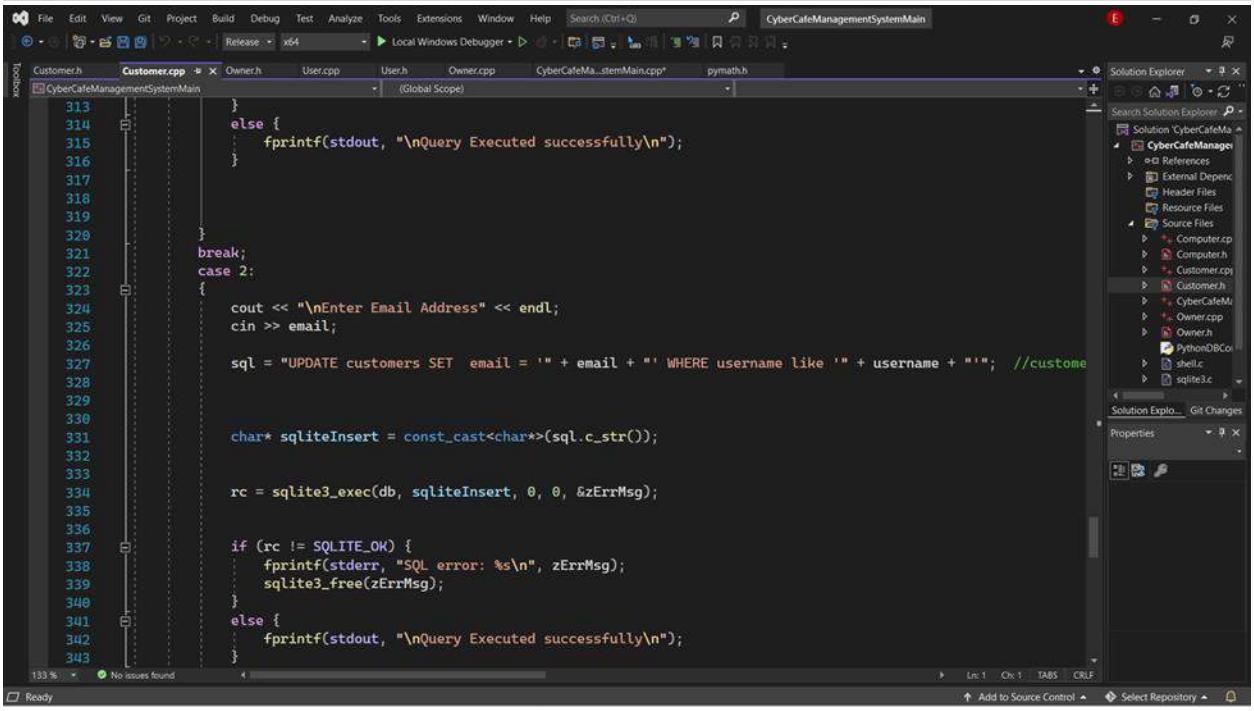


```
238     //Callback method for ID
239 int Customer:: callBackGetId(void* data, int count, char** rows, char** azColName) {
240
241     int i;
242
243     /*for (i = 0; i < count; i++) {
244         printf("%s = %s\n", azColName[i], rows[i] ? rows[i] : "NULL");
245     }*/
246     return int(rows[0]);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

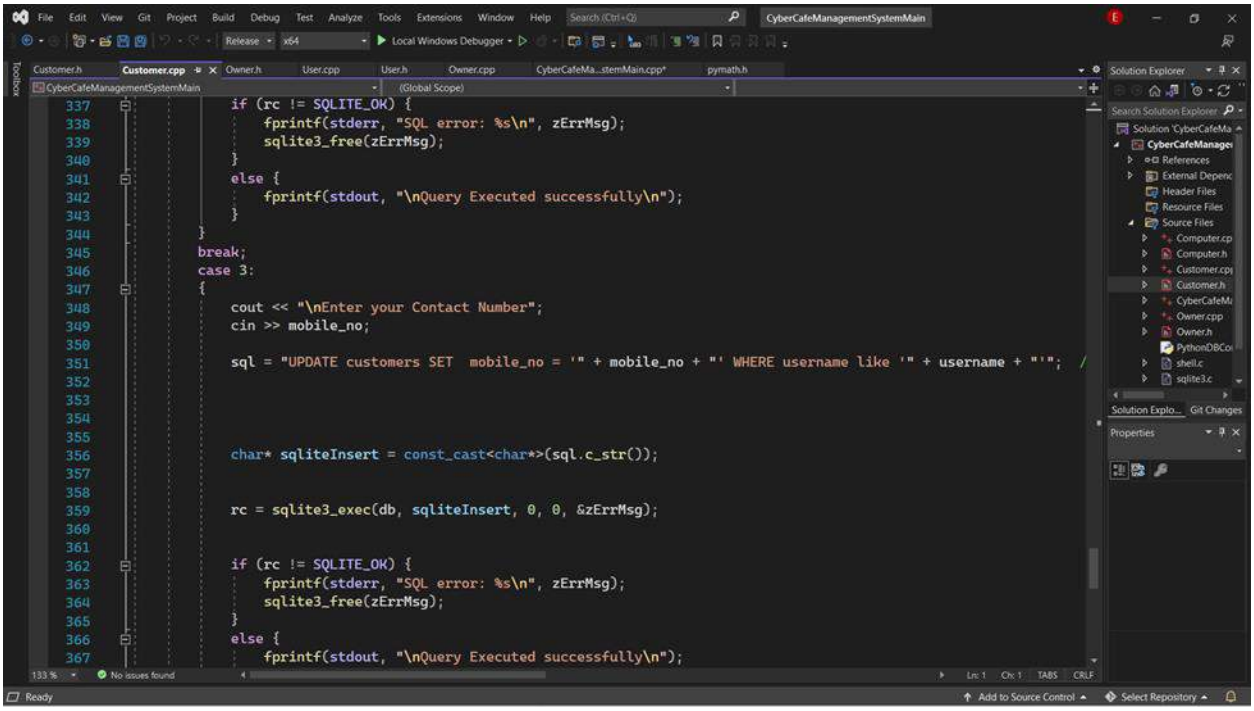
Function definition for customer logout which is overriding User's class virtual function.



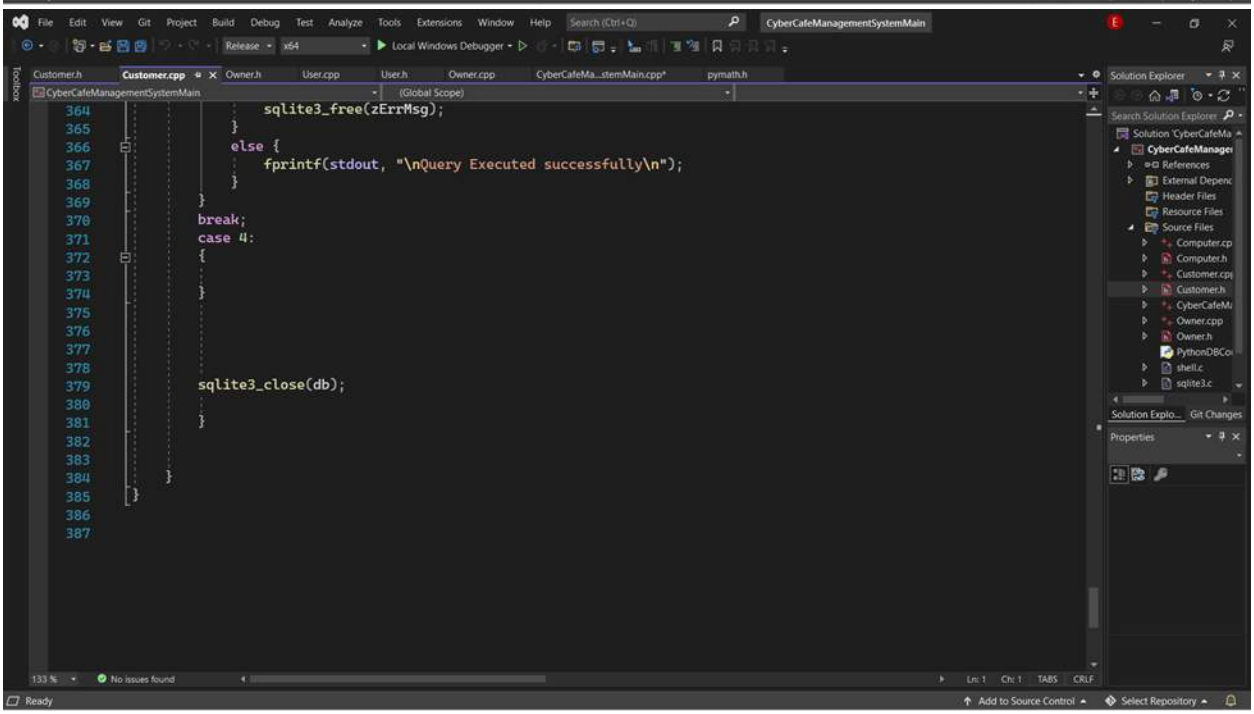
```
}
//MEthod to update Profile of customer
void Customer::updateProfile(void)
{
    rc = sqlite3_open("cybercafemanagementsystem.db", &db);

    if (rc) {
        fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
    }
    else {
        fprintf(stderr, "Opened database successfully\n");
        int choice = 0;
        cout << "What Do you want to update\n 1. Update Name \n 2. Update Email Address \n 3. Update Contact Number
        cin >> choice;

        switch (choice)
        {
        case 1:
        {
            cout << "\nEnter your First Name";
            cin >> fname;
            cout << "\nEnter your Last Name";
            cin >> lname;
            name = fname + " " + lname;

            sql = "UPDATE customers SET  name = '" + name + "' WHERE username like '" + username + "'";  //Customer



            char* sqliteInsert = const_cast<char*>(sql.c_str());
```



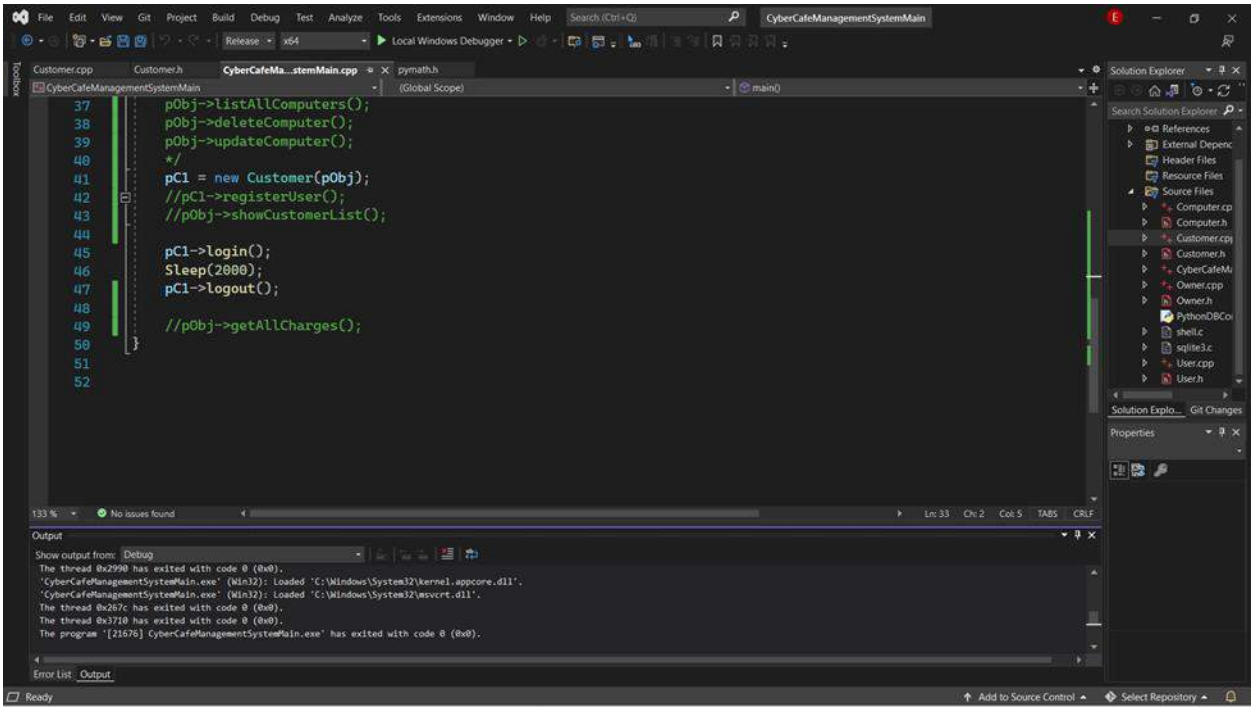
```
            }
            else {
                fprintf(stdout, "\nQuery Executed successfully\n");
            }



        }
        break;
        case 2:
        {
            cout << "\nEnter Email Address" << endl;
            cin >> email;

            sql = "UPDATE customers SET  email = '" + email + "' WHERE username like '" + username + "'";  //custome



            char* sqliteInsert = const_cast<char*>(sql.c_str());


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


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

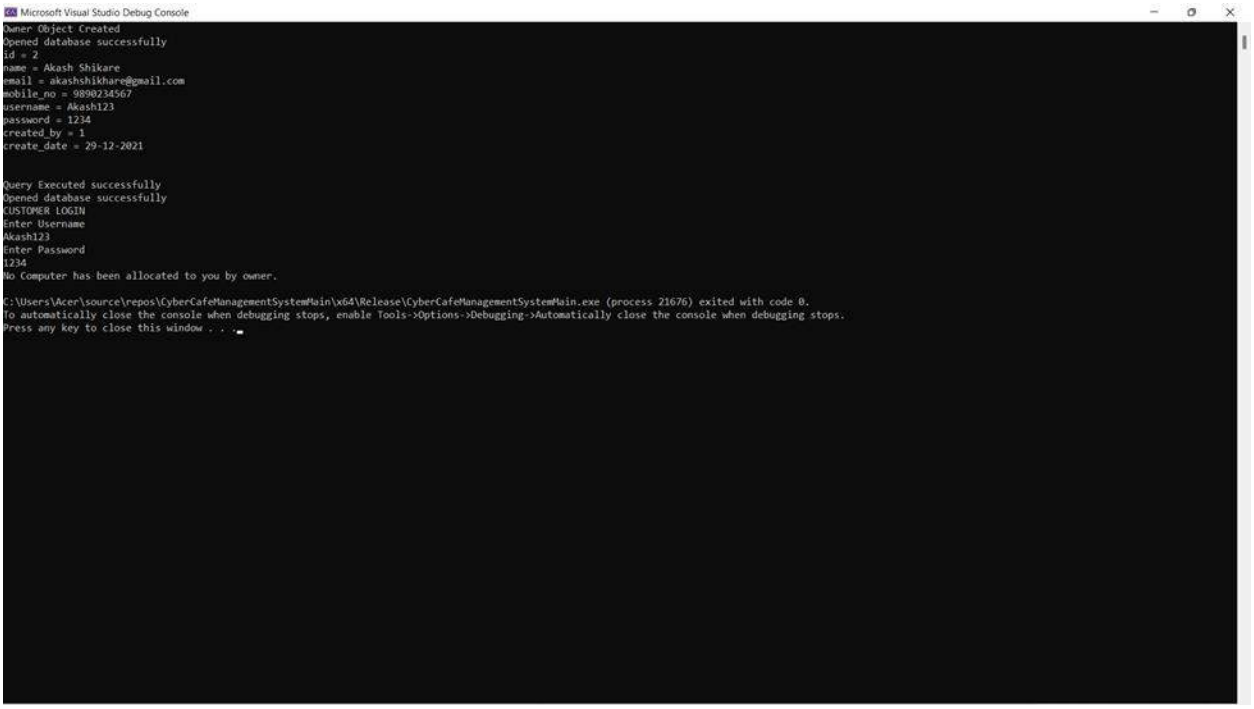


```
337             if (rc != SQLITE_OK) {
338                 fprintf(stderr, "SQL error: %s\n", zErrMsg);
339                 sqlite3_free(zErrMsg);
340             }
341             else {
342                 fprintf(stdout, "\nQuery Executed successfully\n");
343             }
344         }
345 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



