Table of Contents

1. Introduction

This document aims to provide readers with fundamental knowledges needed to develop a console application using C++ programming language that involves data storage and manipulation including all CRUD (Create, Read, Update, Delete) process.

1. Project Setup

This section will explain the installation and configuration of the necessary software which will be required and used to develop the console application.

1.1. Visual Studio

Throughout this guide, the Microsoft Visual Studio IDE will be used to program, compile and run the C++ program. You can install the IDE by downloading the Visual Studio Installer through this link https://visualstudio.microsoft.com/downloads/ (15/10/2023) and select the Community Version which is free for students and non-profit uses.

Open the downloaded visual studio installer and go to the available tab and install Visual Studio Community and **ensure that you have the necessary components (Desktop Development with C++, Universal Windows Platform Development) selected** in the workload tab.
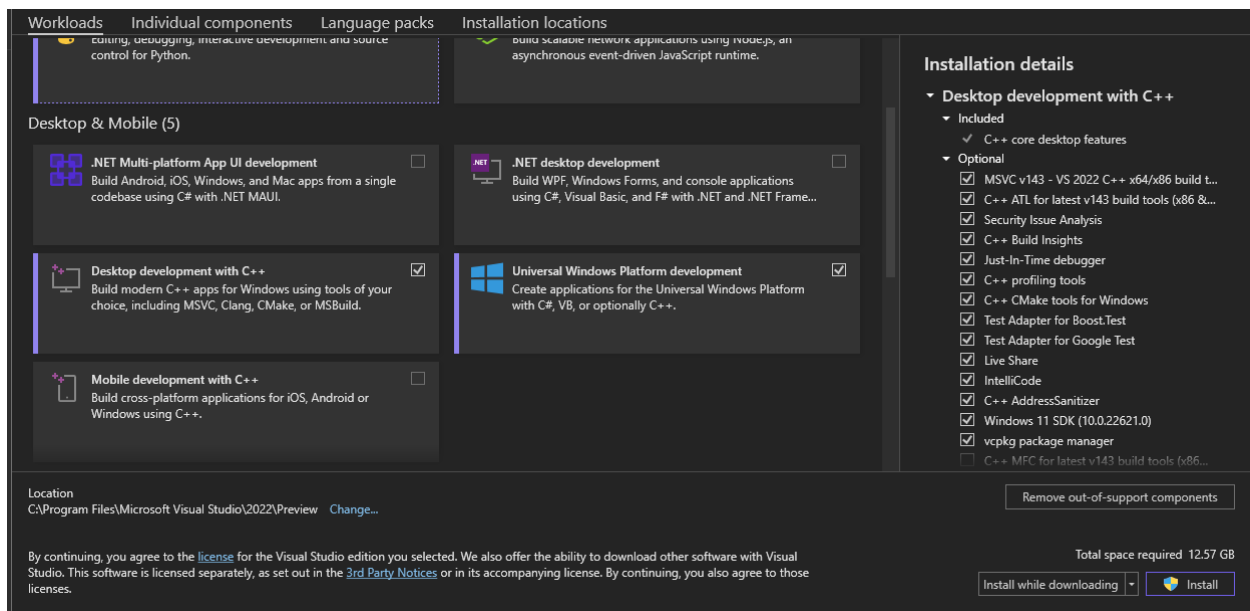

Other installation parameters, path, language etc. is up to your personal preferences. Proceed with the installation and wait until it finishes.

1.2. MySQL Database

As mentioned previously, the system to be developed throughout this guide will store and manipulate data. These data will have to be stored in a RDBMS (Relational Database Management System). There are variety of database option that can be used, in this case the MySQL or more specifically its variation MariaDB will be used.

1.2.1. XAMPP

XAMPP stands for X-Operating System, Apache, MySQL, PHP and Perl. It is an open-source web server solution package which consists of multiple available components. In short it includes multiple services. However, to develop a simple C++ console application, we will be only using two of the services which is the Apache and MySQL. You can get the installer for XAMPP from Apache official page https://www.apachefriends.org/download.html (15/10/2023). Download the installer and follow the instructions to install it. **You may install all component if you want but ensure that the following component (MySQL and phpMyAdmin) is selected** since they are compulsory to have for this project. Refer to https://www.youtube.com/watch?v=-f8N4FEQWyY&ab_channel=edureka%21 for a video guide on XAMPP installation if necessary.
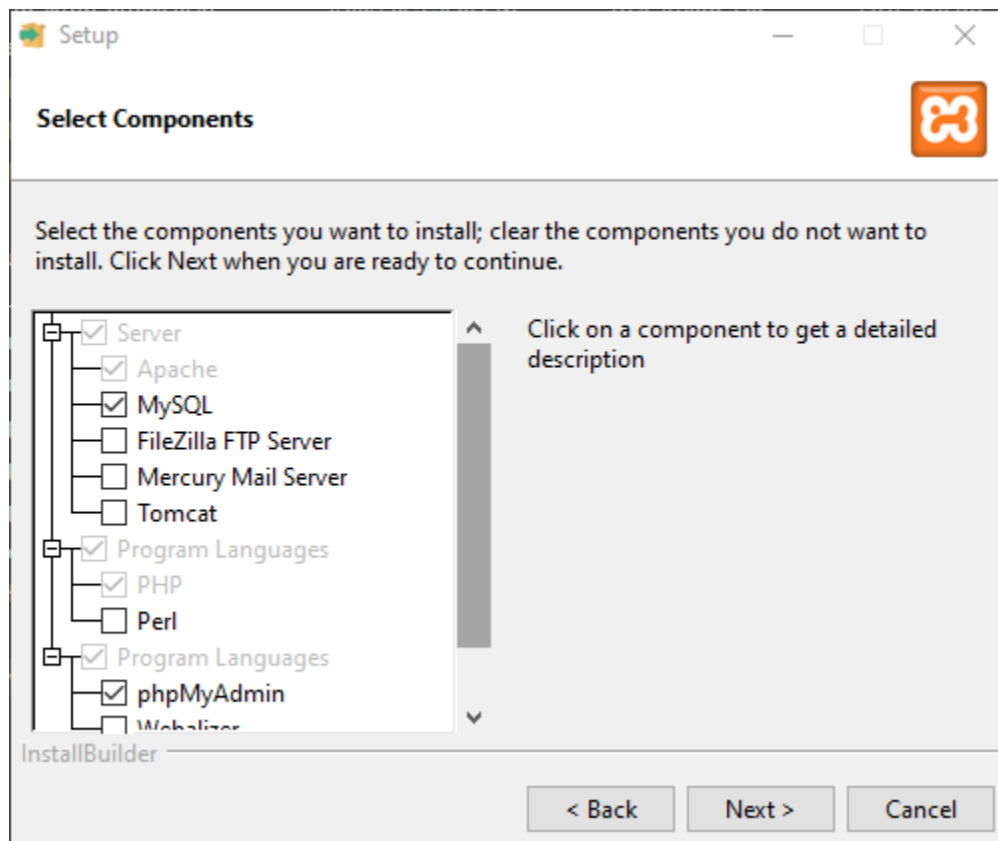
1.2.2. MySQL

Now that you have installed XAMPP and the necessary component in the previous section, you can start your database through the XAMPP control panel.
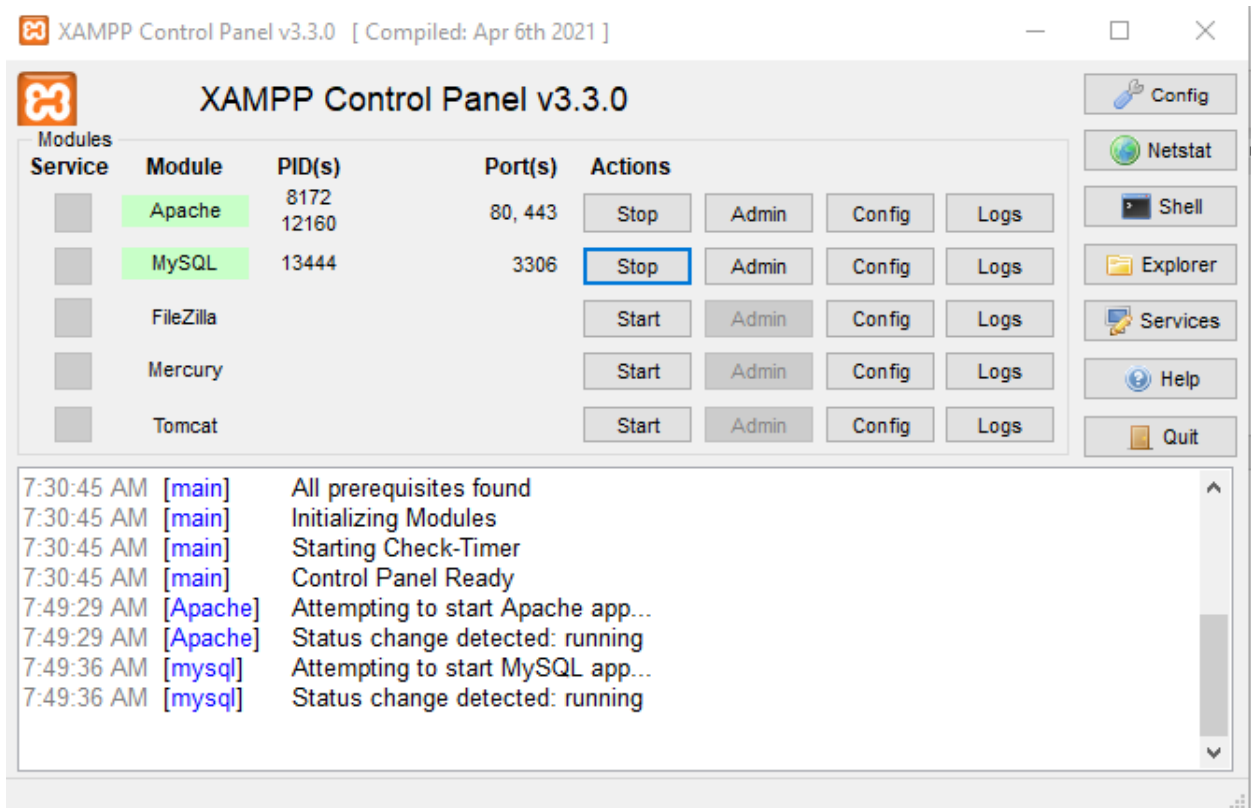

Again, there are multiple services provided by XAMPP. However, the most important service is MySQL which will be your database. Currently your database server will be running in your local server (localhost). You need to ensure that your MySQL is running whenever you want to run your C++ application later since it will need to connect to the MySQL server to perform data operations. Take note of the Port number being used by your MySQL, by default it should be 3306 unless you changed it yourself which you must use later in your C++ codes.

1.2.3. PHPMyAdmin

PHPMyAdmin is part of the XAMPP which provide user with GUI to manage their database easily instead of using CLI (Command Line Interface). In this guide we will only show basic usage of PHPMyAdmin to import the demo database used in the example project. You are advised to not to rely too much on the convenience of PHPMyAdmin which omits lot of manual queries. Try to understand the process of manipulating and managing database via queries.

To run PHPMyAdmin you must ensure that Apache service is running in your XAMPP control panel since PHPMyAdmin is basically a php web application that runs on the Apache server locally. Then, click the admin button in the MySQL row in the control panel to start it.
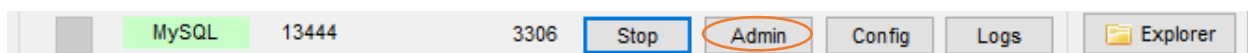
The following diagram shows the interface of PHPMyAdmin. On the left-hand side, you can see the list of databases already created on your MySQL. To create a new database simply click new (1) and then fill in the name (2) and click create (3).
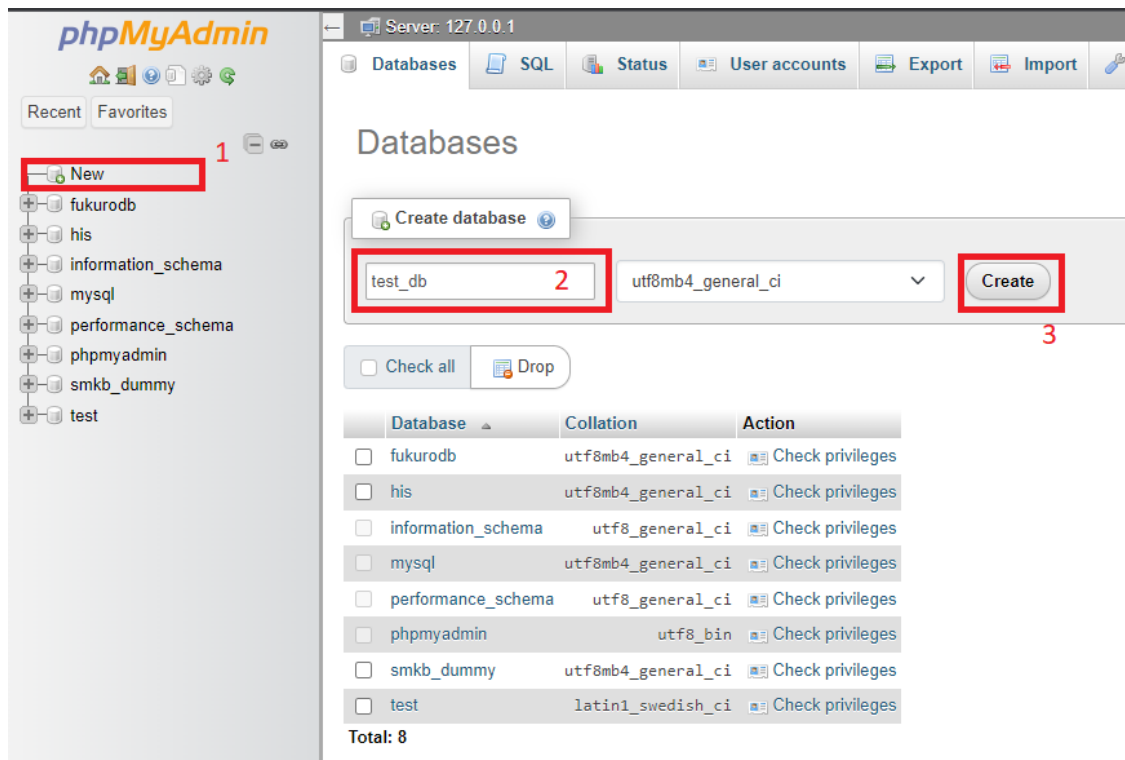
Then import the SQL file provided, by selecting your database, go to import and locate the file. Finally scroll down and find the import button.
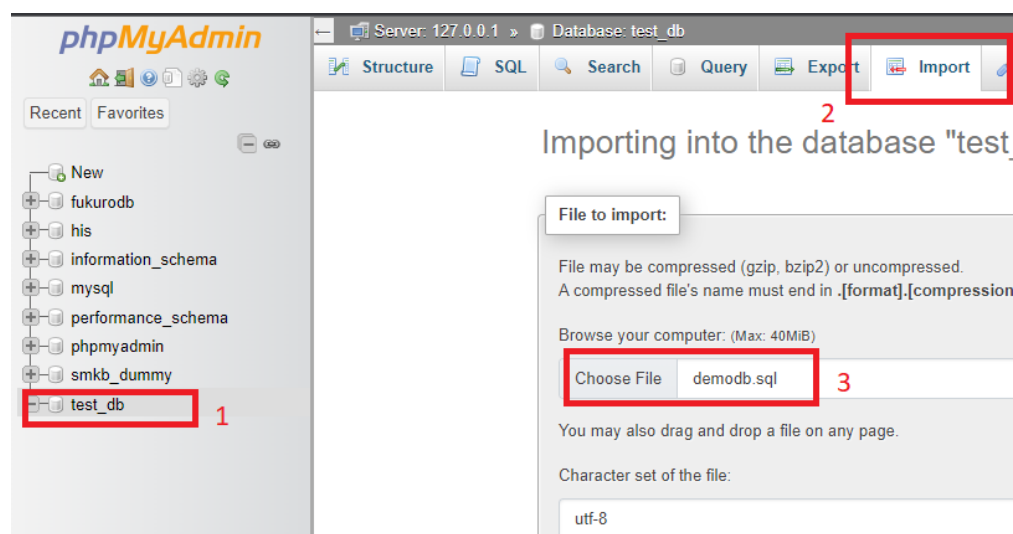
After you have imported the database, you can now proceed to follow through this guide. Other uses of PHPMyAdmin such as creating table, managing structure, constraints and data will not be further explained in this document.