```
364                 sqlite3_free(zErrMsg);
365             }
366             else {
367                 fprintf(stdout, "\nQuery Executed successfully\n");
368             }
369         }
370         break;
371         case 4:
372         {
373
374         }
375
376
377
378
379         sqlite3_close(db);
380
381         }
382
383
384     }
385 }
386
387
```

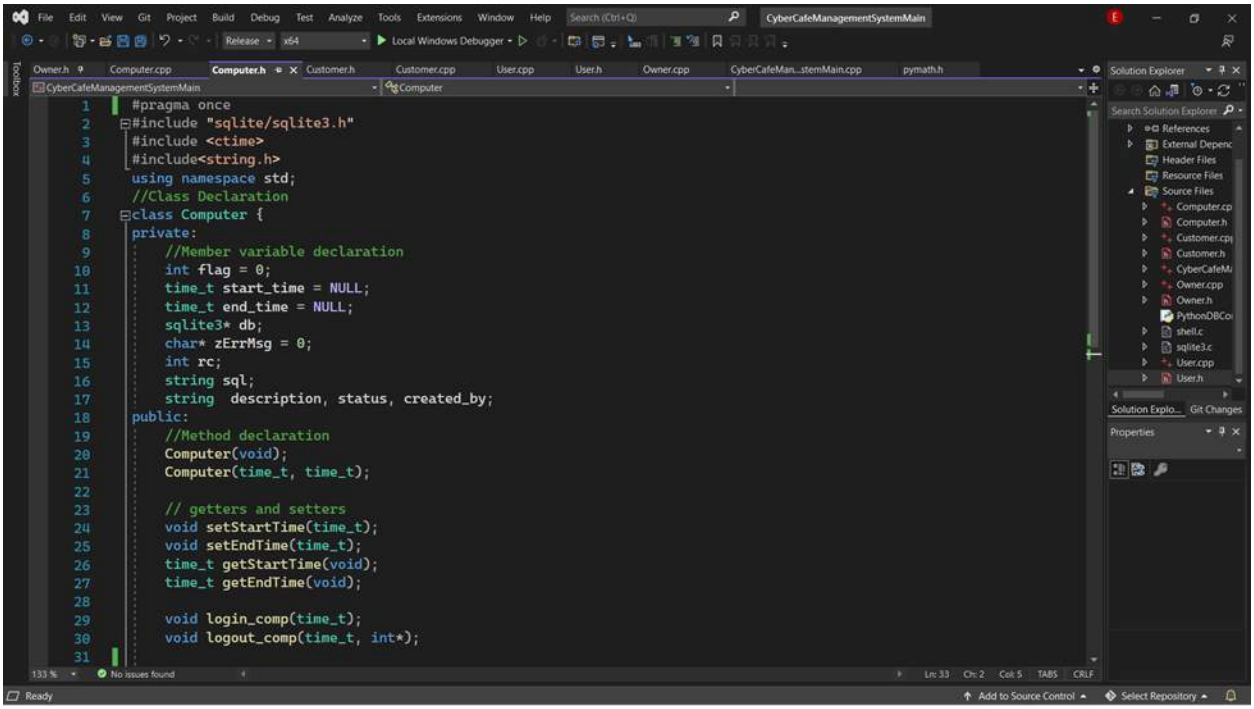




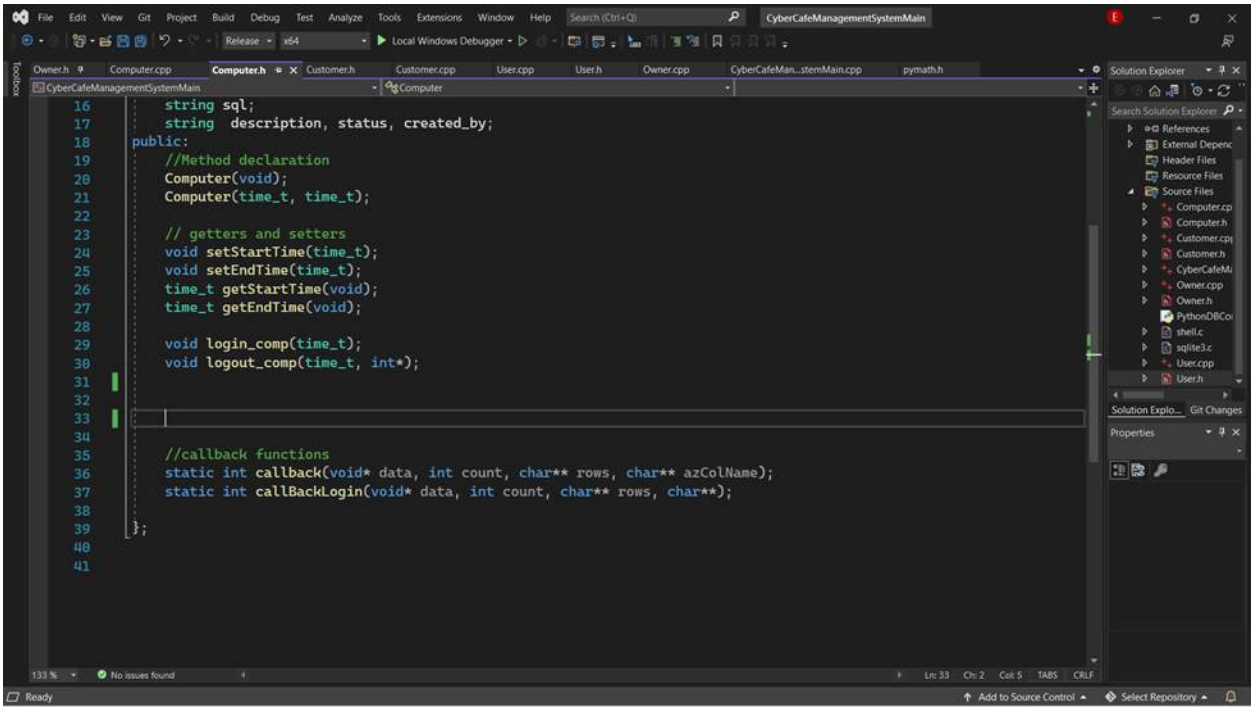
Function definition to update customer profile in which customer can update its name, mobile number, email.

Computer class

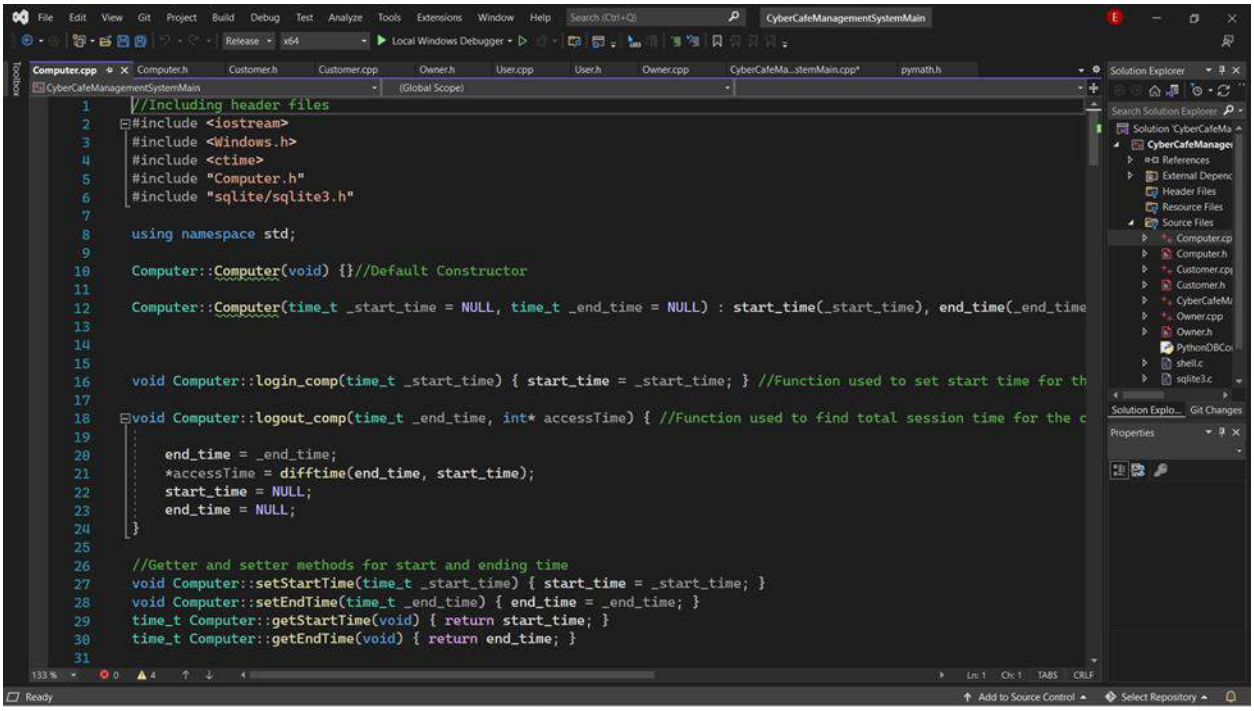
Class declaration



```
#pragma once
#include "sqlite/sqlite3.h"
#include <ctime>
#include<string.h>
using namespace std;
//Class Declaration
class Computer {
private:
    //Member variable declaration
    int flag = 0;
    time_t start_time = NULL;
    time_t end_time = NULL;
    sqlite3* db;
    char* zErrMsg = 0;
    int rc;
    string sql;
    string  description, status, created_by;
public:
    //Method declaration
    Computer(void);
    Computer(time_t, time_t);

    // getters and setters
    void setStartTime(time_t);
    void setEndTime(time_t);
    time_t getStartTime(void);
    time_t getEndTime(void);

    void login_comp(time_t);
    void logout_comp(time_t, int*);
```

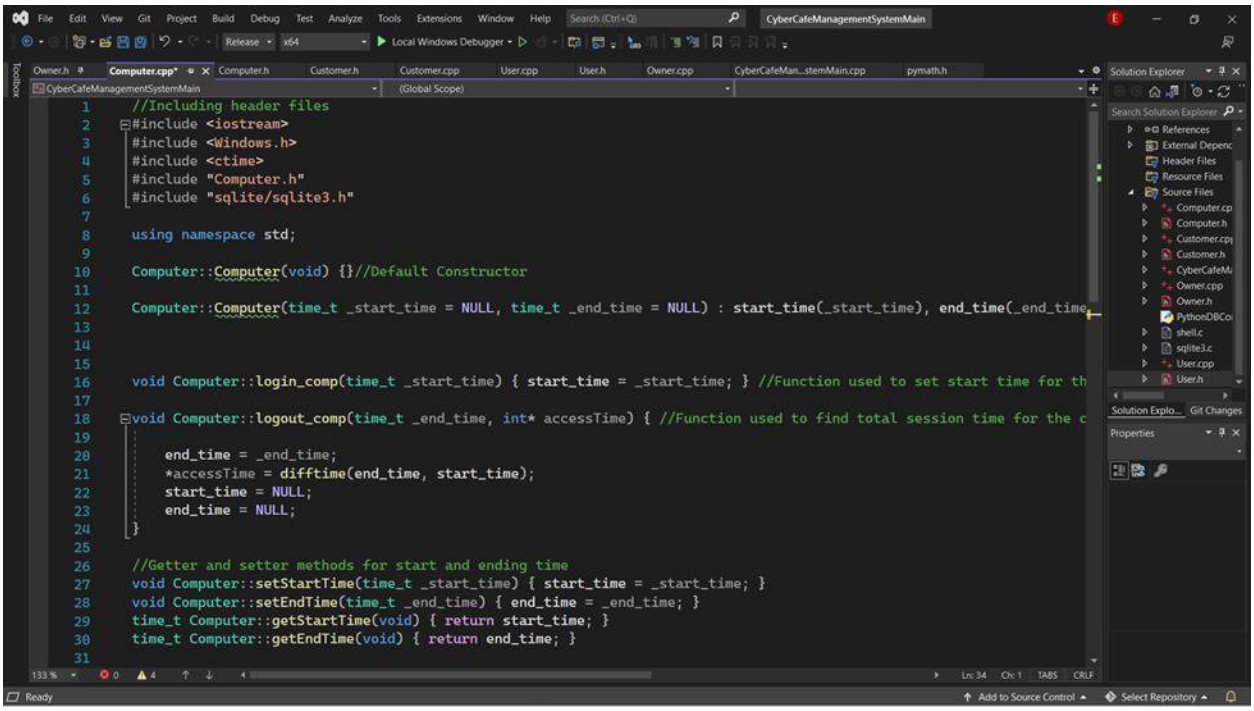


```
    string sql;
    string  description, status, created_by;
public:
    //Method declaration
    Computer(void);
    Computer(time_t, time_t);

    // getters and setters
    void setStartTime(time_t);
    void setEndTime(time_t);
    time_t getStartTime(void);
    time_t getEndTime(void);

    void login_comp(time_t);
    void logout_comp(time_t, int*);


    //callback functions
    static int callback(void* data, int count, char** rows, char** azColName);
    static int callBackLogin(void* data, int count, char** rows, char**);

};
```

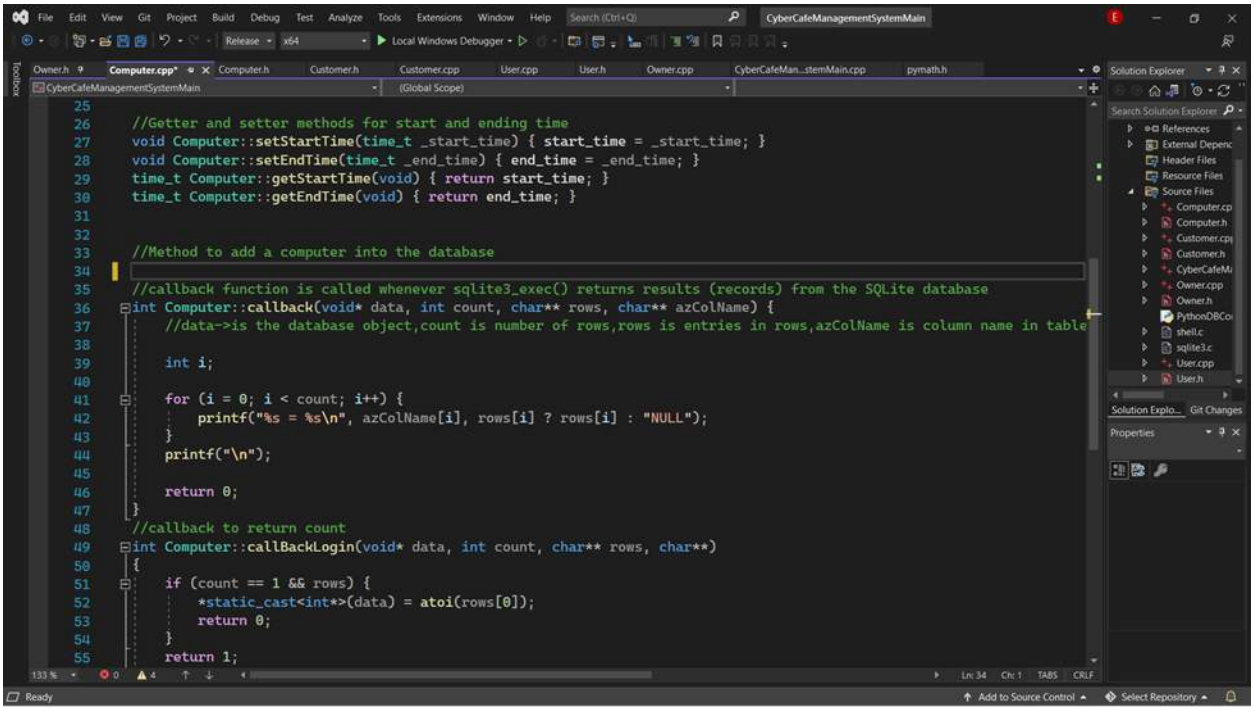


```
//Including header files
#include <iostream>
#include <Windows.h>
#include <ctime>
#include "Computer.h"
#include "sqlite/sqlite3.h"

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 th

void Computer::logout_comp(time_t _end_time, int* accessTime) { //Function used to find total session time for the c