1.3. MySQL Connector

After following the steps in 1.1 and 1.2, you should now have a Visual Studio Installed and XAMPP with its necessary components. This section will explain how to connect a C++ application with your MySQL database. The content of this section is similar to the explanation in https://www.youtube.com/watch?v=QsKnRk1gzxM&ab_channel=AmirulAsraf. Thus, you may refer to the video if necessary but do note that the video was recorded, and the version shown might not be same with what you will see.

C++ by default does not support interacting with MySQL database. Thus, we need to get an additional library which can fulfil this purpose. There is multiple option of libraries for C++ which allows database interaction which is mysql.h and others but in this guide, we will be using mysql/jdbc.h. The following are the step by steps explanation on how to download and import the library into your project.

1. Go to https://dev.mysql.com/downloads/connector/cpp/ or you can google MySQL C++ connector. Choose your operating system and download the connector file in ZIP archive.

2. Extract the downloaded file and make sure you remember where you extracted it since we need to locate it later. Basically, the file will consist of bin, include, and lib64 folder along with other files.
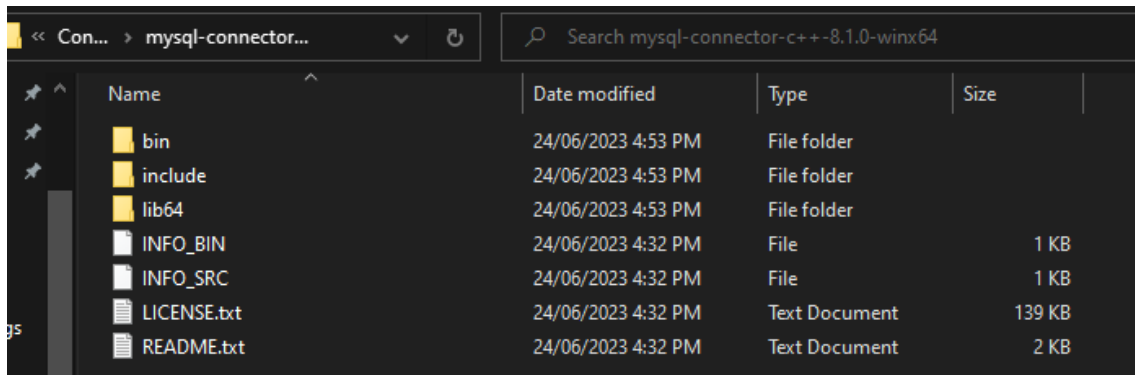
3. Now open Visual Studio and create your project.
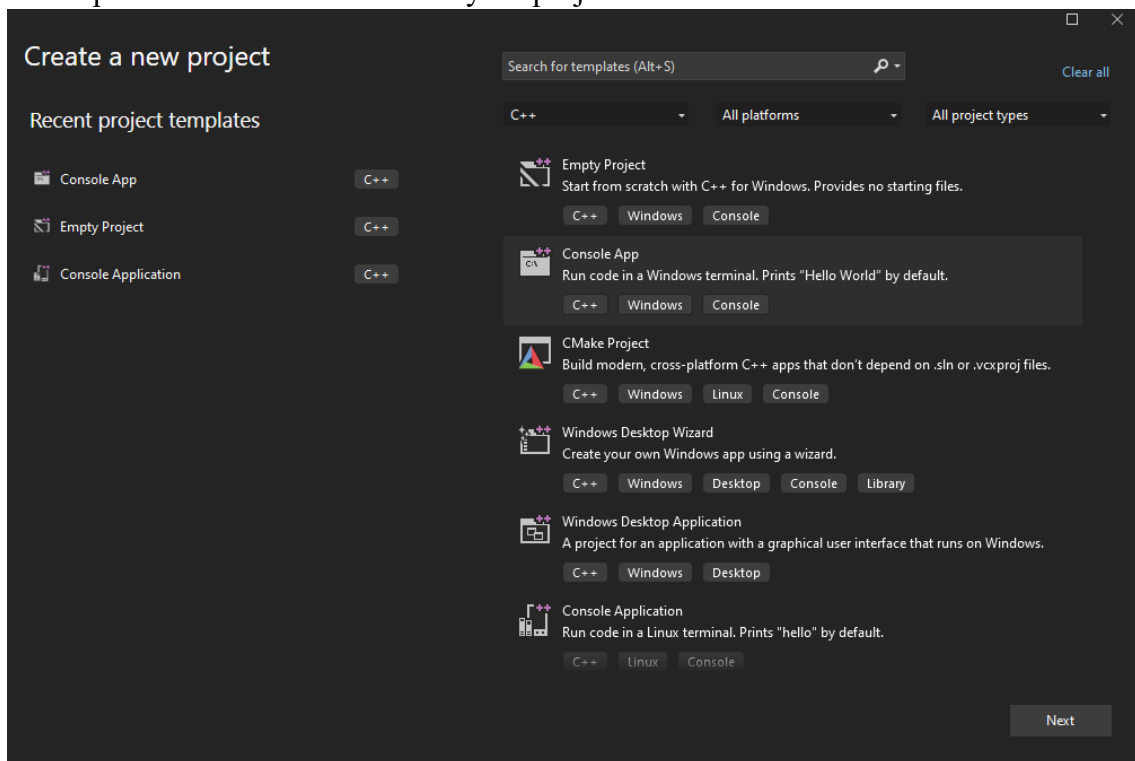
4. If you selected empty project, you might not see C/C++ option in your project property. In that case you need to create 1 .cpp file first in your project.

5. Go to your project properties via the solution explorer. Make sure you are right clicking the project name, not the solution since it will be different.
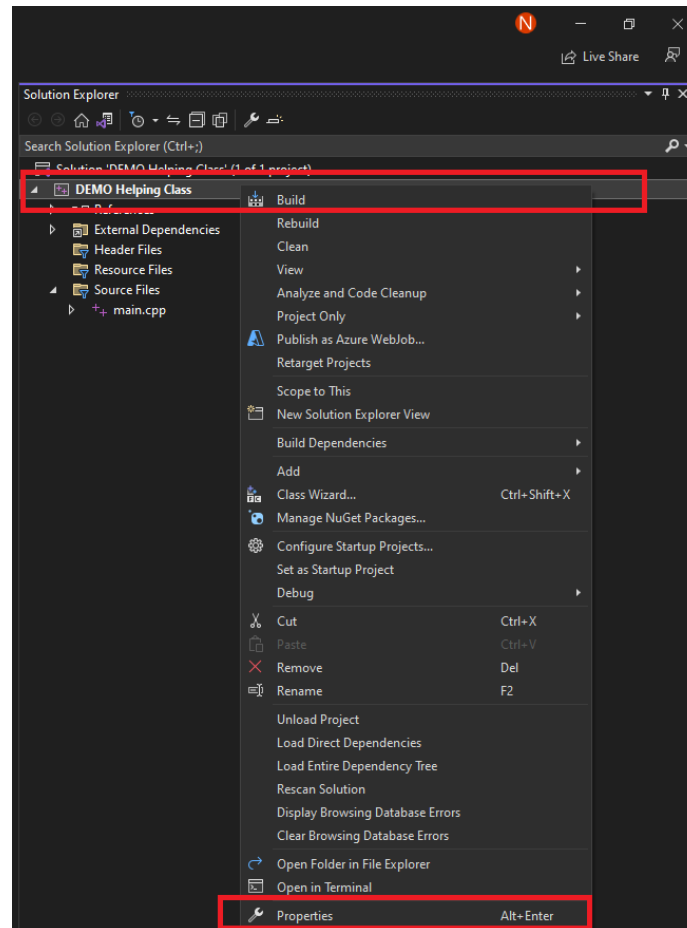
6. In your project properties, change the configuration to "release"
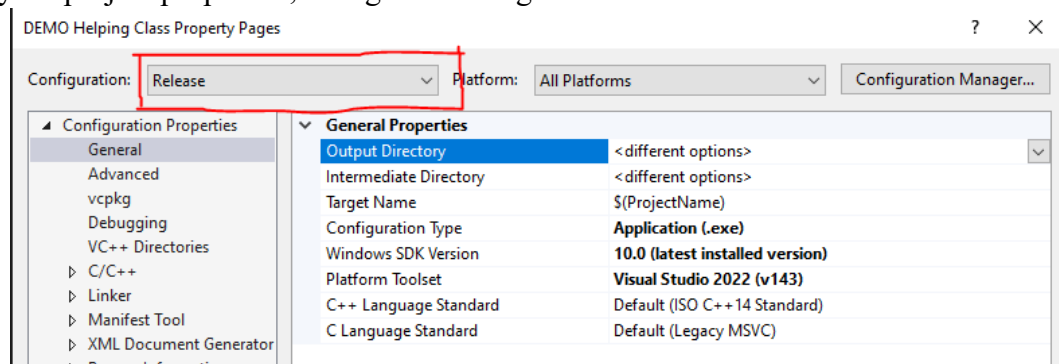
7. In your project properties, go to Configuration Properties > C/C++ > General. Then select the Additional Include Directories and click edit.
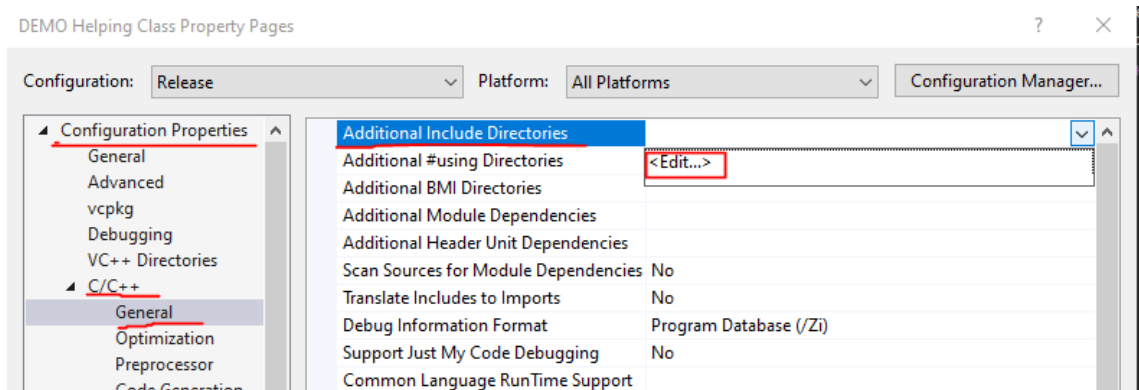
8. Then click on the empty row and lick the button at the right of it. Locate the folder which you extracted the ZIP files in step 2 previously.
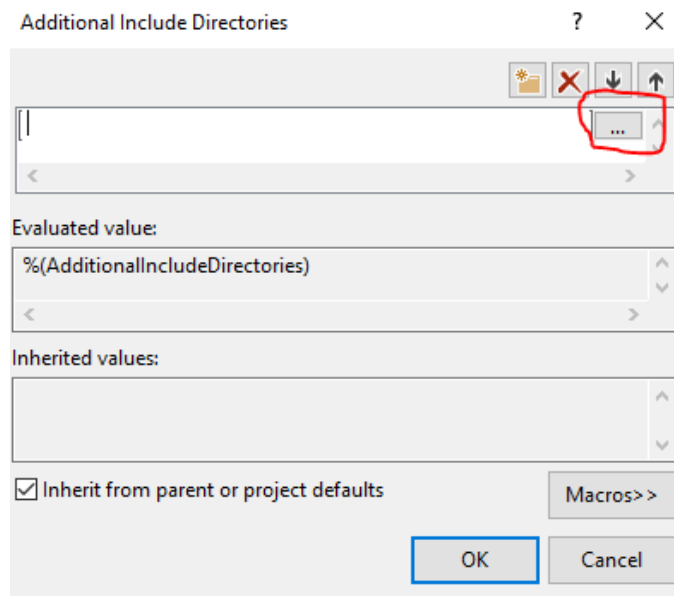
9. Select the "include" folder from your connector folder.
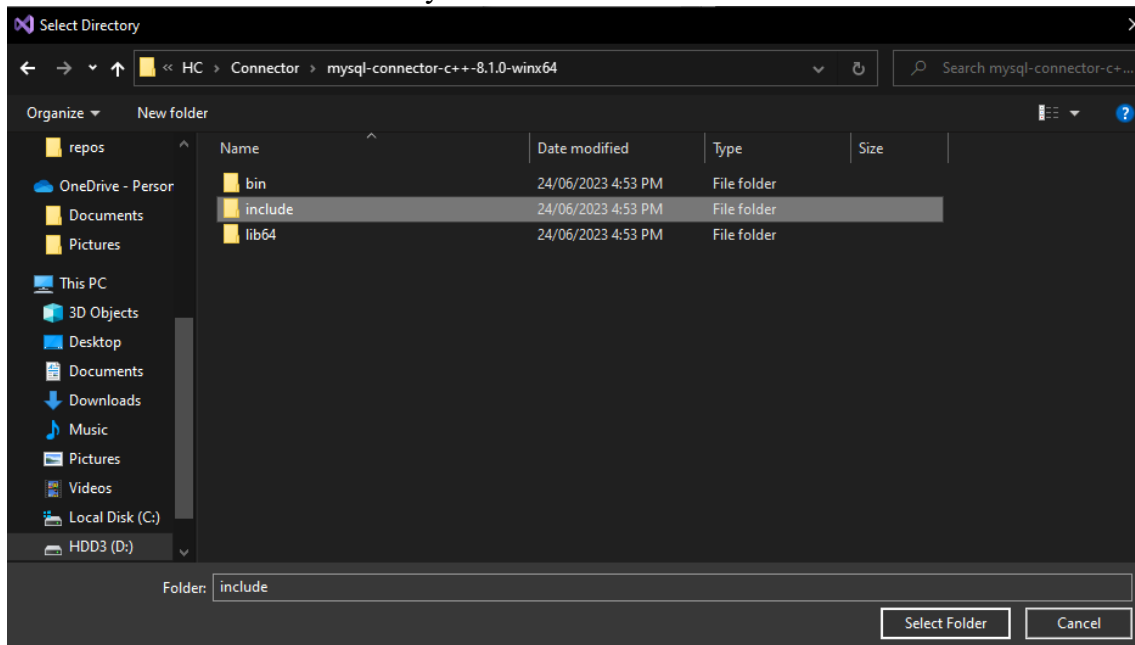
10. Press Oks. Now your project properties should look like this. Ensure that there is a value in the additional include directories.
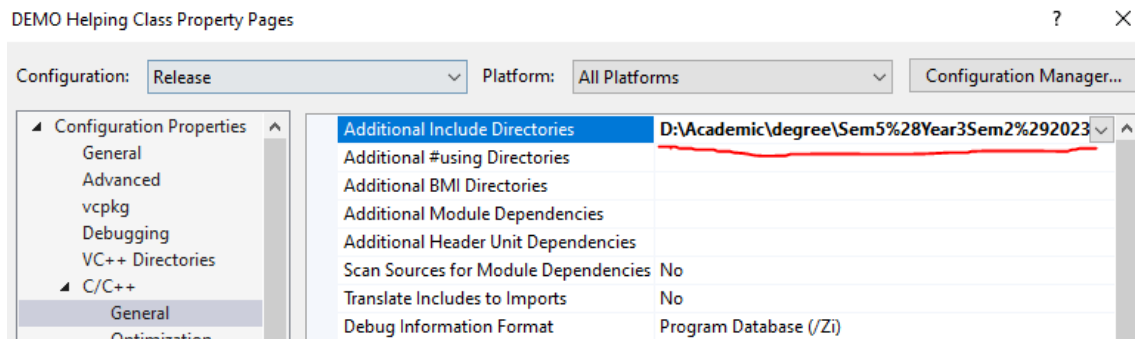
11. In your project properties go to Configuration Properties > Linker > General. Then select the Additional Library Directories to edit it.
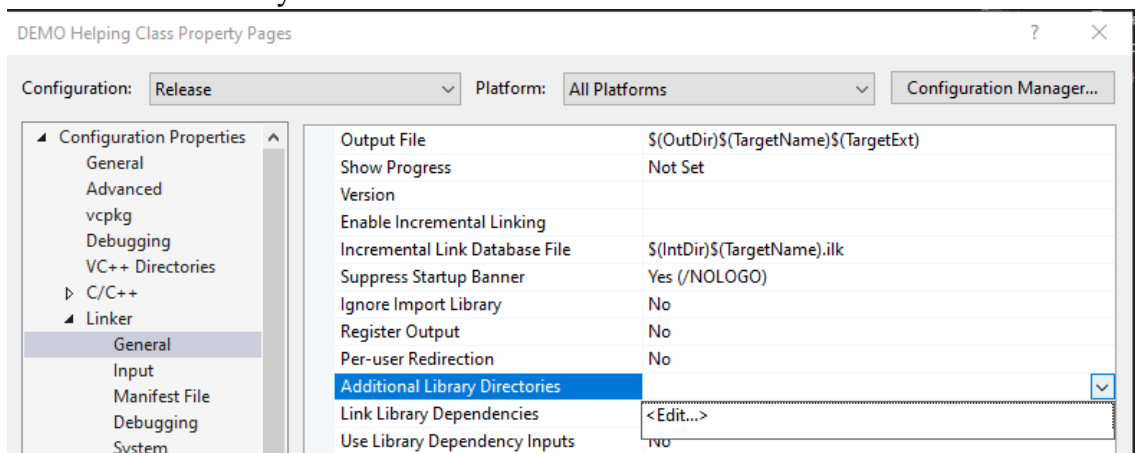
12. Then click on the empty row and lick the button at the right of it. Locate the folder which you extracted the ZIP files in step 2 previously.
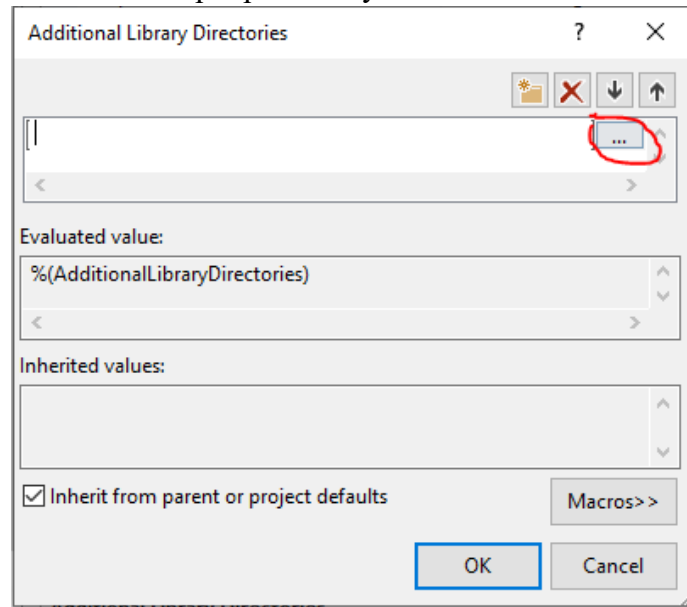
13. Select the lib folder from your connector folder.
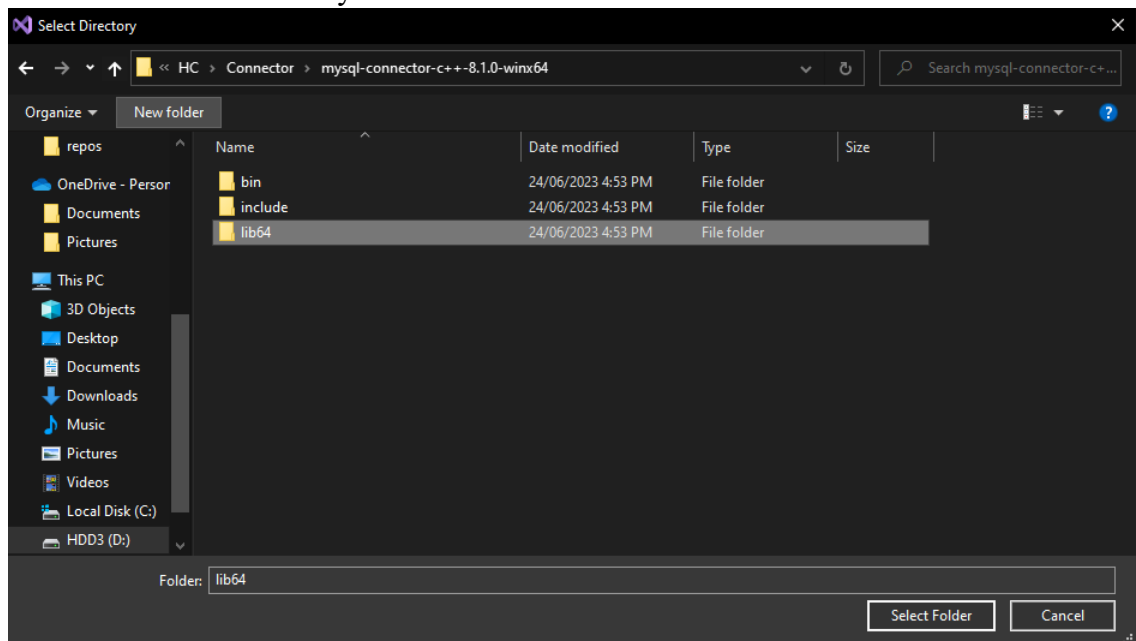