    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; }
```

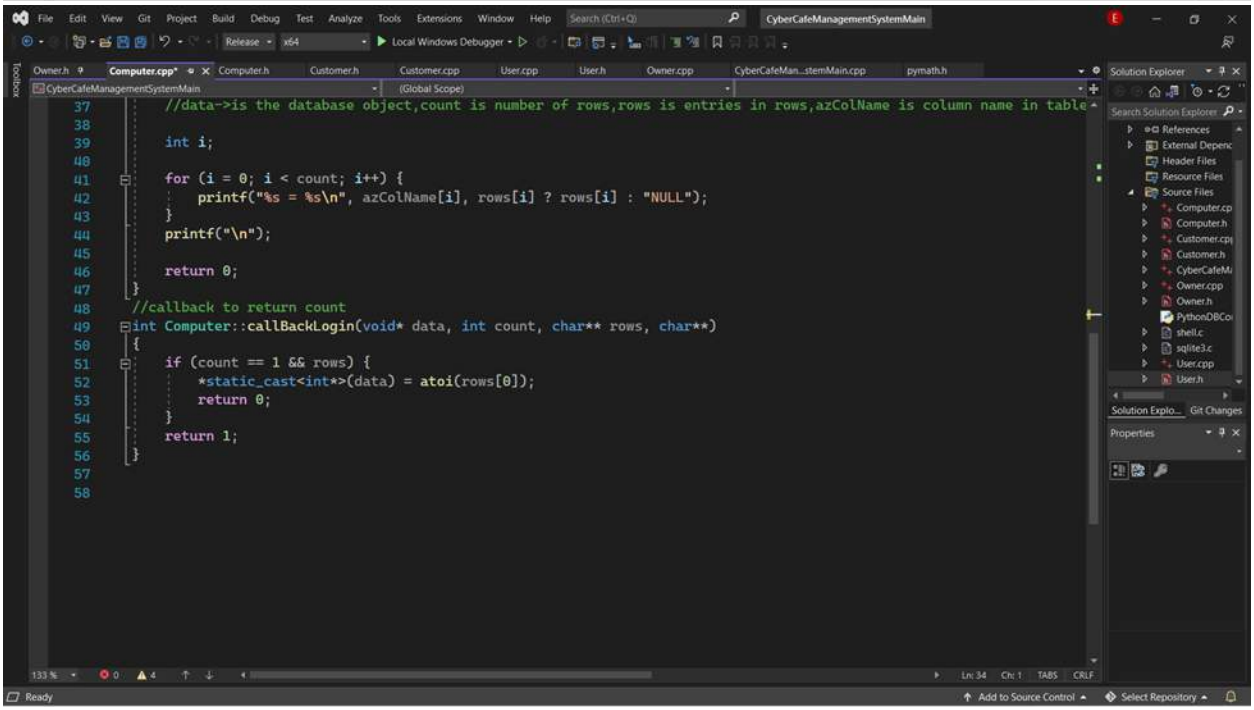
Class declaration of Computer which contain declaration of computer data members getter, setters and Computer class default and parameterized constructor definition.



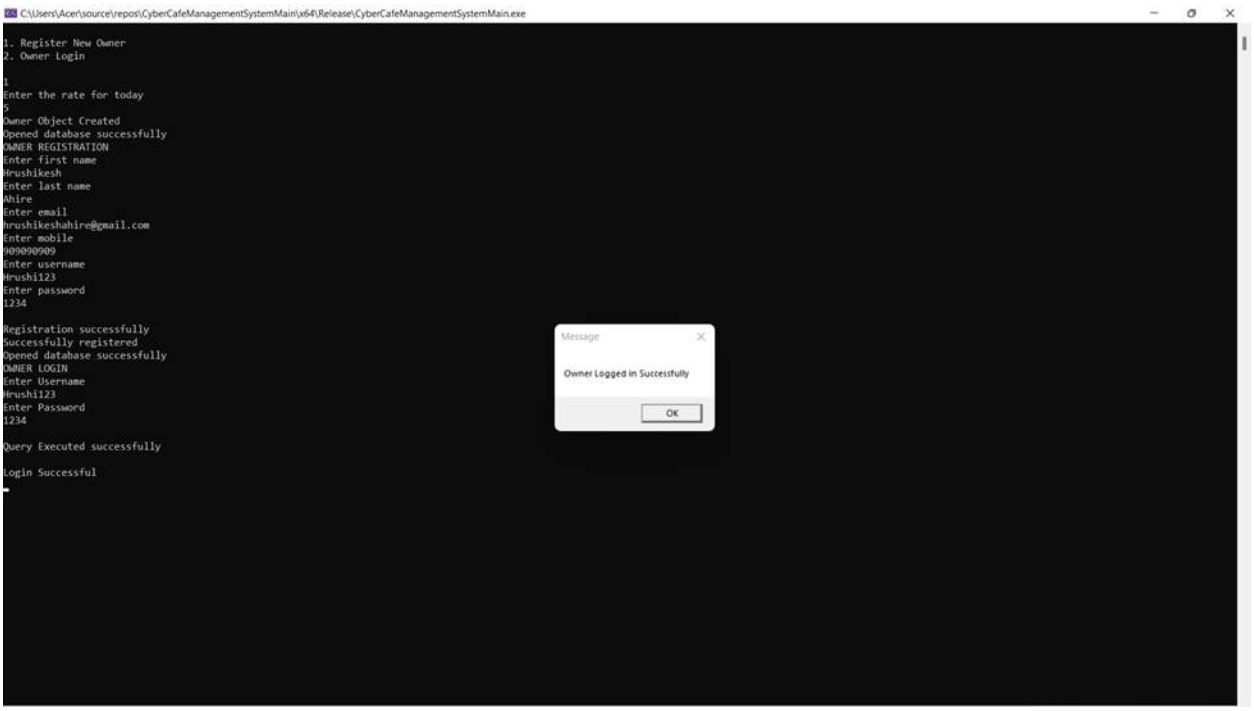
```
//Including header files
#include <iostream>
#include <Windows.h>
#include <ctime>
#include "Computer.h"
#include "sqlite/sqlite3.h"

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 th

void Computer::logout_comp(time_t _end_time, int* accessTime) { //Function used to find total session time for the c

    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; }
```



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



```
37      //data->is the database object,count is number of rows,rows is entries in rows,azColName is column name in table
38
39      int i;
40
41      for (i = 0; i < count; i++) {
42          printf("%s = %s\n", azColName[i], rows[i] ? rows[i] : "NULL");
43      }
44      printf("\n");
45
46      return 0;
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          return 0;
54      }
55      return 1;
56  }
57
58
```



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

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.

CMakeLists.txt

```
cmake_minimum_required (VERSION 3.10)
set(CMAKE_BUILD_TYPE Debug)
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++14")

project (CCMS)
add_executable(CyberCafeManagement CyberCafeManagementSystemMain.cpp)

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.

Makefile

```
##############################################################################
###################### Makefile Template #####################################
##############################################################################

# Compiler settings - Can be customized.
CC = g++
CXXFLAGS = -std=c++11 -Wall
LDFLAGS =

# Makefile settings - Can be customized.
APPNAME = CCMS
EXT = .cpp
SRCDIR = .
OBJDIR = .

############## Do not change anything from here downwards! #############
SRC = $(wildcard $(SRCDIR)/*$(EXT))
OBJ = $(SRC:$(SRCDIR)/%$(EXT)=$(OBJDIR)/%.o)
DEP = $(OBJ:$(OBJDIR)/%.o=%.d)
# UNIX-based OS variables & settings
RM = rm
DELOBJ = $(OBJ)
# Windows OS variables & settings
DEL = del
EXE = .exe
WDELOBJ = $(SRC:$(SRCDIR)/%$(EXT)=$(OBJDIR)\\%.o)

##############################################################################
###################### Targets beginning here ################################
##############################################################################

all: $(APPNAME)

# Builds the app
$(APPNAME): $(OBJ)
    $(CC) $(CXXFLAGS) -o $@ $^ $(LDFLAGS)

```

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.

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.

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.obj Computer.h sqlite3.h
24      $(CXX) -c Computer.cpp -o Computer.obj
25
26  Customer.obj: Customer.h sqlite3.h
27      $(CXX) -c Customer.cpp -o Customer.obj
28
29  Helper.obj: Helper.h sqlite3.h
30      $(CXX) -c Customer.cpp -o Customer.obj
31
32  clean:
33      -rm *.obj
34      -rm *.exe
```

Code To Create obj files.

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.

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.obj 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.