14. Click Oks. Now your project properties should look like this. Ensure that there is value in the Additional Library Directories.
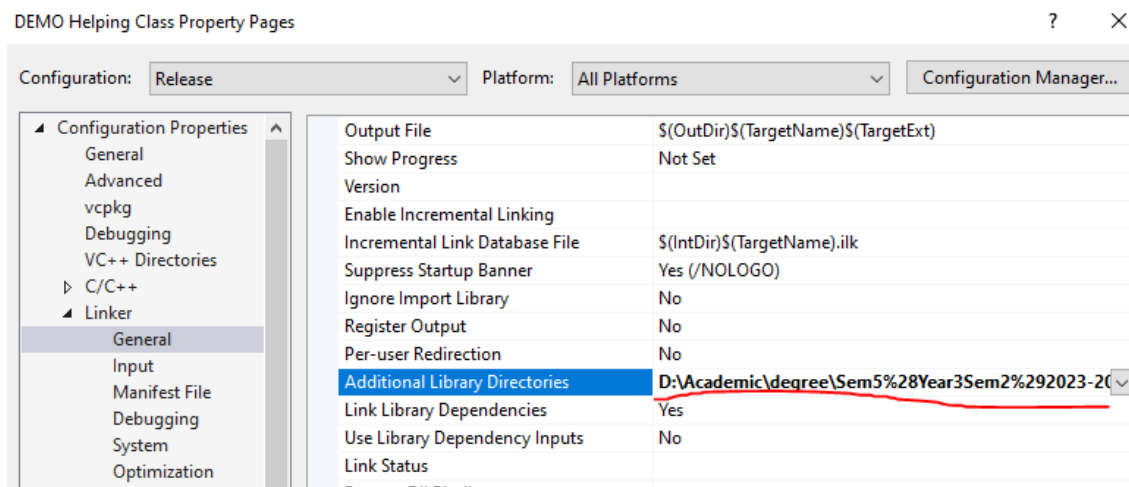
15. (optional) In this part we want to check the library dependency file name. By default, it will most likely be "mysqlcppconn.lib". But in case of MySQL themselves make changes you can confirm the file name by going to your connector folder/lib64/vs14
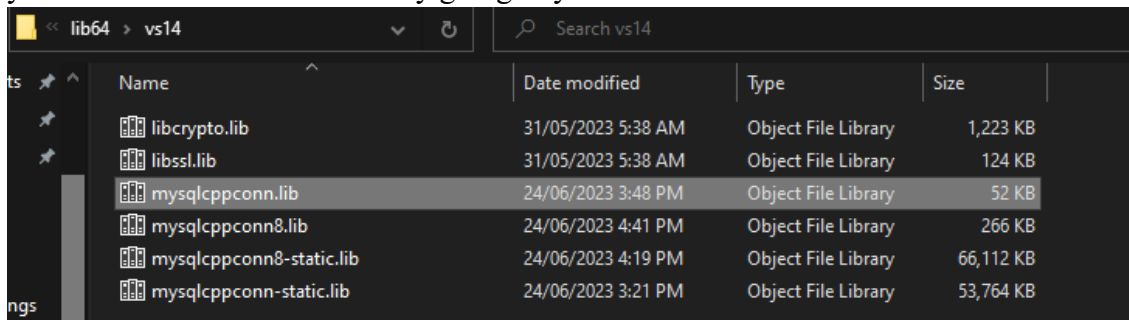
16. In your project properties. Go to Configuration Properties > Linker > Input. Select the Additional Dependencies to edit it.
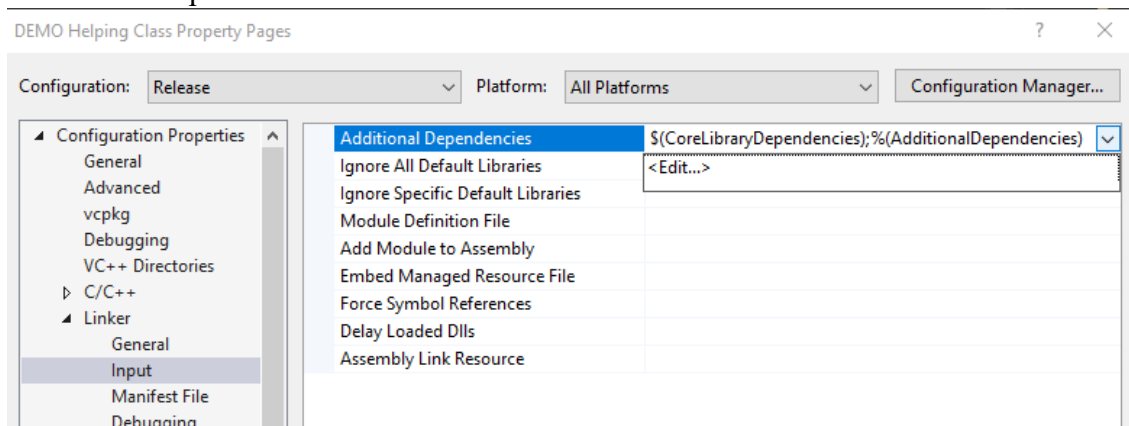
17. Then put "vs14/mysqlcppconn.lib". This basically tells your IDE which component of the library that you added into the linker just now you would like to use in your project which in this case is the mysqlcppconn.lib
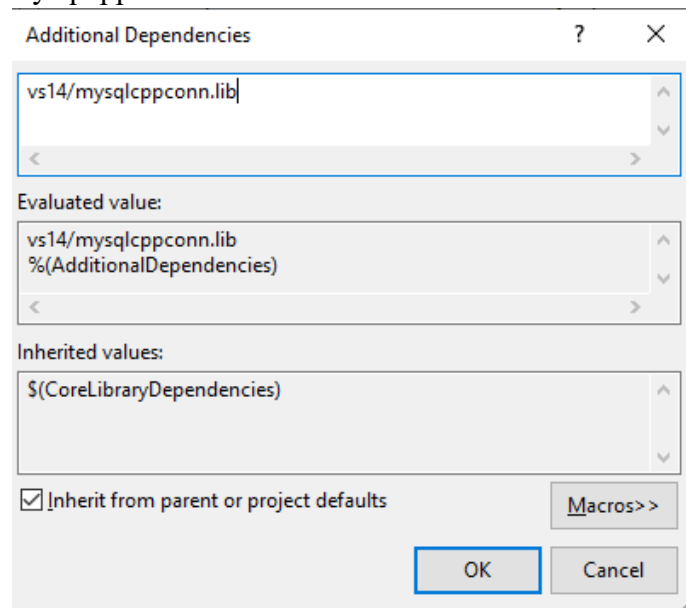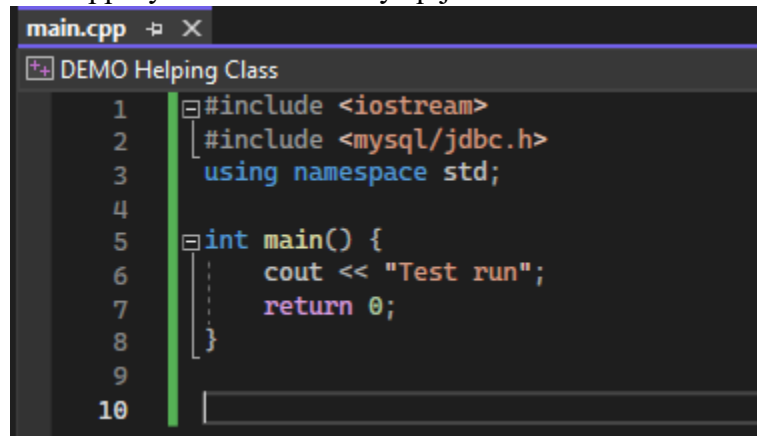
18. Press ok. Now your project properties should look like this.
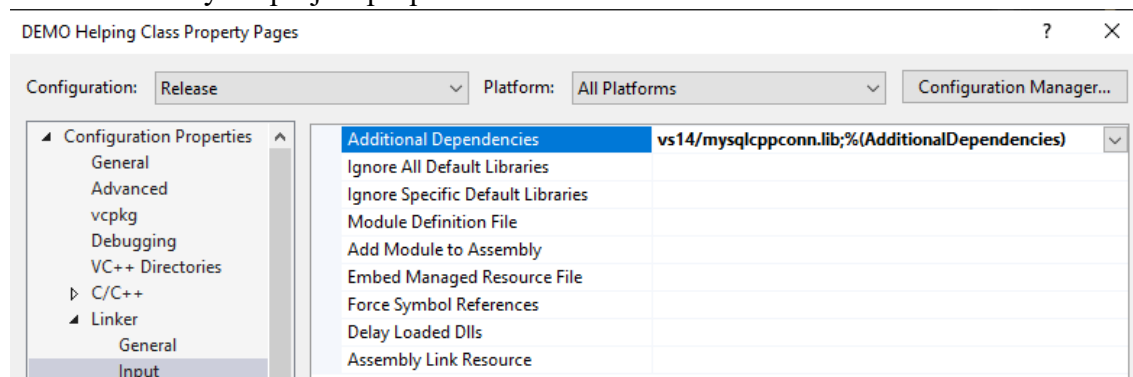
19. Click apply and then the configuration on project property part is done.
20. (optional) If you want to double check basically there is 3 important config to check which is the
    a. C/C++ > General > Additional Include Directories
    b. Linker > General > Additional Library Directories
    c. Linker > Input > Additional Dependencies

21. Now go to your main.cpp. Firstly, ensure that the development config at top of the screen is changed to release since your project properties is configured for release mode.
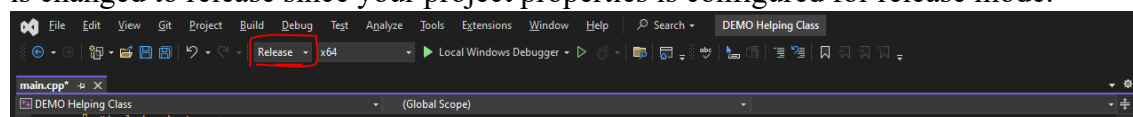
22. Then, in your main.cpp try to include the mysql/jdbc.h


```cpp
#include <iostream>
#include <mysql/jdbc.h>
using namespace std;

int main() {
    cout << "Test run";
    return 0;
}
```

23. Now if you can compile and run without error then it means you have successfully imported a third-party library and the mysql/jdbc.h is now ready to be used which will be explained in the next sections.

Note: moving/deleting or any changes on your connector folder might affect your C++ application when you recompile it which will result to your IDE unable to locate the necessary dependencies. Thus, put your connector folder somewhere you might not change frequently.

Also, importing other third-party library might follows similar steps which is adding entry in Additional Include Directories, Additional Library Directories, and Additional Dependencies. Thus, understanding this step could be helpful when you want to use other third-party libraries such as GNU plot integration to generate graph or other libraries.

2. Development

In this section, the implementation in form of coding will be explained in detail including the logic behind the codes to help you understand how to perform the fundamental database operations (CRUD) and implement it into your own specific project needs.

2.1. OOP in C++

This section will explain the implementation of an OOP (object-oriented programming) approach in C++ language. It will only cover the basic and fundamental usage of class which involves only the concept of class, object, attribute, method, and static member.

2.1.1. Class

Class definition in layman term would be a blueprint or concept that define the structure or characteristic of how an object which belongs to the class should be. It consists of the declaration of the attributes, method etc. which defines how the object instantiated from the class should behave in a programming context.

It is used to classify or group data and associates it logically, for example person have lot of data such as name, age, gender etc. Storing each of this data individually is not feasible since we can't identify which age is for which person with what name etc. With class we associate each of these attributes of a person into a single instance of object from class Person so we can manipulate it.

Analogically if the blueprint **(class)** of a house shows that the house will have 5 doors. Then when a house **(object)** is built **(instantiated)** according to the blueprint **(class)**, all of the houses **(object)** will have the same characteristics of having 5 doors.