```
DWORD WINAPI registerThread(LPVOID param)
{

    _createOwner();
    return 0;
}
```

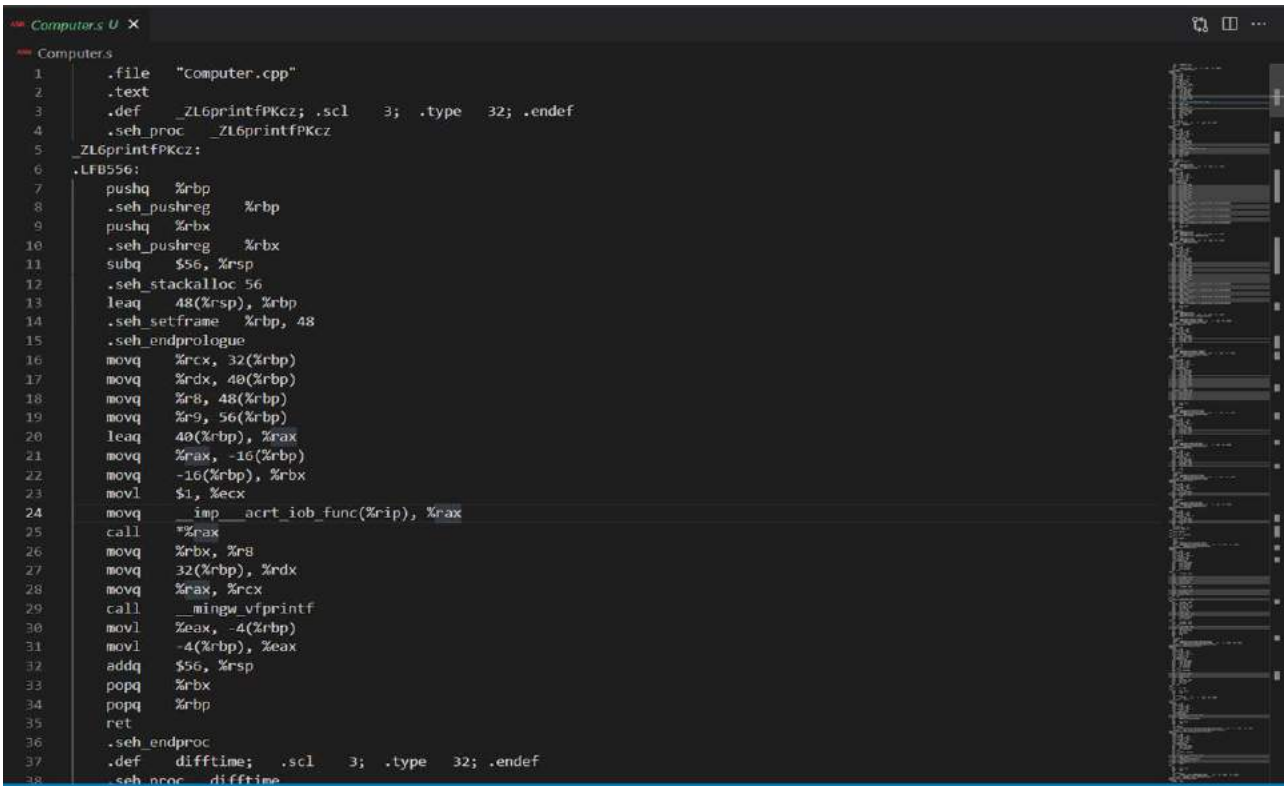
Milestone 4

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



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 :

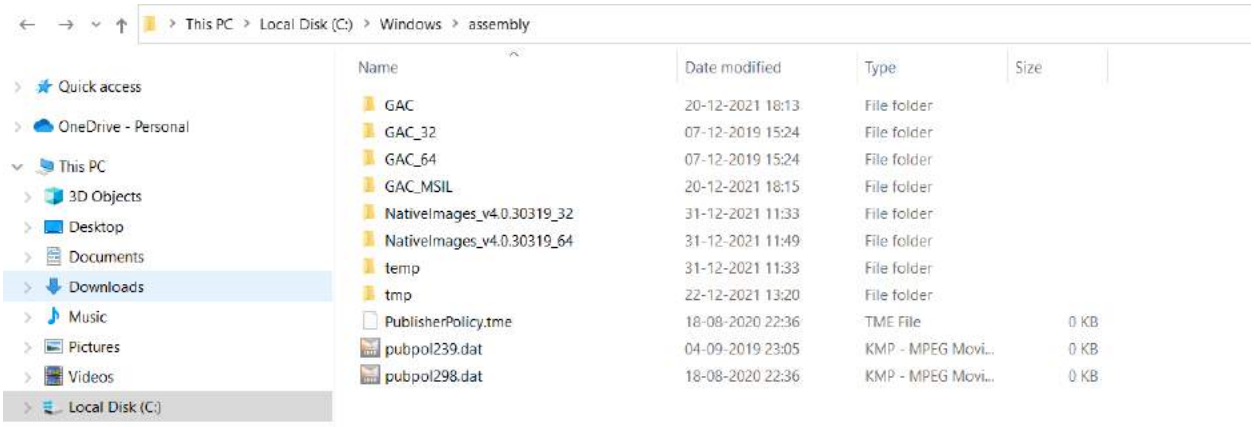
- MOV - move data from one location to another
- ADD - add two values

- 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

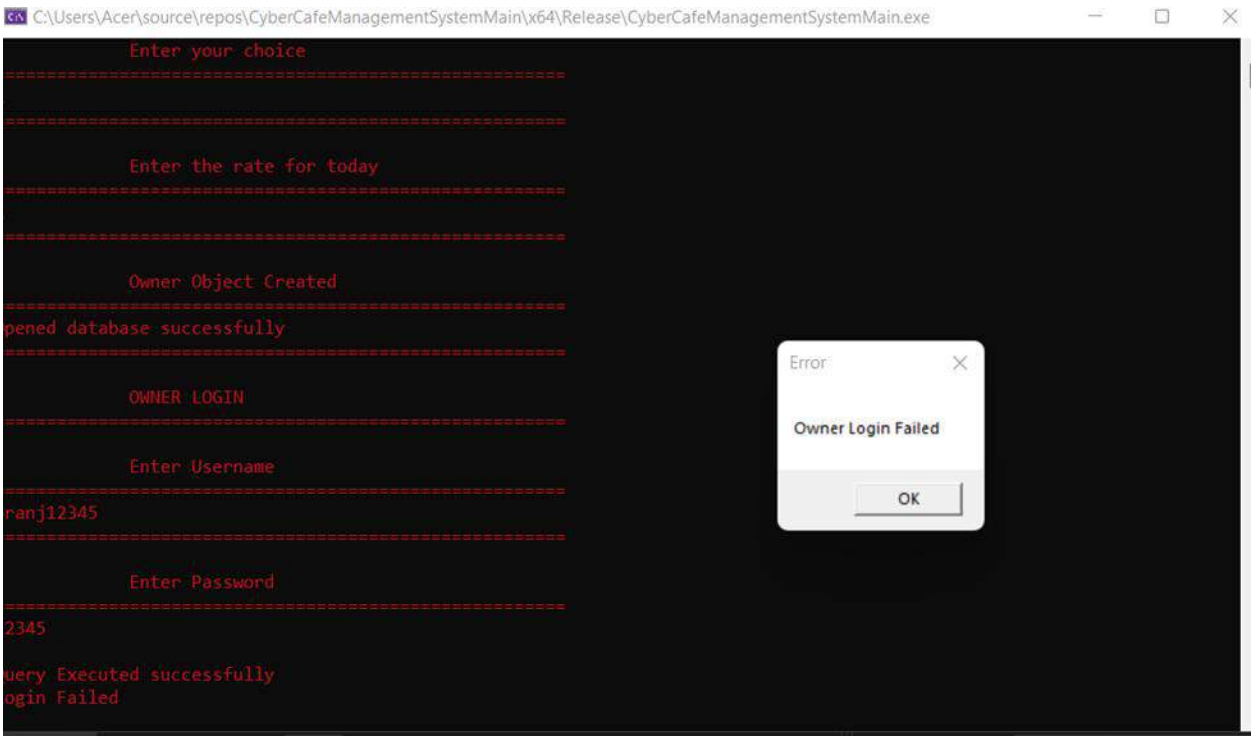
GAC is a shared location in your machine that keeps assemblies, DLLs for common sharing purposes. Assume that you are using *system.data.dll* in your project. So one option is you can place the DLL with your project. But assume you are creating another project which is also using the same .NET assembly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

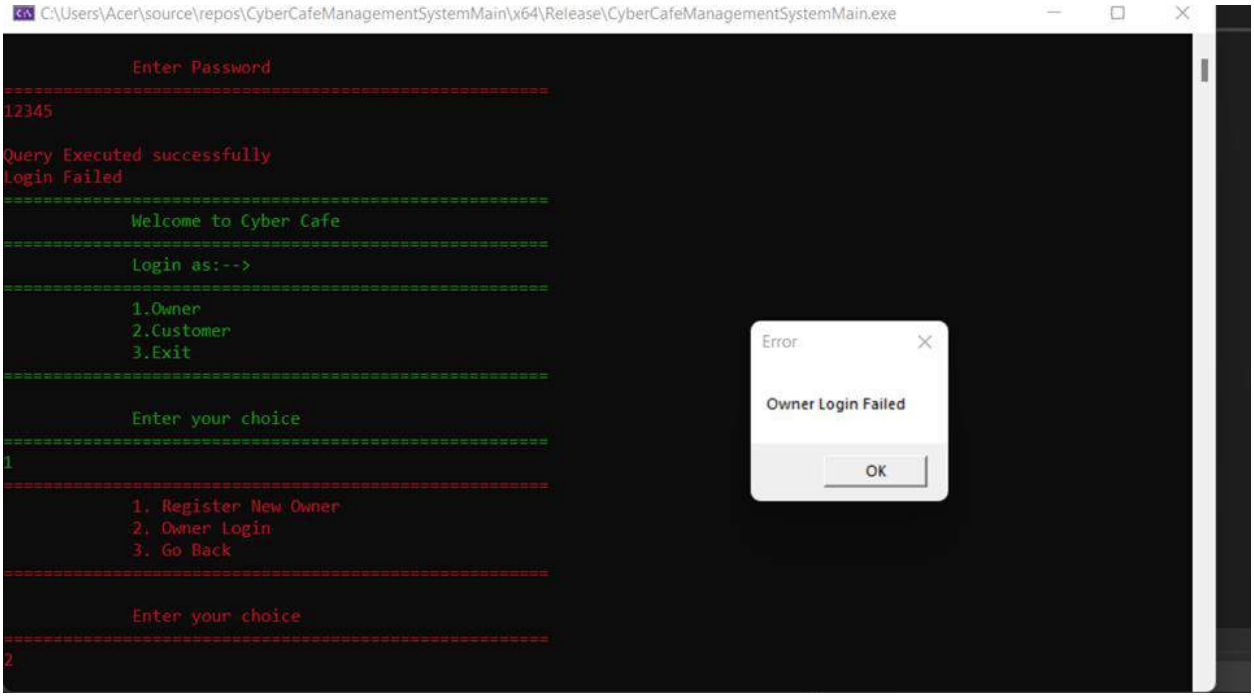


Debugging(Windows)

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

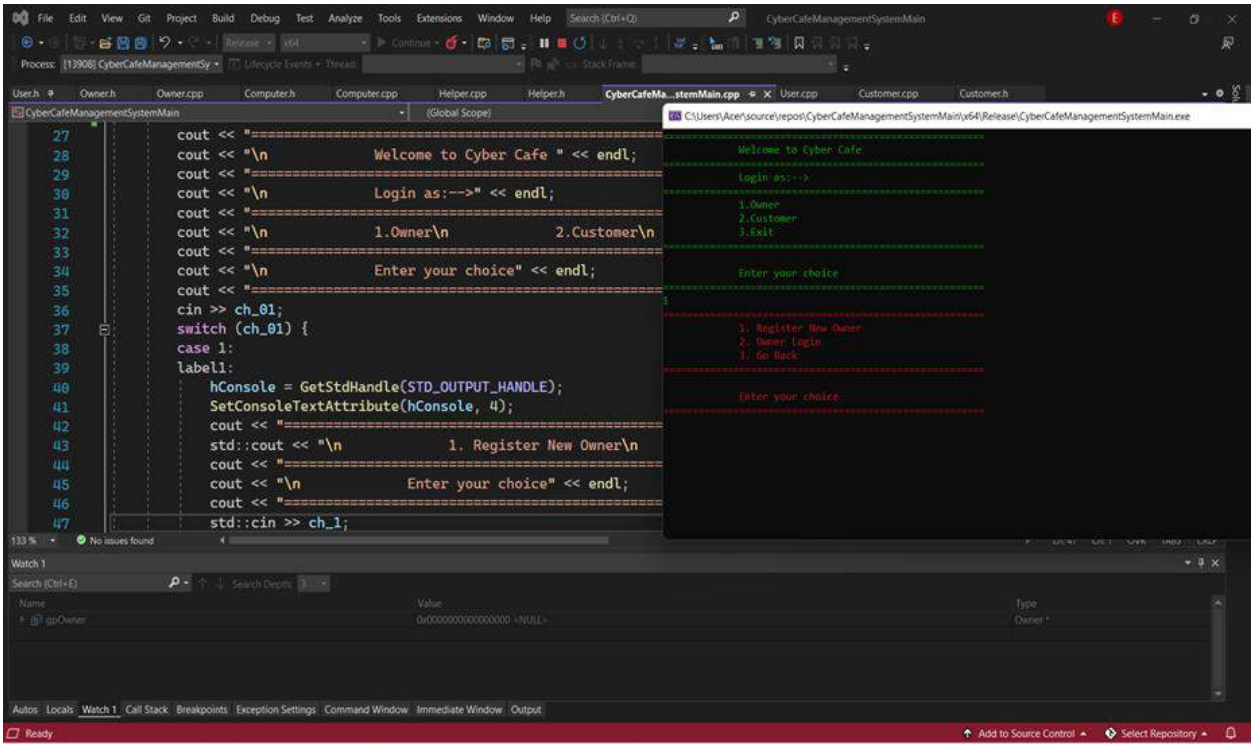


2) Then After login failed it goes to the home Page. If After trying to login again. It says login failed without even asking the username and password.

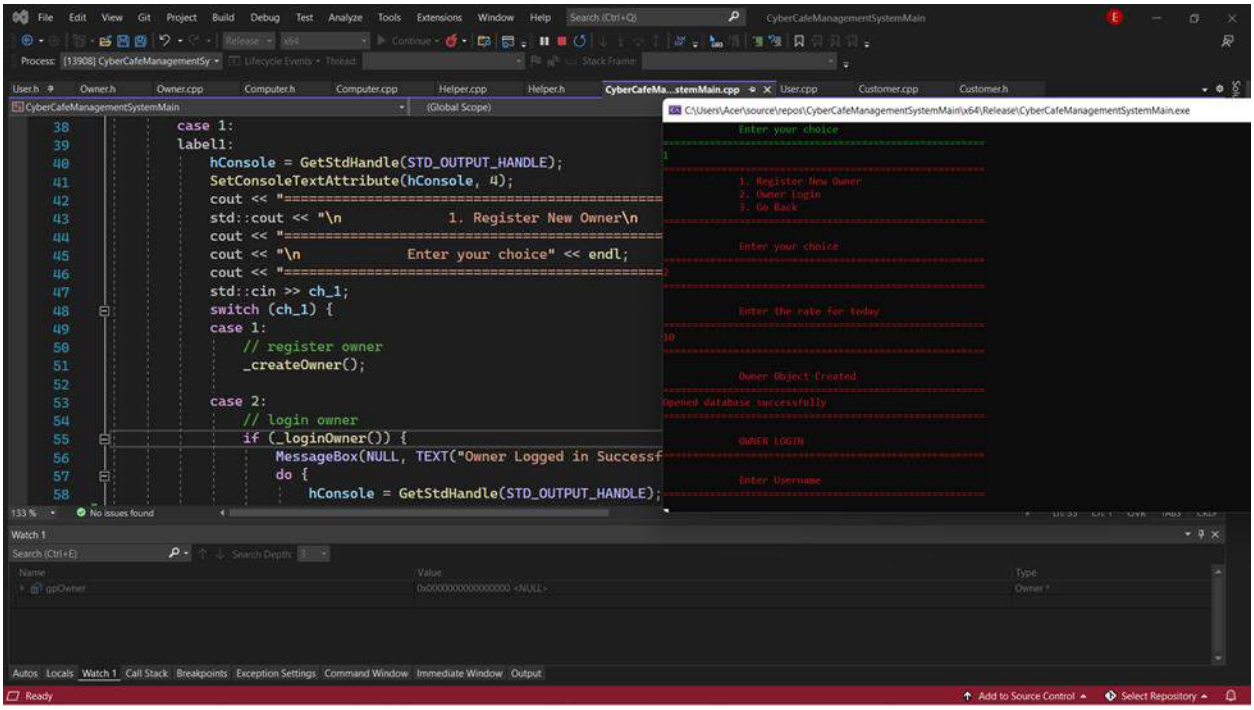


3) Starting to Debug: Breakpoint at main.

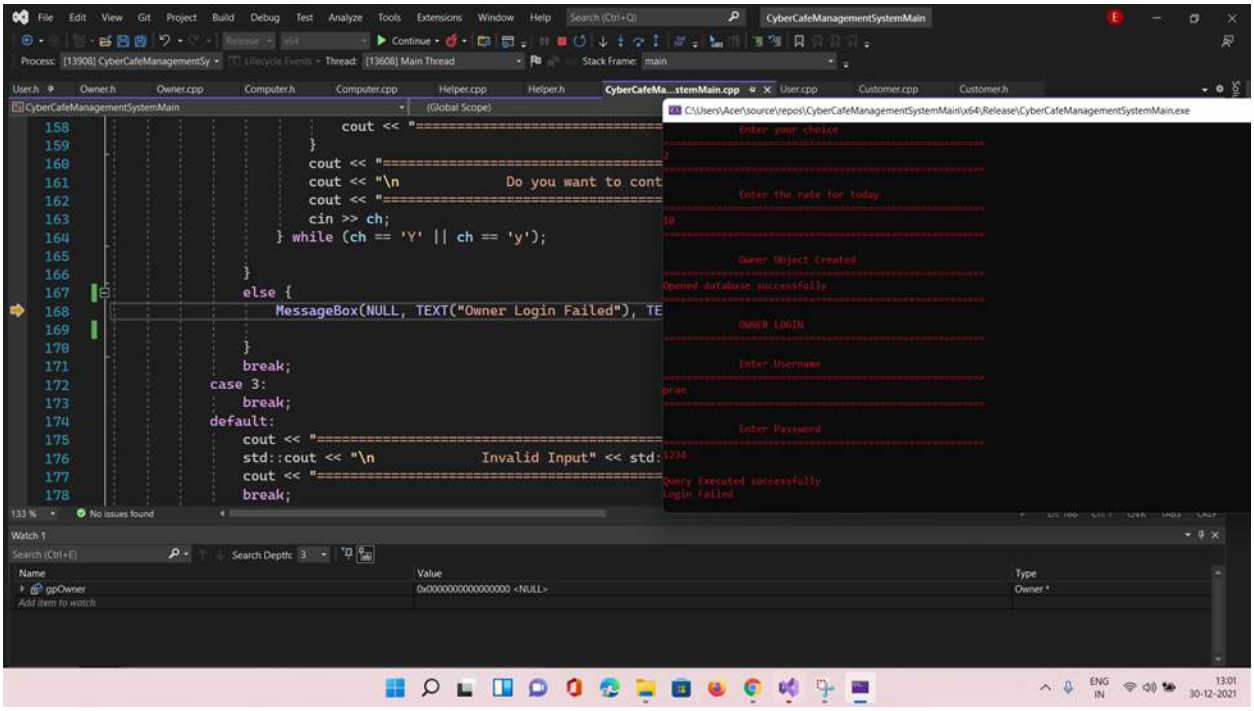
4) Started Debugging: Started watching gpOwner.



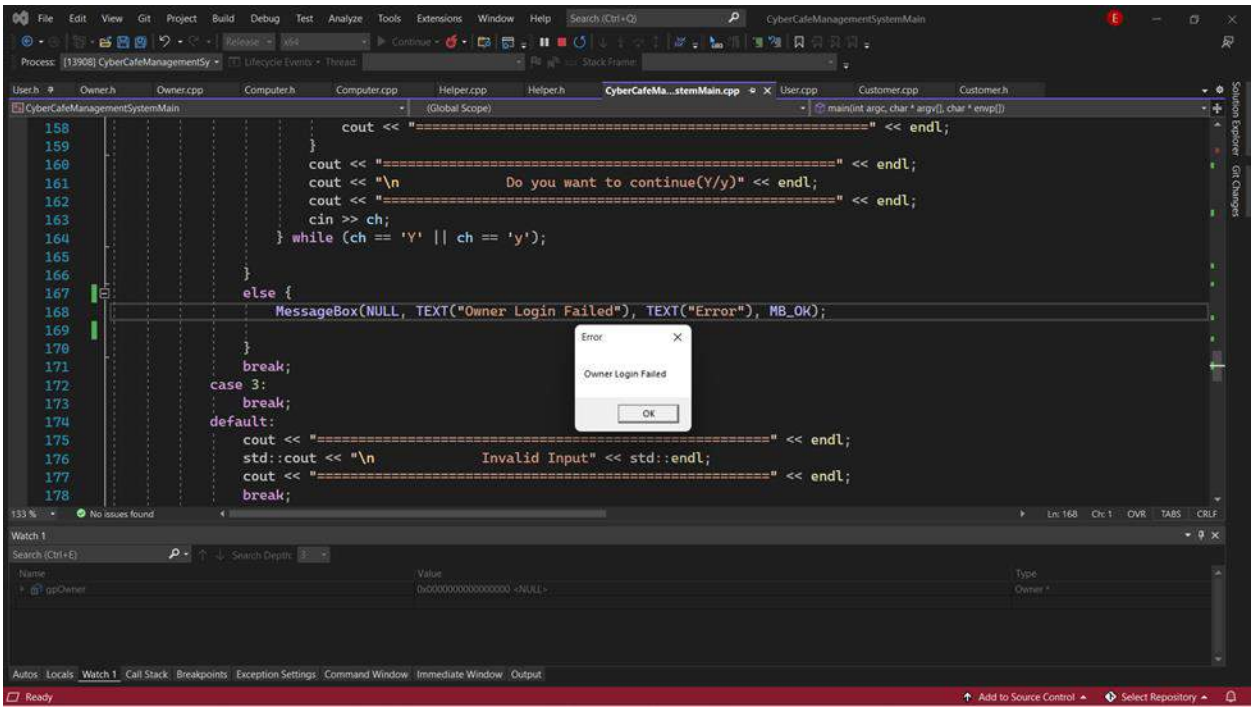
5) Entered 1 to login as Owner.



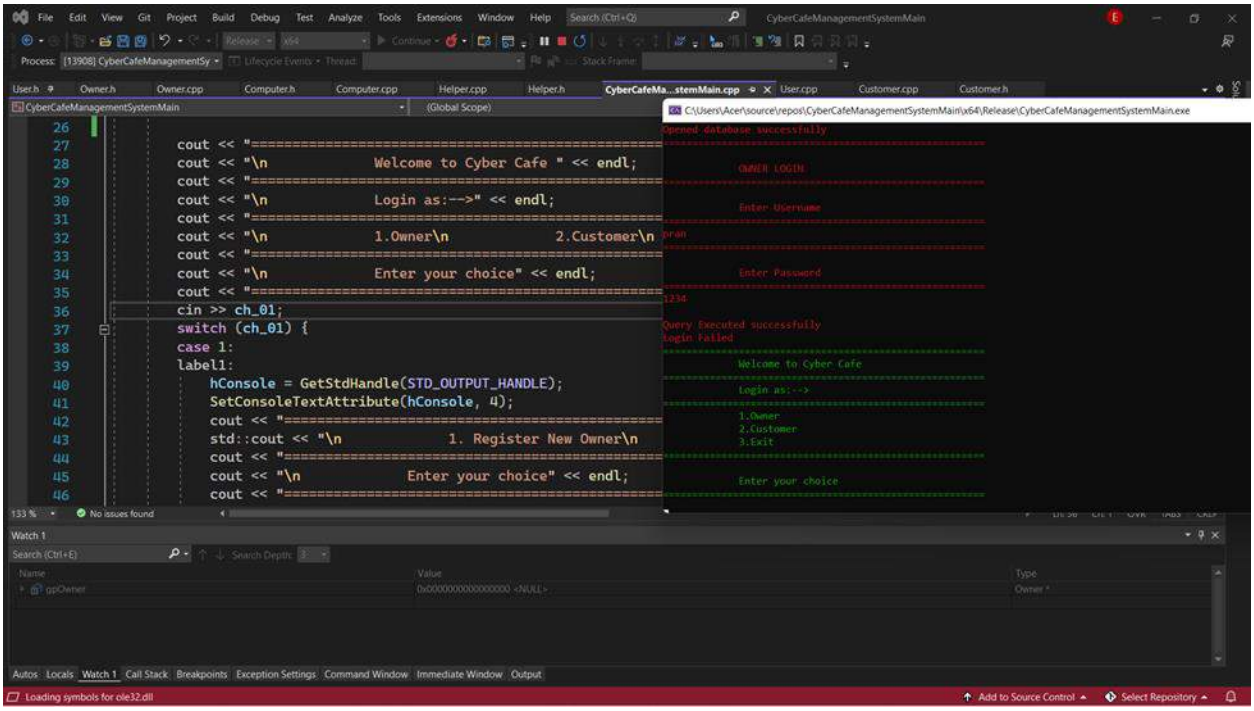
6) Entered 2 to Owner Login and entered the rate_per_min.



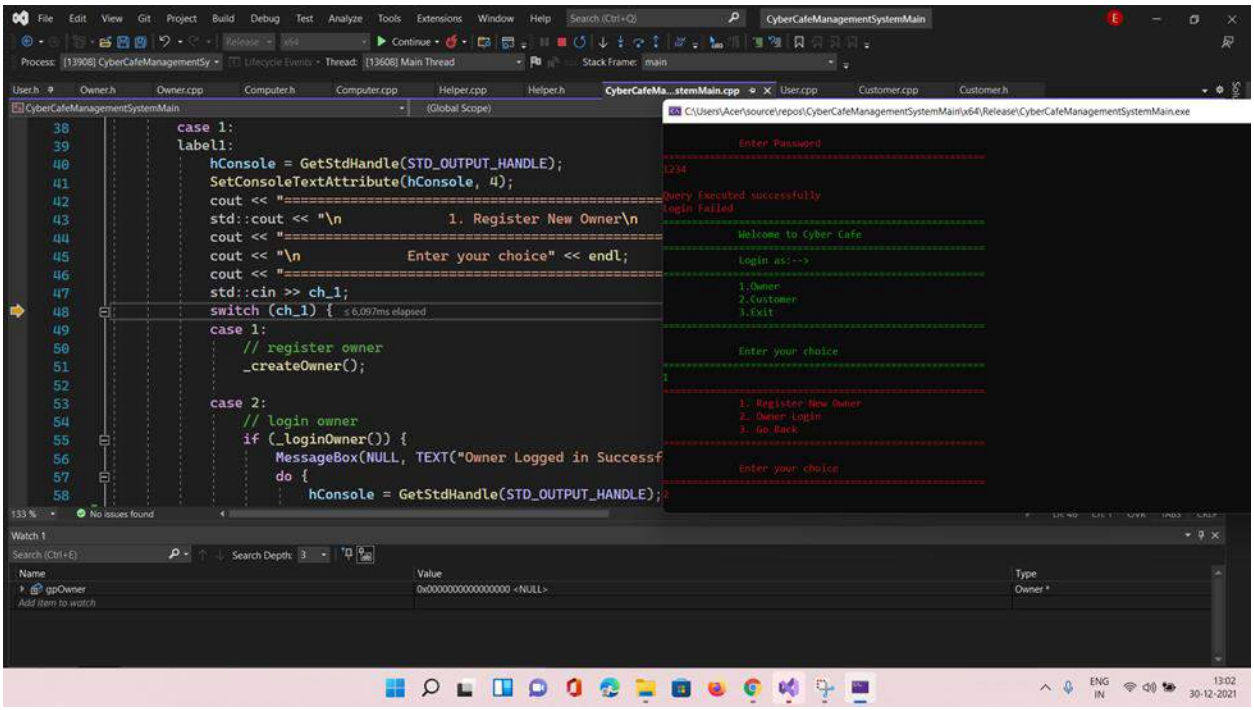
7) Entered Wrong Credentials For First Login.



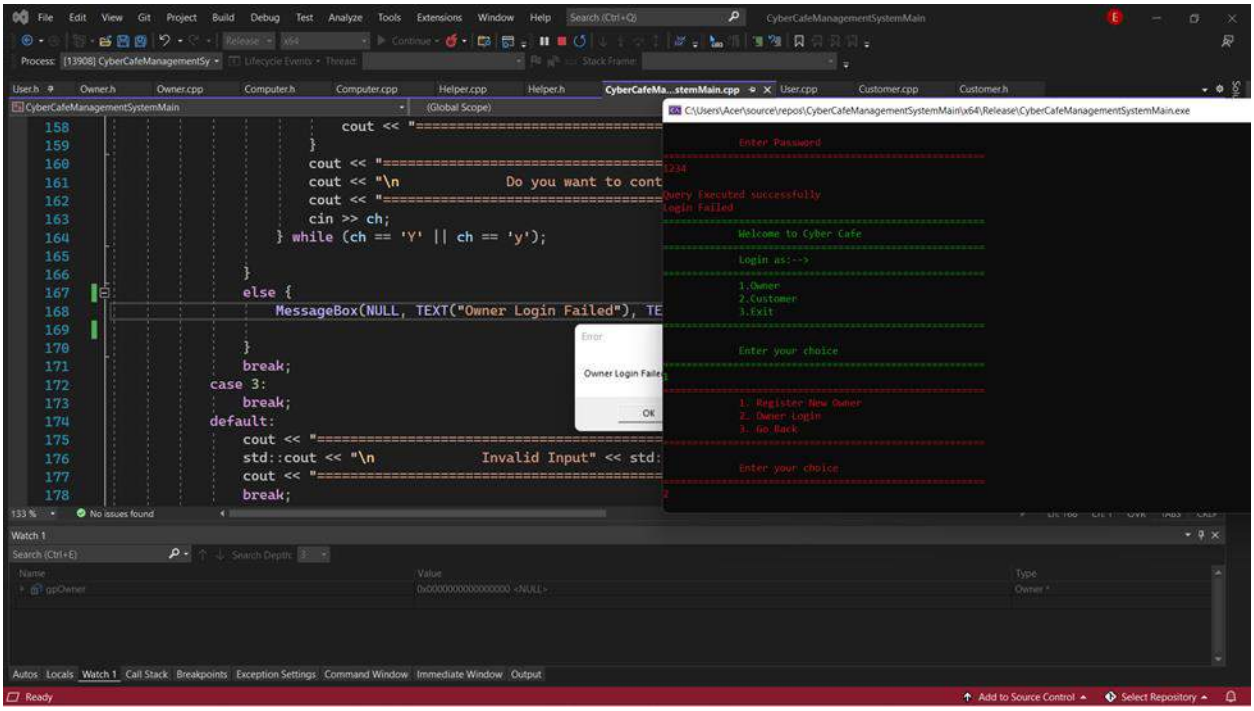
8) Login Failed.



9) Went to homepage.

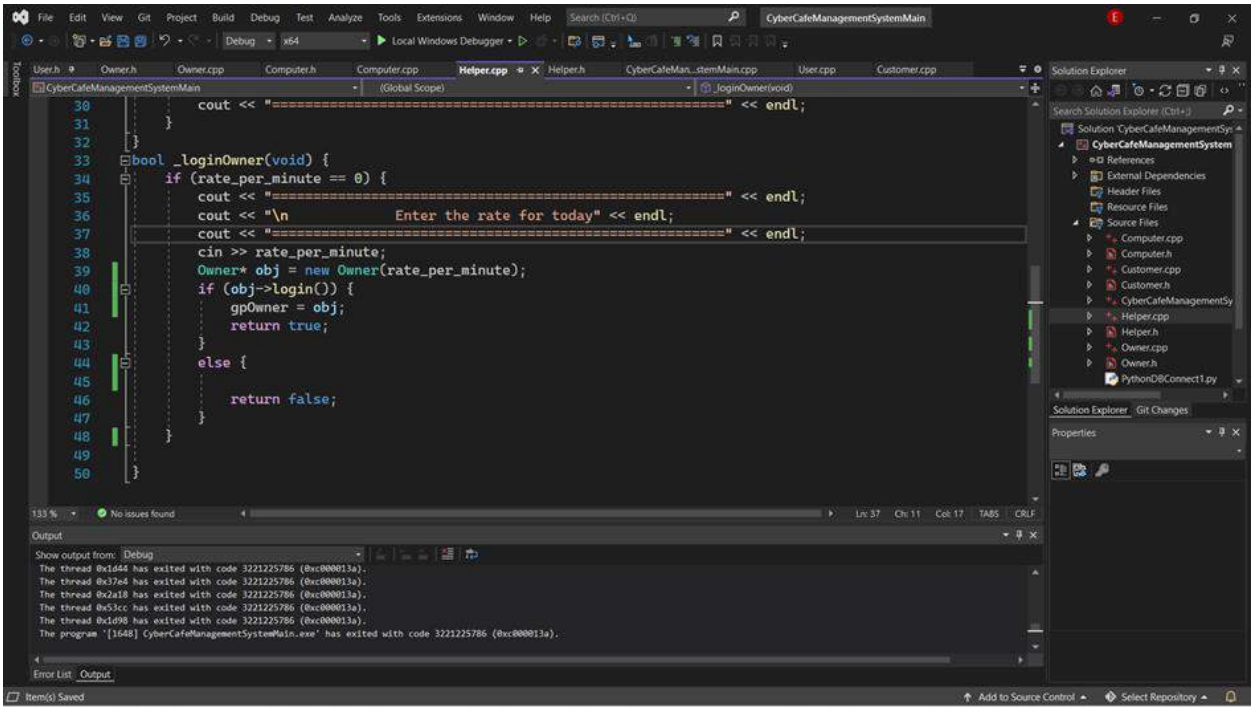


10)Entered 1 for Login as Owner. And entered 2 for Owner Login.

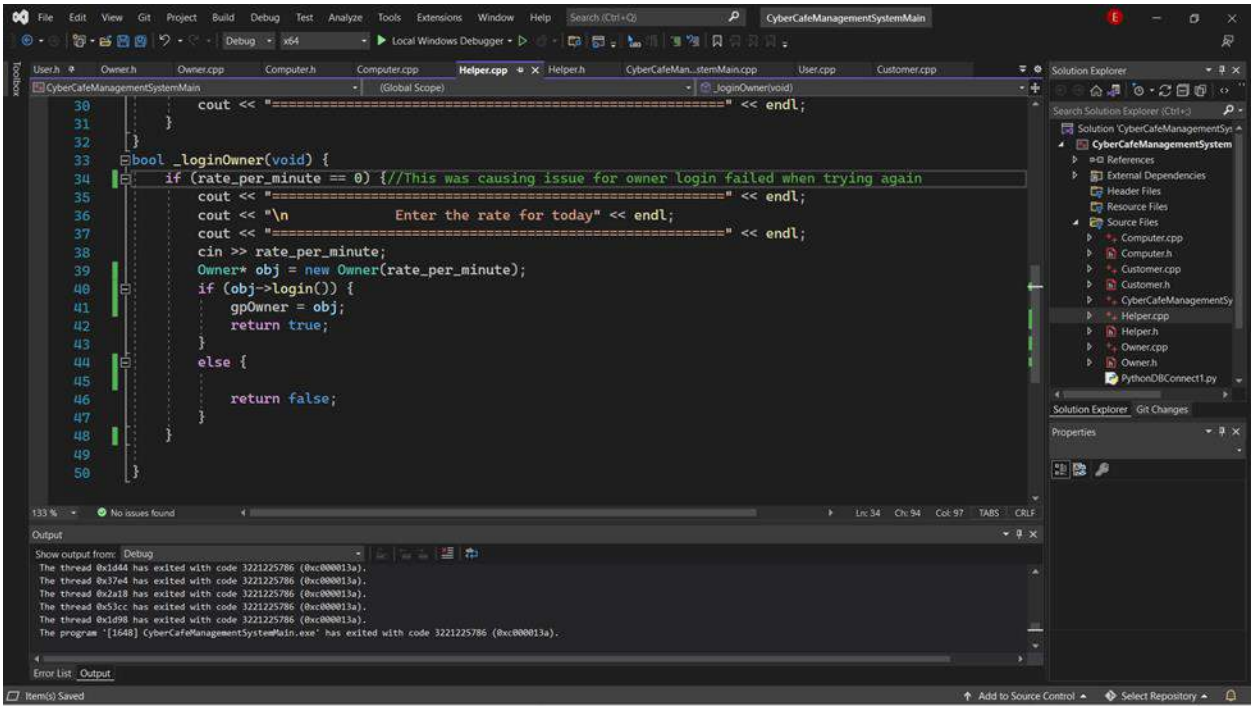


11) Without asking the username and password it says Login Failed and the gpOwner values still doesn't change. It Remains Null.

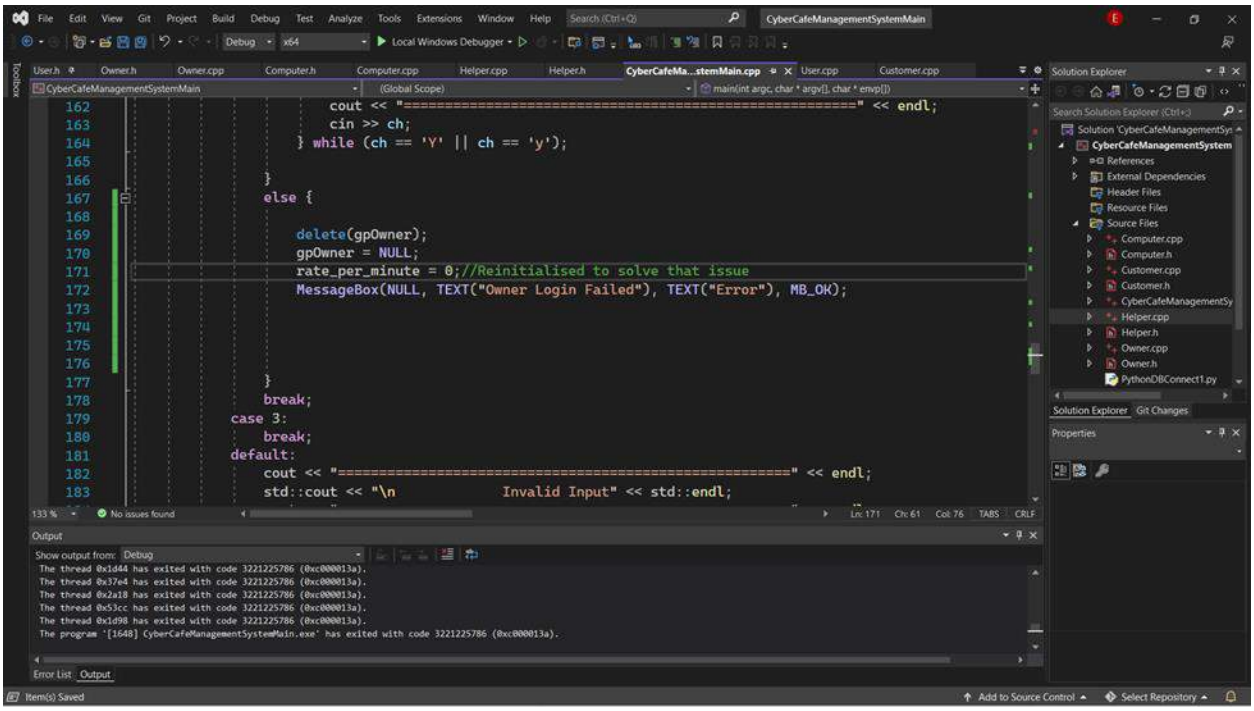
12)Tried Watching rate_per_miniute as well.



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



14) Reinitialized rate_per_minitue to 0 when login failed.



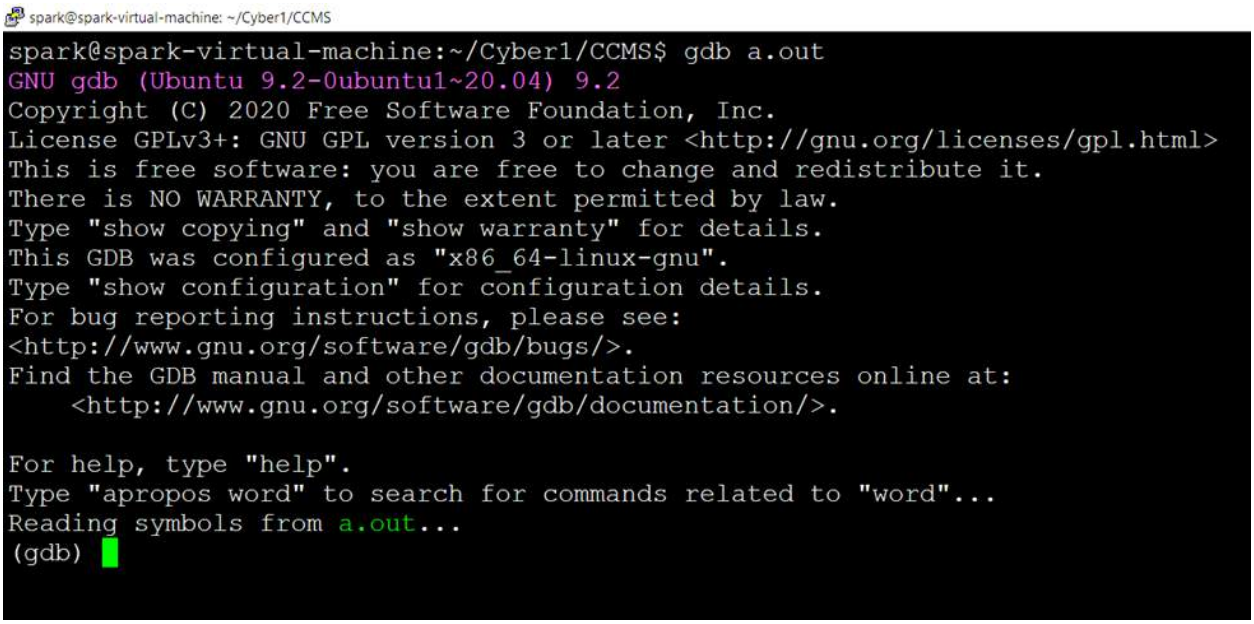
Open-Source Debugging

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$
spark@spark-virtual-machine:~/Cyber1/CCMS$
spark@spark-virtual-machine:~/Cyber1/CCMS$
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.



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

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

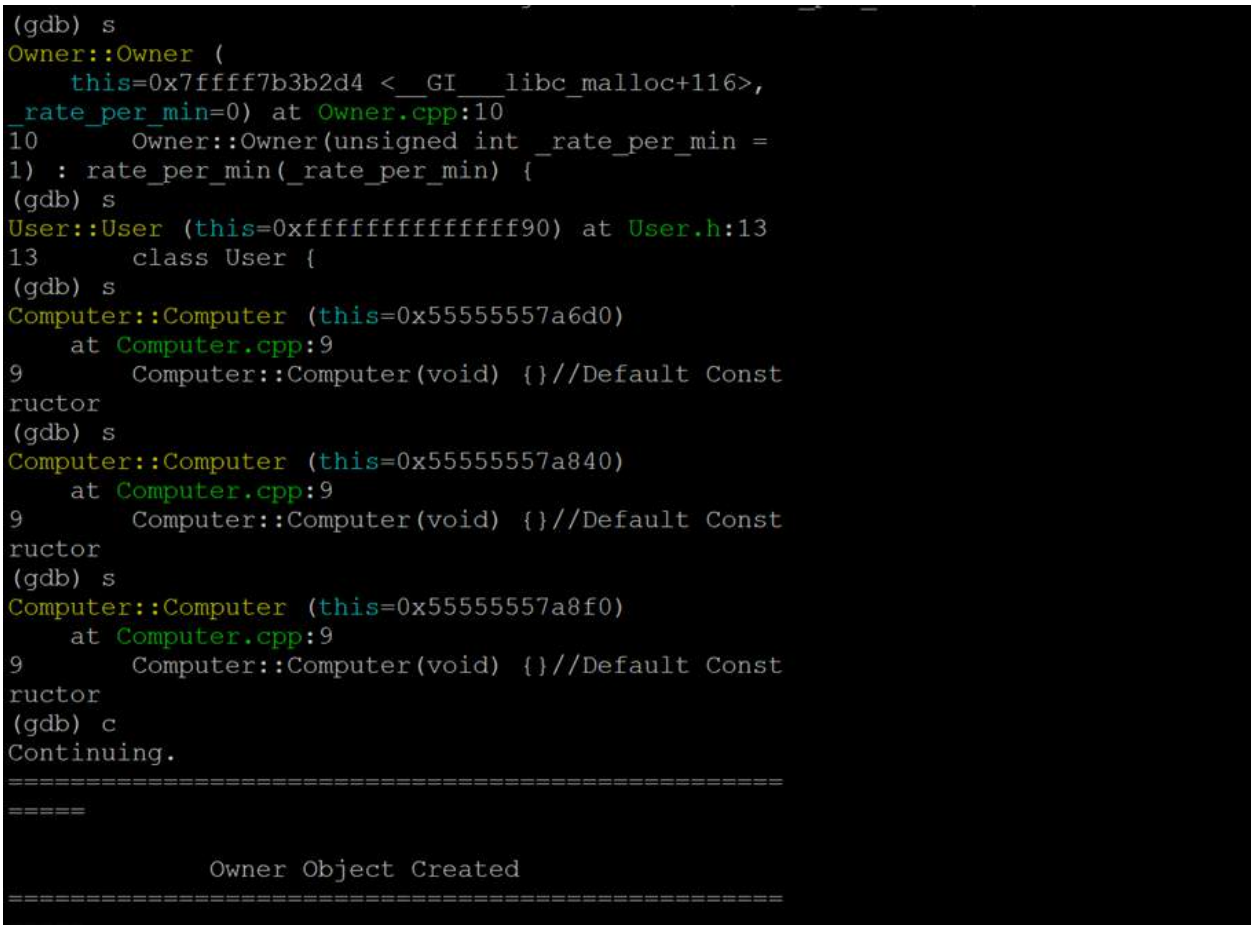
```
(gdb) run
Starting program: /home/spark/Cyber1/CCMS/a.out
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
=======================================================
              Welcome to Cyber Cafe
=======================================================
              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=0x7fffffffe3e8, envp=0x7fffffffe3f8) at CyberCafeManagementSystemMain.cpp:58
warning: Source file is more recent than executable.
58                                  if (_loginOwner()) {
(gdb)
```

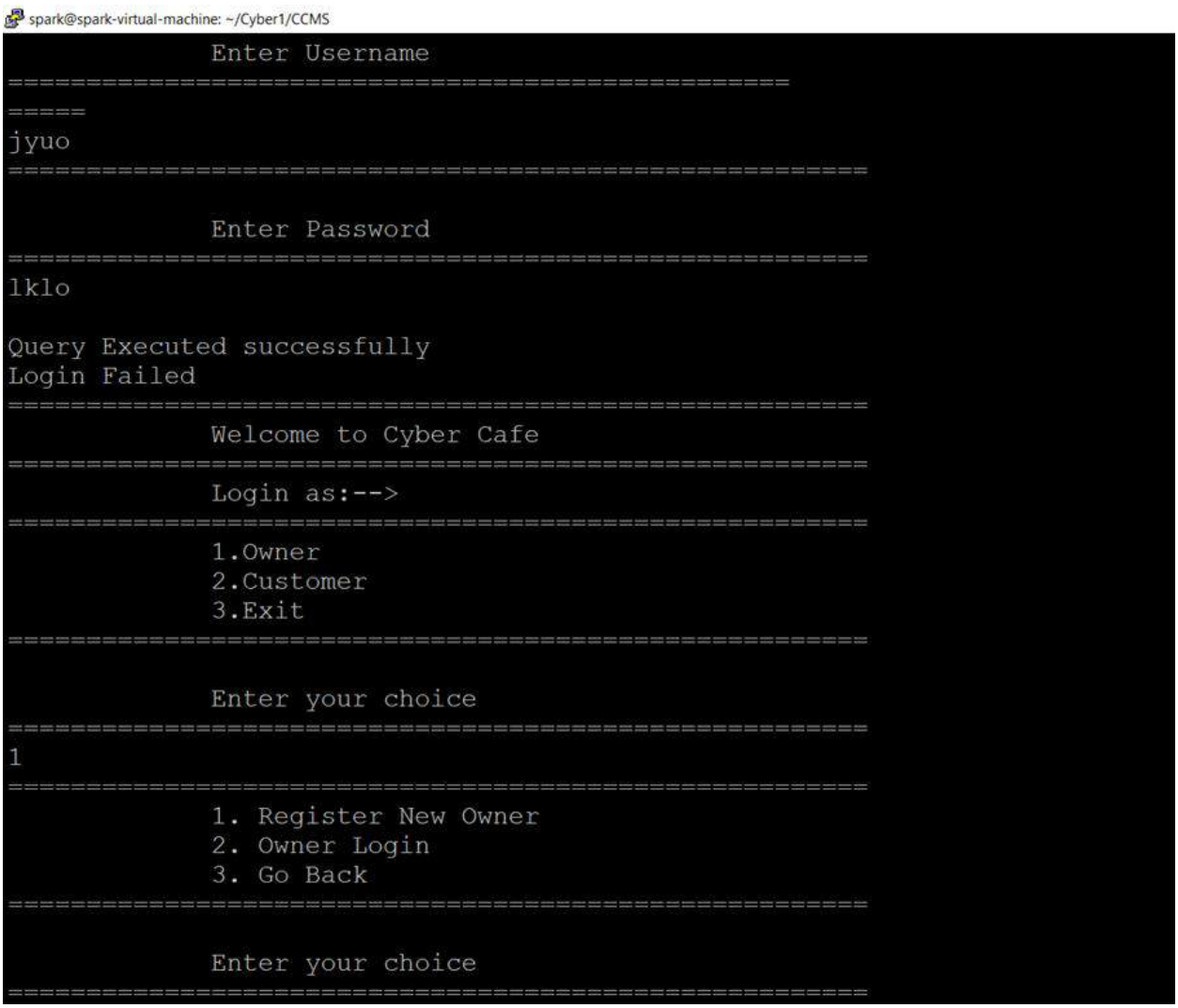
5. We use the command "s" to step over.