In C++ project you can create class by right clicking the project in the solution explorer and choose Add > class.
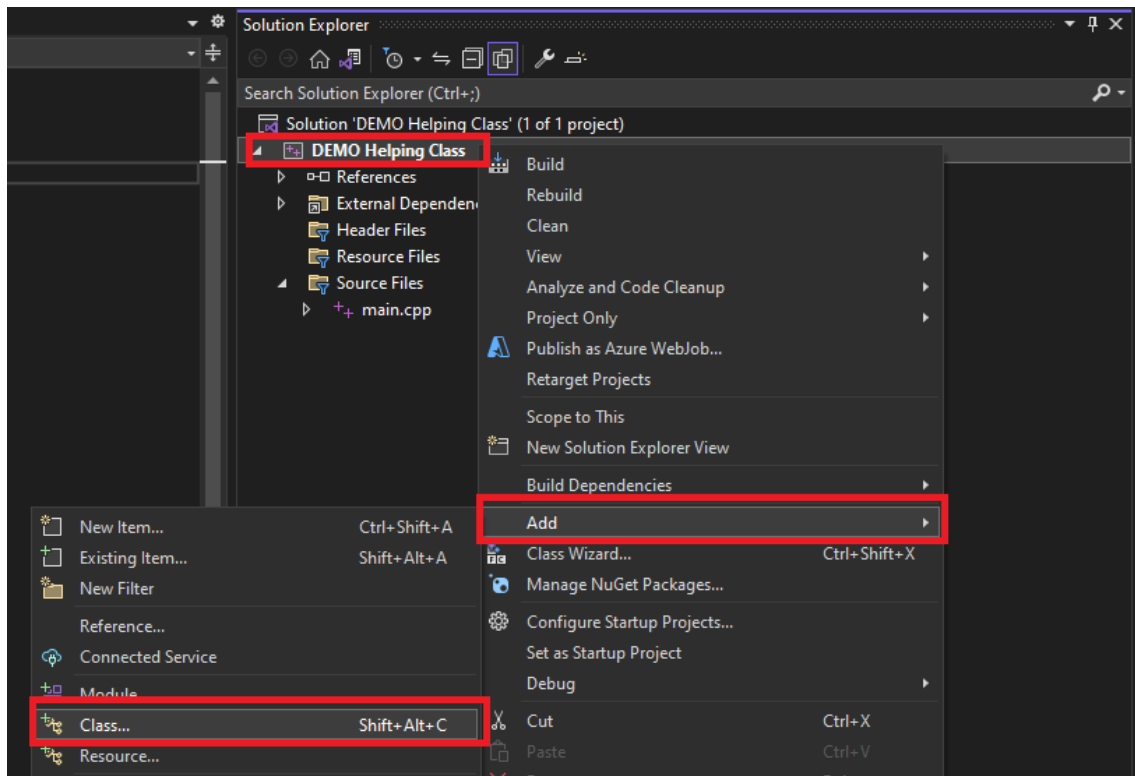
Next fill in your class name according to your needs. In this example we are creating the DBConnection class which will handle the database connection and transaction later. Although not compulsory, it is recommended to practice standardization and naming conventions while naming your classes. Basically, your class name should be related to what it is for or what it should do like in this example we give DBConnection to class that handles database connections and operation.

Besides that, it is also a good practice to standardize your naming convention by implementing standards such as Pascal Case. In Pascal Case naming, the name starts with capital later and if it contains multiple words each first letter of the words should be capitalized. In this case, the abbreviations of database, DB is in capital letter while the next word connection starts with the capital C. Again, it is not compulsory but is indeed a good practice to standardize your naming to ease readability and tracking your files and class later on.
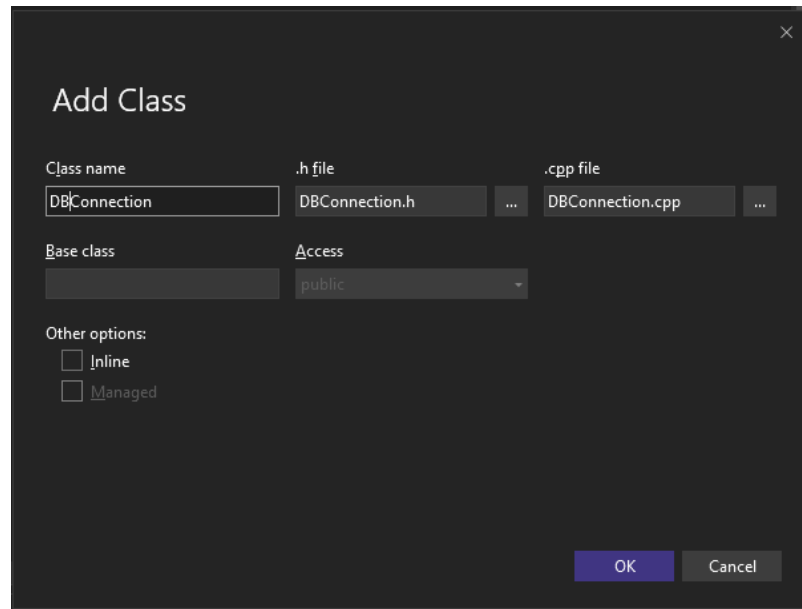
After clicking ok, Visual Studio will generate two files for your class which is the classname.h and classname.cpp. **Now create another class named "Account" which we will use to manage user account data.**

You should be able to see your created classes in your solution explorer.
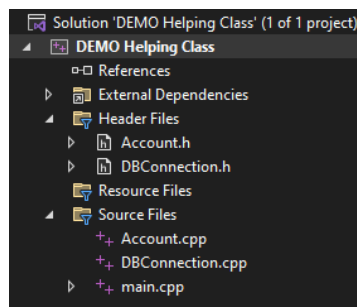
2.1.2. Class.h file
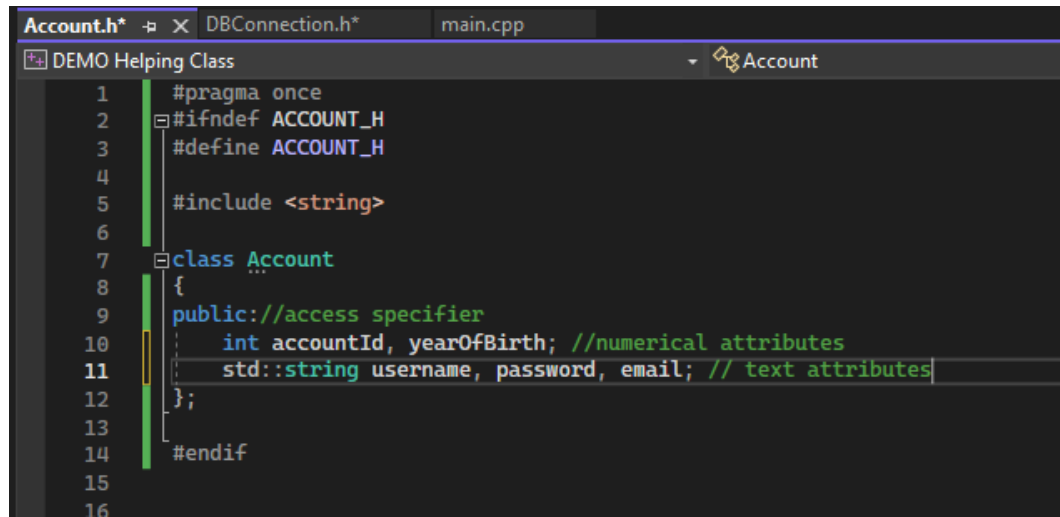
The .h file extension indicates that the file is **a header file** which in the context of C++, contains the declaration of the class members (what it should have). This is where you **declare** your class attributes and methods.

2.1.2.1. Attributes

Attributes are variable declared and associated to an object of a class. It should be the properties or characteristics that the class should have. For example, previously we have created an empty class "Account". Now we need to determine what property or characteristic or attributes an account should have according to your system requirements.

In this case, for demonstration purposes we assume that the class should have accountId, username, password, email, and year of birth.

Next, we need to determine the appropriate data type to be assigned for this attribute. Then we can declare it in the class header file as shown in the following diagram:

Based on the diagram: *(number in bracket refers to line)*

- The #pragma once (1) #ifndef CLASSNAME_H (2), #define CLASSNAME_H (3) and #endif (14) are the necessary code which tells the compiler to only compile the class once. This is necessary since there might be multiple files which include your custom class later on which may cause ambiguity error.
- public: is the access specifier for the class members defined after it. In this case all attributes declared in line 10 and 11 are assigned with access level public.
  There are a variety of access specifiers that have their own purpose to achieve encapsulation in OOP. For the sake of simplicity this guide will only be used by the public.
- Line 10,11 is the declaration of the attributes of Account class which is very similar to declaration of variable.