```
(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 << "=================================================" << endl;
(gdb) s
=================================================
37                              cout << "\n             Enter the rate for today" << endl;
(gdb) s

            Enter the rate for today
38                              cout << "=================================================" << endl;
(gdb) s
=================================================
39                              cin >> rate_per_minute;
(gdb) s
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.



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



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=0x7fffffffe3e8,
    envp=0x7fffffffe3f8)
    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.
0x00007ffff7f7e9c1 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.

```
(gdb) quit
spark@spark-virtual-machine:~/Cyber1/CCMS$
```

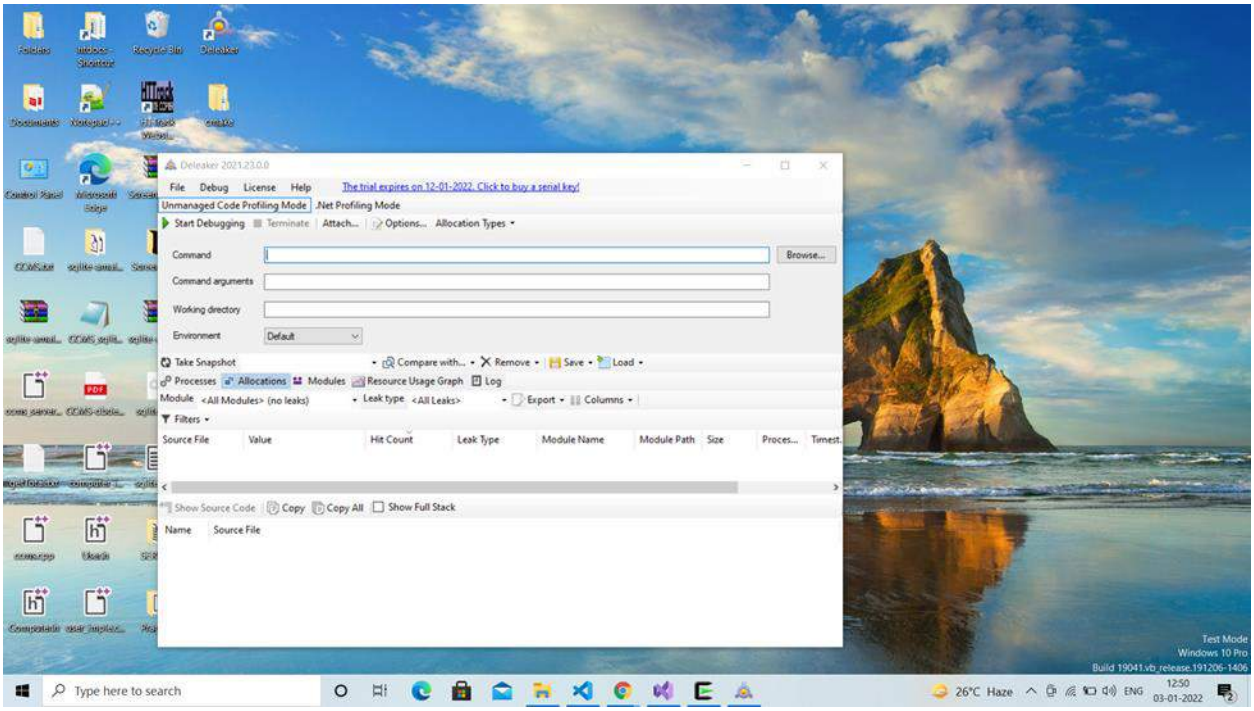
11. First time the rate_per_miniute was initialized to 2, but when login failed it was not reinitialized to 0. So, we set gpOwner to NULL and reinitialize the rate_per_minute to 0.

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

Memory Leak (Windows)

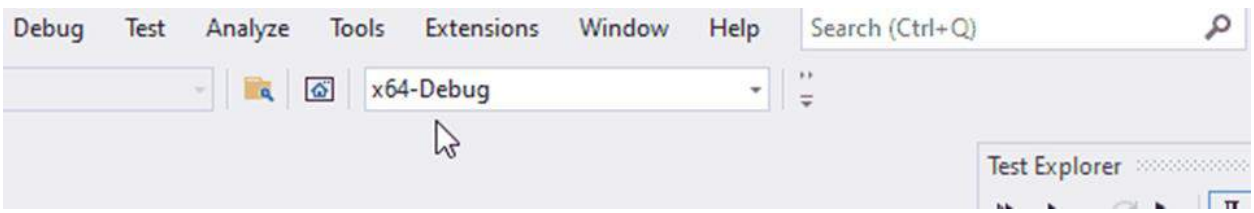
Using "Deleaker" windows Software for Memory Leak Detection.

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

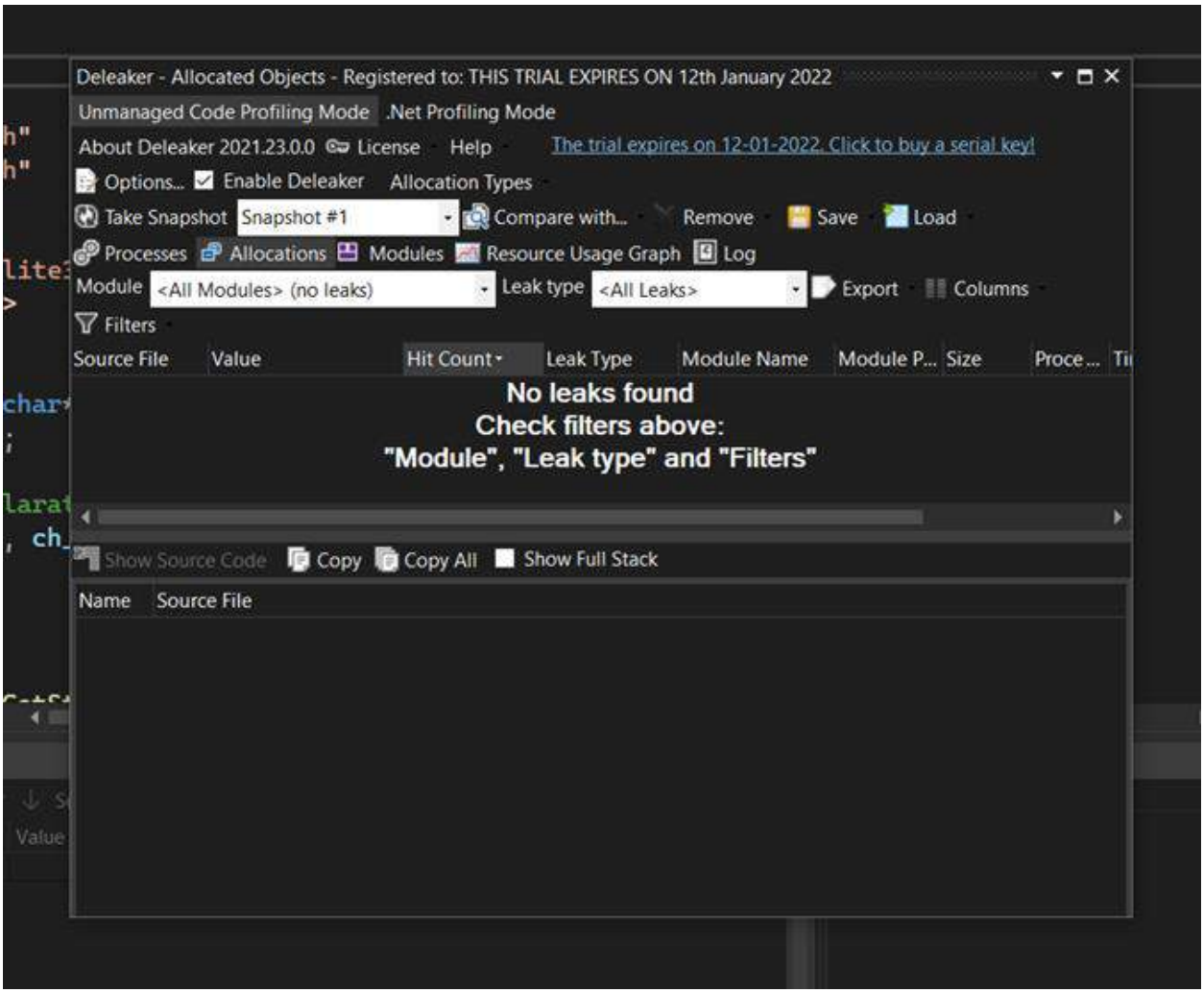


2) Once downloaded it will register itself automatically with the Visual studio.

3) Open Visual Studio. Go to Menu->Extensions->Deleaker Window.

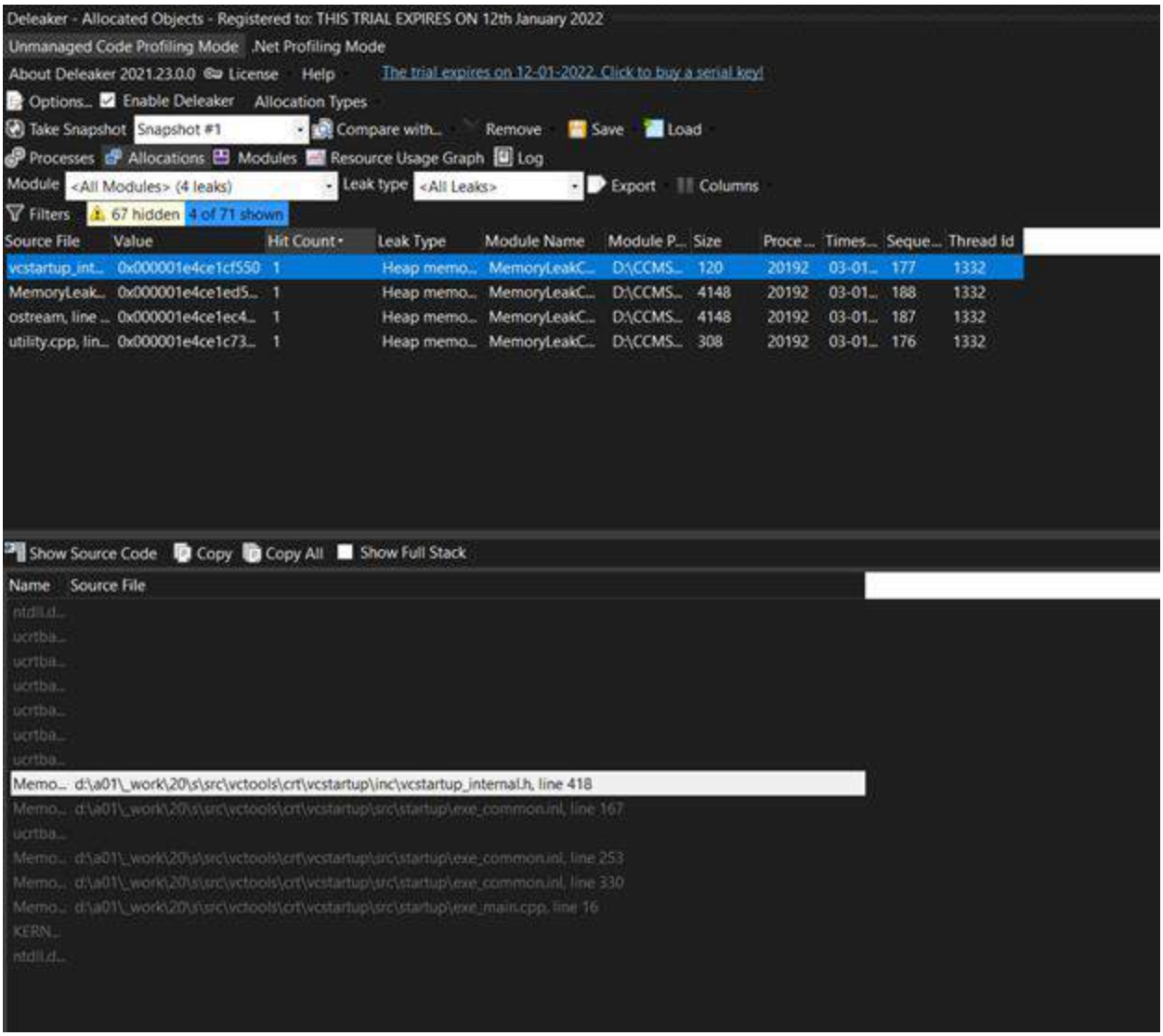


4) This window will appear.



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.



Memory Leak detection

(Open source)

2) Memory Leak Detection (Open Source) :

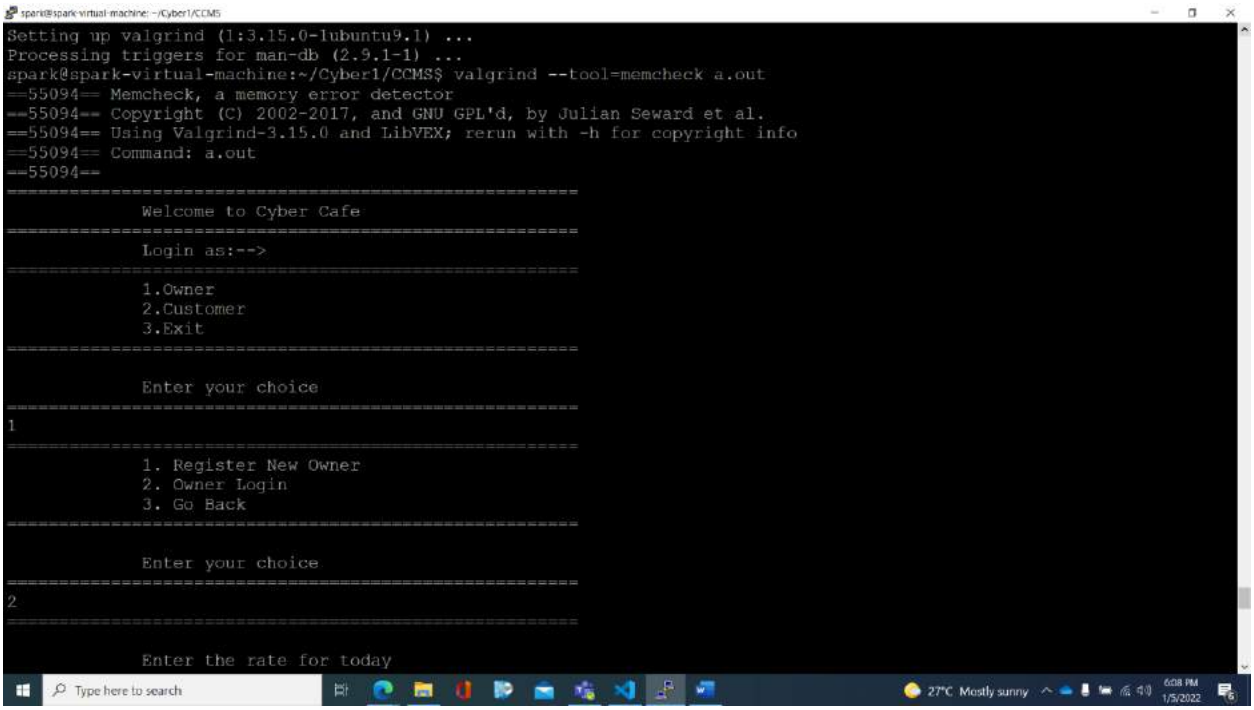
- Step 1 : Install “valgrind”.

  Sudo apt install valgrind

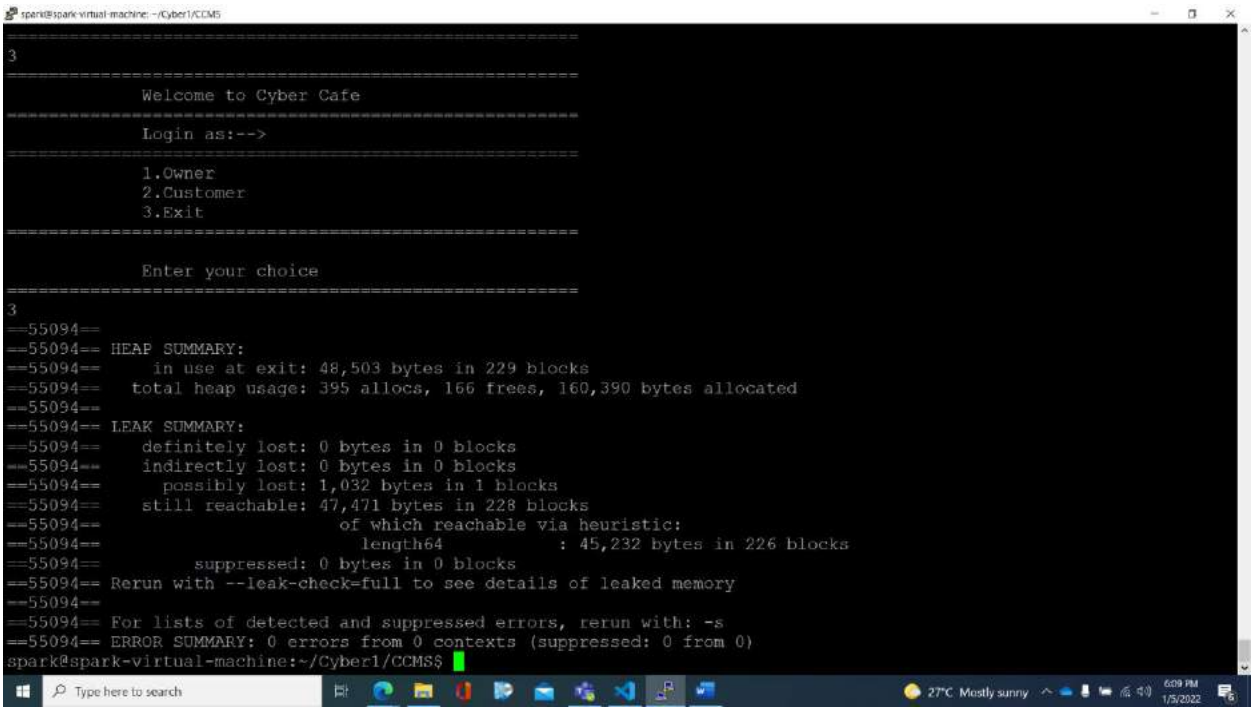
- Step 2 :

  Compile the code. And run the following command.

  valgrind –tool = memcheck a.out



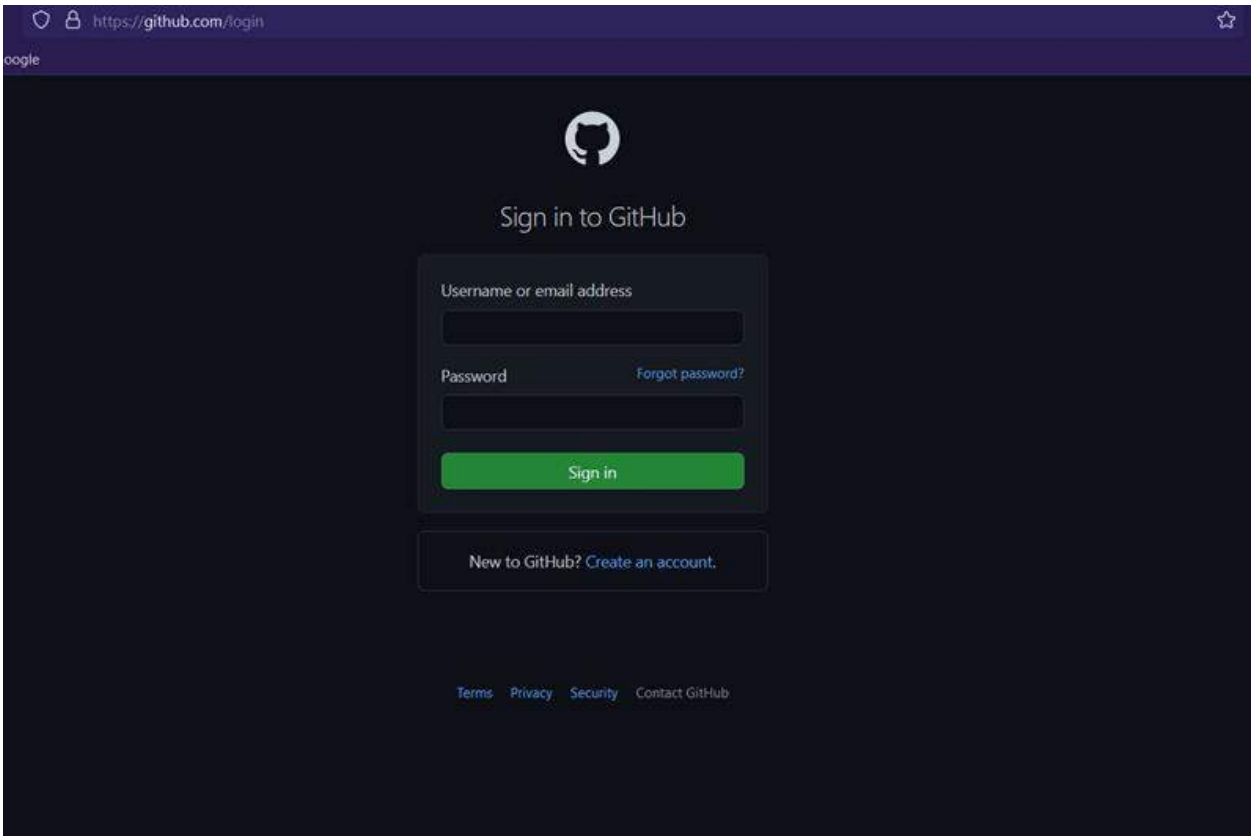
- Step 3 : At the exit of the code, we get the output of the memory leaked.


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$
Type here to search
27°C Mostly sunny
6:09 PM
1/5/2022

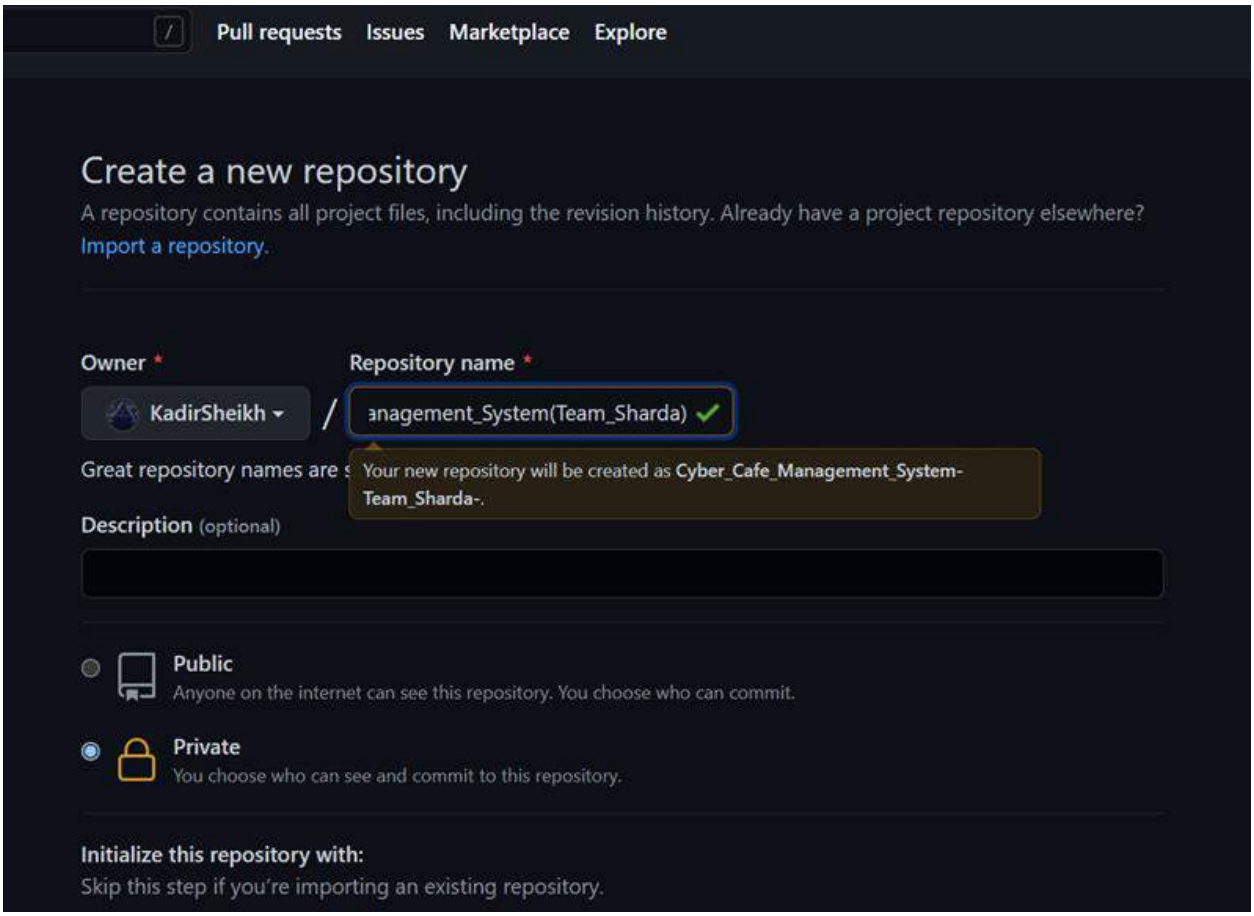
Milestone 5

Version Control (GitHub)

1) Login into github : visit github.com to login or signup.



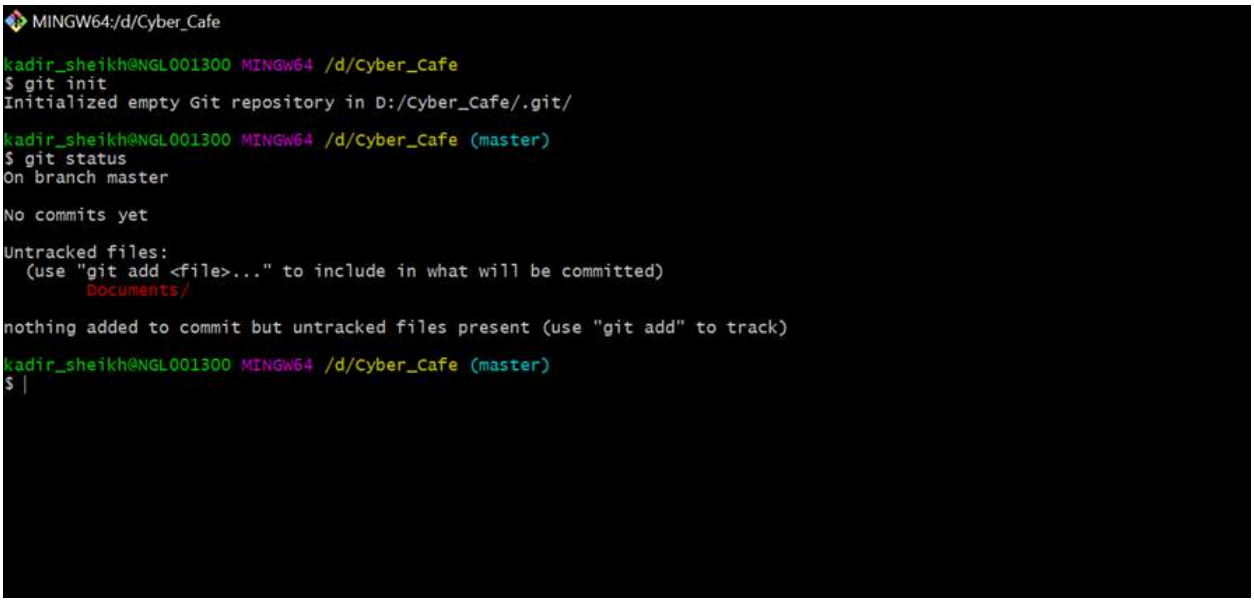
2) Creating Repository : Create a git repository



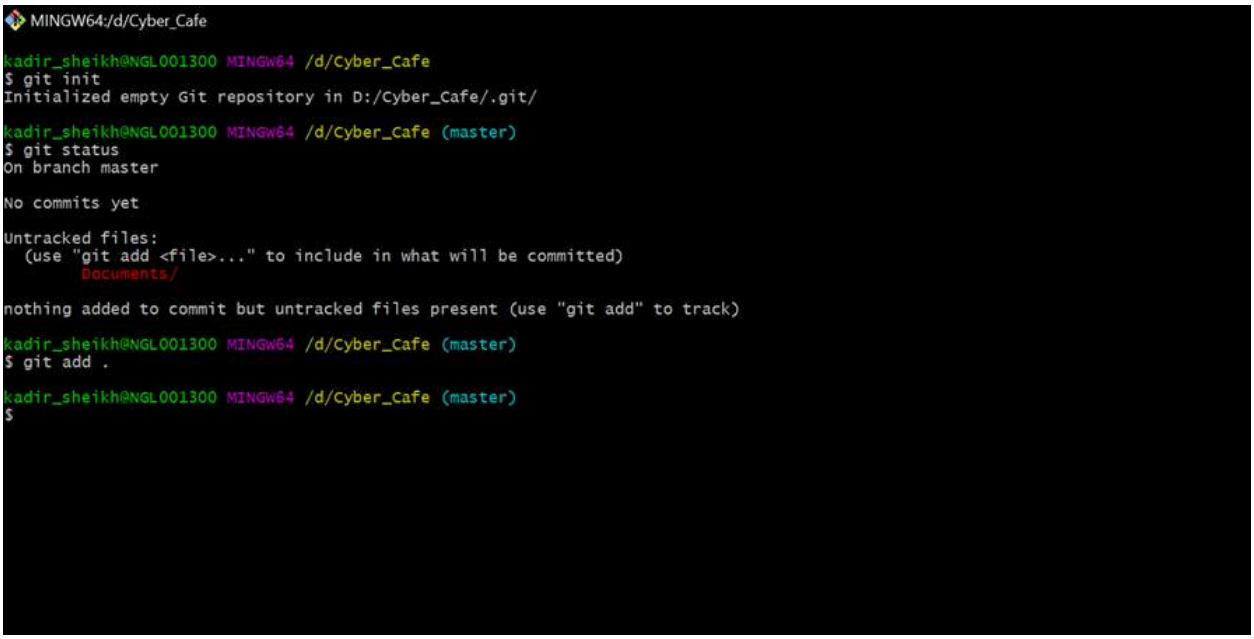
3)Inviting Teammates to collaborate :

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

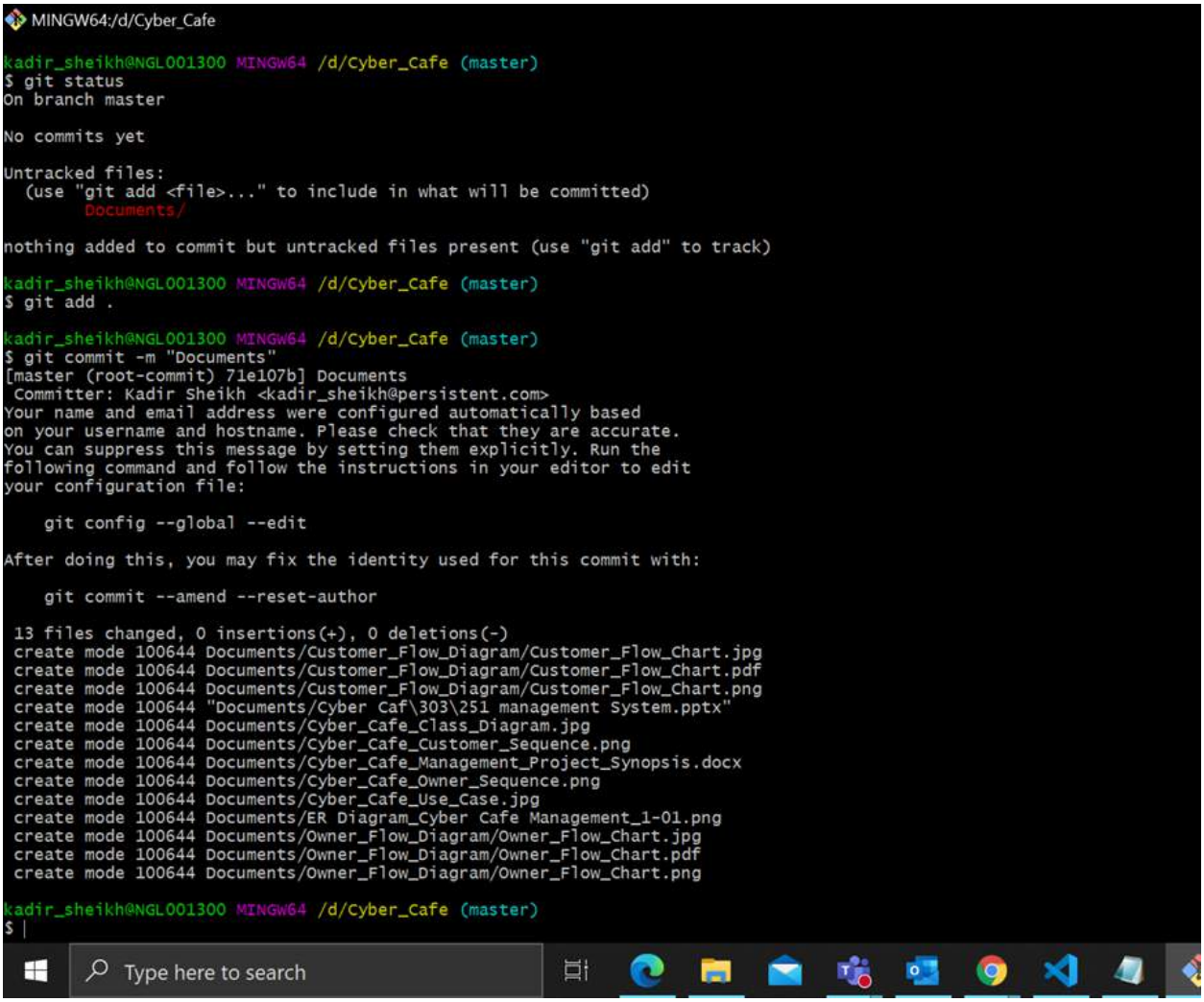
5)Use Git commands :



$ git status command to see untracked files.

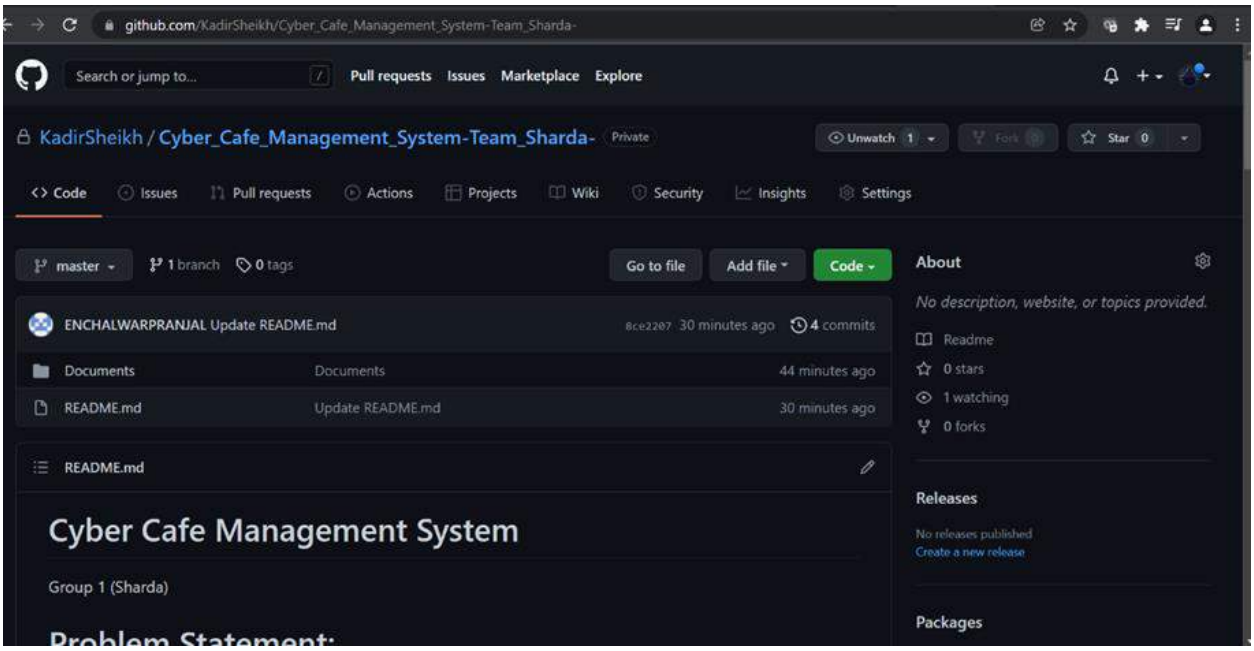


$ git add to track files.

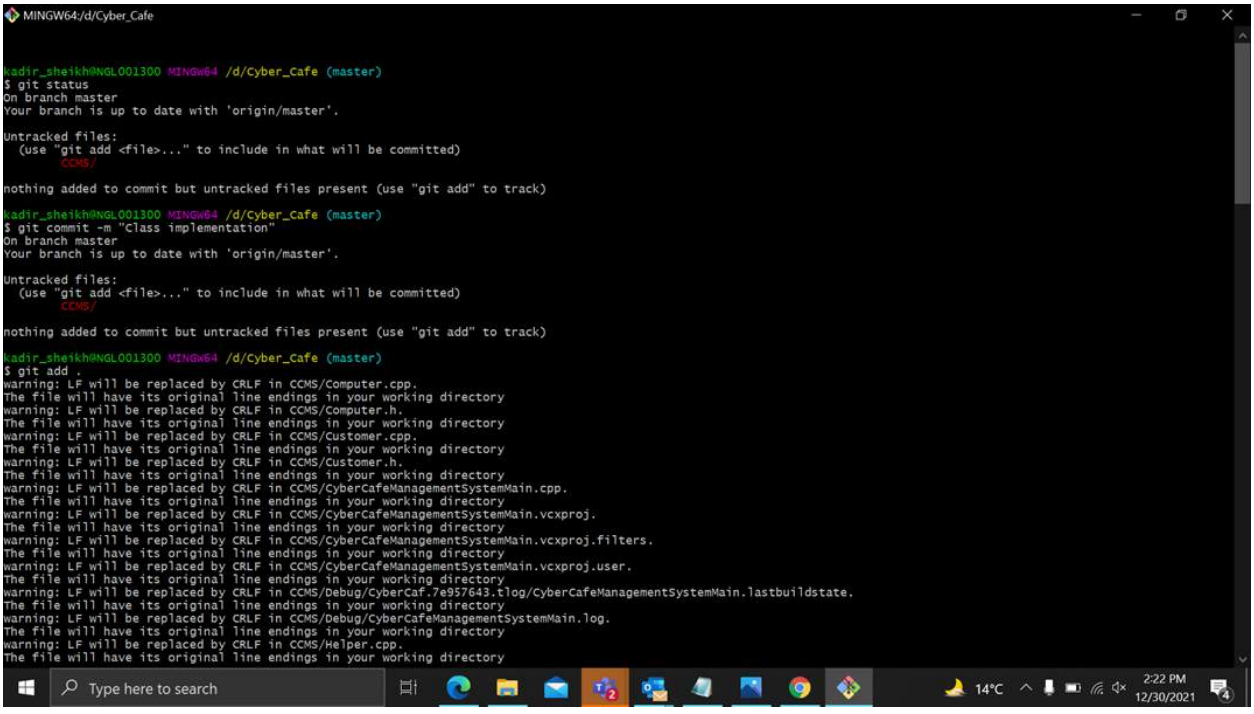


$ git commit to commit files on remote repository.

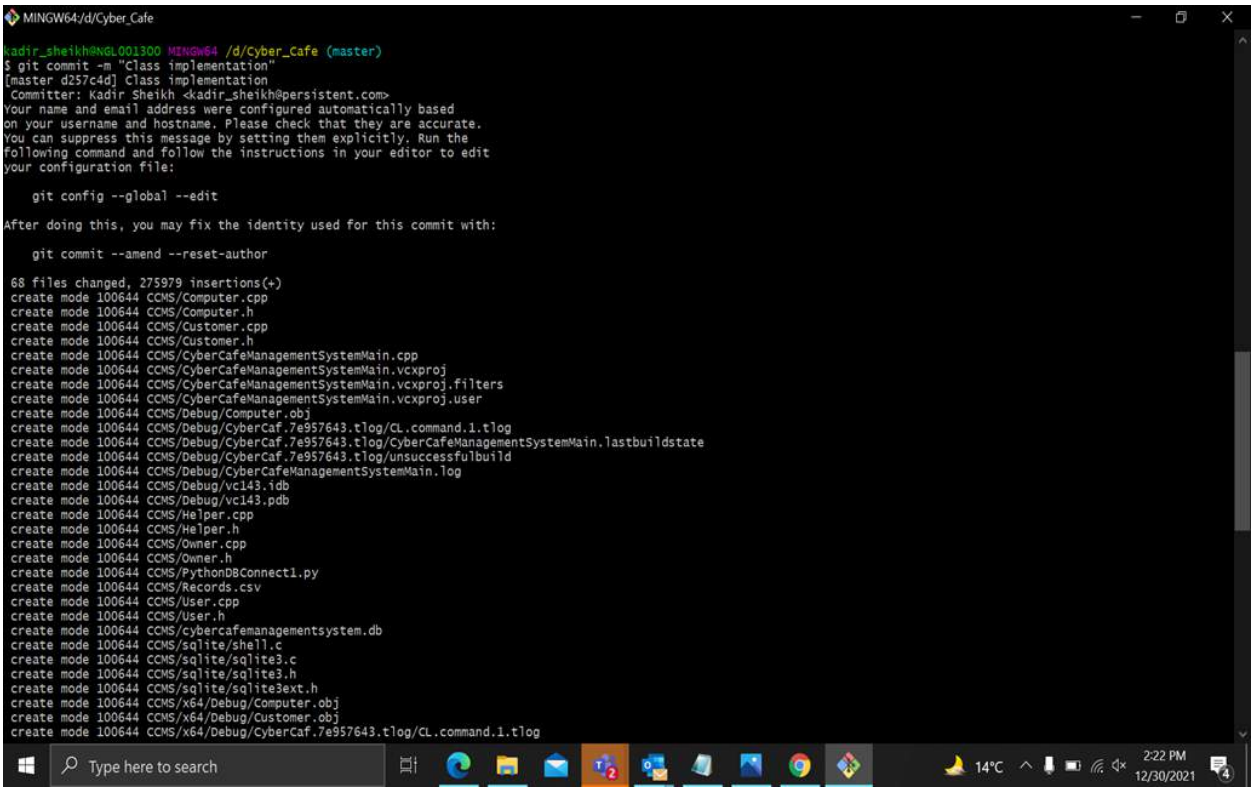
6)File Structure of Project on GitHub :



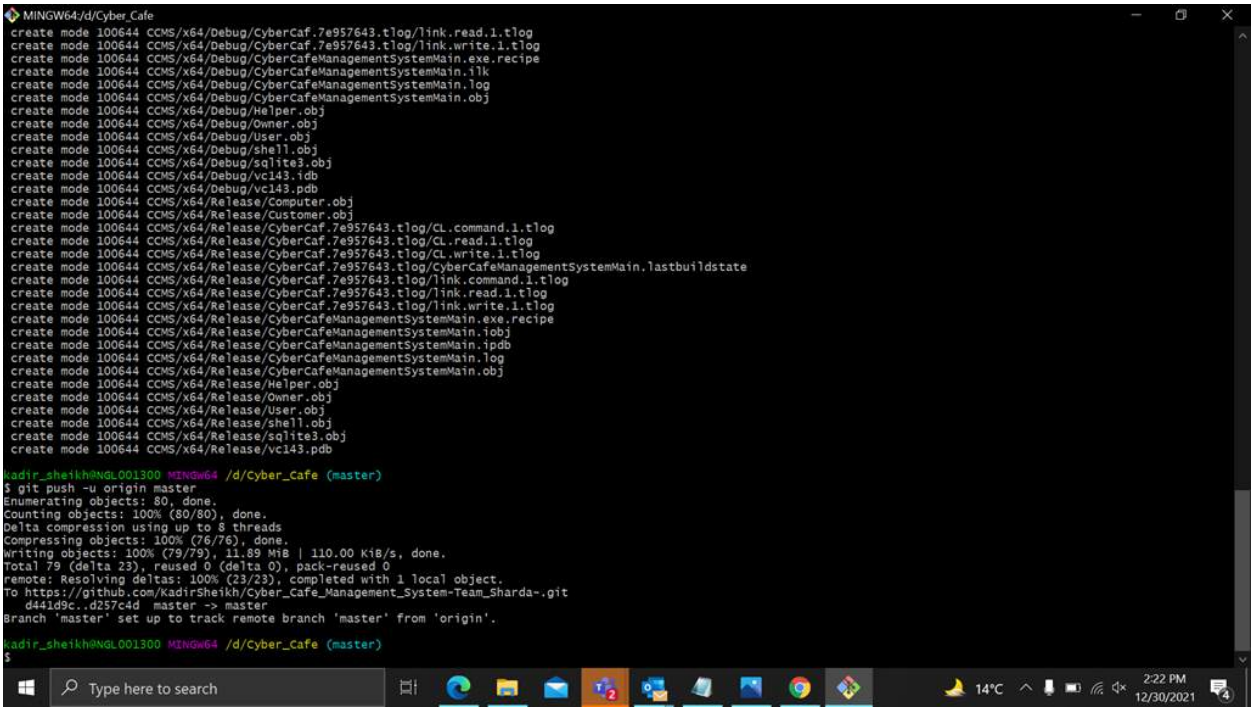
Checking untracked files using "git status" command



Adding “git add.” to track all the untracked files



git commit –m “message” to commit files on remote repository.



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.