2.1.2.2. Methods
Methods are basically a function declared and associated to an object of a class. Method defines behavior of the class. It defines what the object instantiated from the class can do or what an object of the class should do when a certain method is called.

For example, we can have a class "Account" that has a method to calculate age using the year of birth attribute of the class. While the behavior **(method)** or the way the age is calculated will use same logic across all objects of the same class, the output may vary since each account **(object)** representing different user can have different date of birth **(attributes)**.

In C++ header file, we only declare the methods by determining its return type, name and parameters if any as shown in the following diagram.

```
Account.h  ⊕ ✕ DBConnection.h*        main.cpp
DEMO Helping Class                                  ▾   (Global Scope)                                       ▾
      1     #pragma once
      2     #ifndef ACCOUNT_H
      3     #define ACCOUNT_H
      4
      5     #include <string>
      6
      7     class Account
      8     {
      9     public://access specifier
     10         int accountId, yearOfBirth; //numerical attributes
     11         std::string username, password, email; // text attributes
     12         Account(); // constructor method
     13         Account(int accountId, int yearOfBirth, std::string username, std::string password, std::string email);
     14         bool login();
     15         void insert();
     16         void update();
     17         void remove();
     18         int getAge();
     19         ~Account(); //destructor method
     20     };
     21
     22     #endif
     23
```
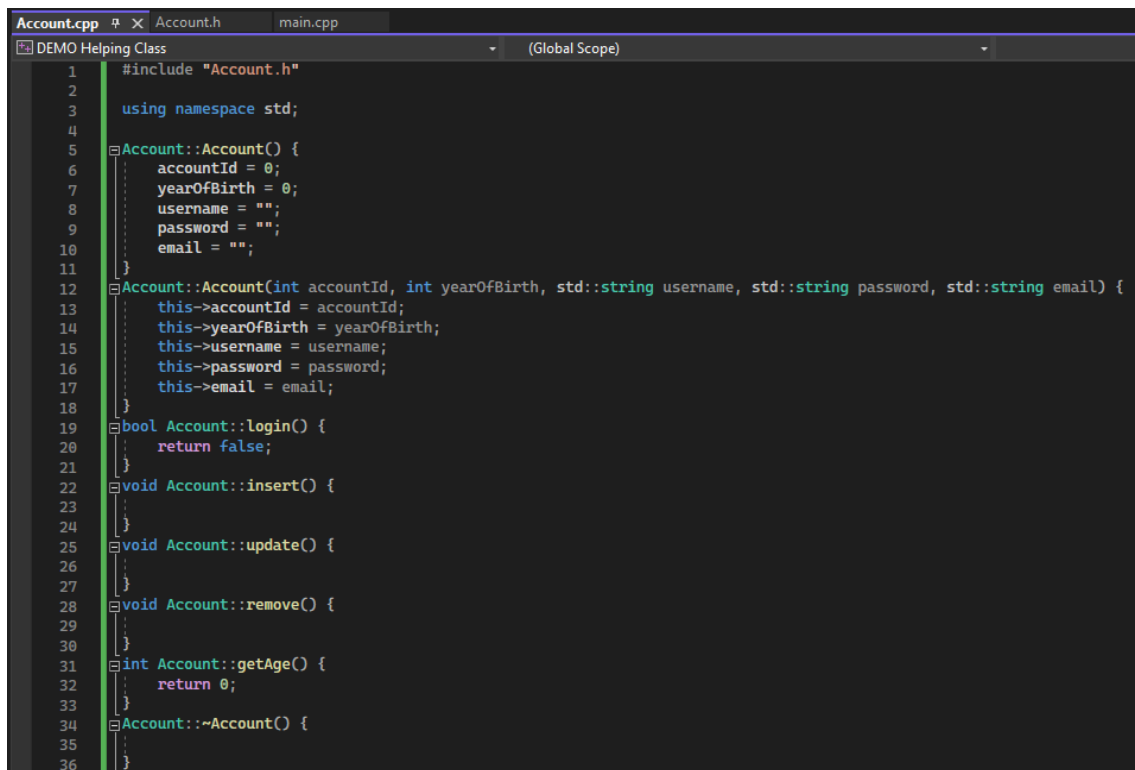
- Line 12 and 13 is the declaration of the constructor method. The characteristic of constructor method is that it does not have return type, and the name of the method must be exactly same with the class name. The constructor method will be automatically called when we instantiate an object of the class later.
  There must be at least 1 constructor method in a class.
- As you can see in line 12 and 13, there are two methods with same name but different parameters which is the implementation of method overloading. Generally, method name has to be unique unless as shown in the diagram, it has different parameters which the compiler can use to identify which of the two methods your code trying to call/invoke.
- The method with tilda "~" in line 19 is the destructor method. It will be called when your object goes out of scope. For example, if you instantiate objects of accounts for many users in the search menu. Then, when the user navigates to other menus, these objects will be "out of scope" and no longer used so the destructor will be automatically called. Other than that, destructor will also be called when you explicitly delete your object using the delete keyword.

2.1.3. Class.cpp

The previous section explained about the class header file which contains the **declaration** of the class method and attributes. Next, in order to complete the class, we will need to **define** the **implementation** of its method. For example, what it should do when getAge() is called etc.

Now in your class cpp file, the content should be similar to the following diagram:

```
Account.cpp  ⊣ ×  Account.h        main.cpp
DEMO Helping Class                           ▾    (Global Scope)                                    ▾
      1      #include "Account.h"
      2
      3      using namespace std;
      4
      5    ⊟Account::Account() {
      6          accountId = 0;
      7          yearOfBirth = 0;
      8          username = "";
      9          password = "";
     10          email = "";
     11     }
     12    ⊟Account::Account(int accountId, int yearOfBirth, std::string username, std::string password, std::string email) {
     13          this->accountId = accountId;
     14          this->yearOfBirth = yearOfBirth;
     15          this->username = username;
     16          this->password = password;
     17          this->email = email;
     18     }
     19    ⊟bool Account::login() {
     20          return false;
     21     }
     22    ⊟void Account::insert() {
     23
     24     }
     25    ⊟void Account::update() {
     26
     27     }
     28    ⊟void Account::remove() {
     29
     30     }
     31    ⊟int Account::getAge() {
     32          return 0;
     33     }
     34    ⊟Account::~Account() {
     35
     36     }
```

- #inclide "classname.h" in line 1 is a must so that the compiler can detect the class that you are **defining**.
- Classname::methodname are used to refer to your methods when defining their implementation.
- As you can see, the purpose of constructor method is to initialize necessary attributes which is important to prevent error when you try to manipulate something that doesn't exist (uninitialized)
- In the Account() constructor method we refer to the attributes only by using their names but in the other constructor with parameter in line 12, we refer to each of the class attributes explicitly using this keyword.
  This is important since in the context of the constructor with parameter, the names of the variables in the parameters are identical to the attribute name. That is why it is necessary to use the this-> attribute name to refer to your class attribute since referring by name will refer to the parameters instead in this context. Basically, "this" keyword refers to the current instance/object of the class.

Now, we will try to define a metho with some logic which is the getAge() method that should calculate the account owner current age.

```
Account.cpp  ₽ ✕  Account.h          main.cpp
⊞ DEMO Helping Class                                    ▼      (Global Scope)
     1    ⊟#include "Account.h"
     2     │#include <ctime>
     3      using namespace std;
     4
     5    ⊞Account::Account()│ { ... }
    12    ⊞Account::Account(int accountId, int yearOfBirth, std::string username, std::string p
    19    ⊞bool Account::login()│ { ... }
    22    ⊞void Account::insert()│ { ... }
    25    ⊞void Account::update()│ { ... }
    28    ⊞void Account::remove()│ { ... }
    31    ⊟int Account::getAge() {
    32          time_t current = time(0); // get time in epoch seconds (since 1900)
    33          tm now; // create a struct/object of tm to hold data
    34          localtime_s(&now, &current); //populate the now object with data from current
    35
    36          return (now.tm_year + 1900) // +1900 to the epoch year to get current year
    37             - yearOfBirth; // minus user year of birth to get their age
    38     │}
    39    ⊟Account::~Account() {
    40
    41     │}
    42
```
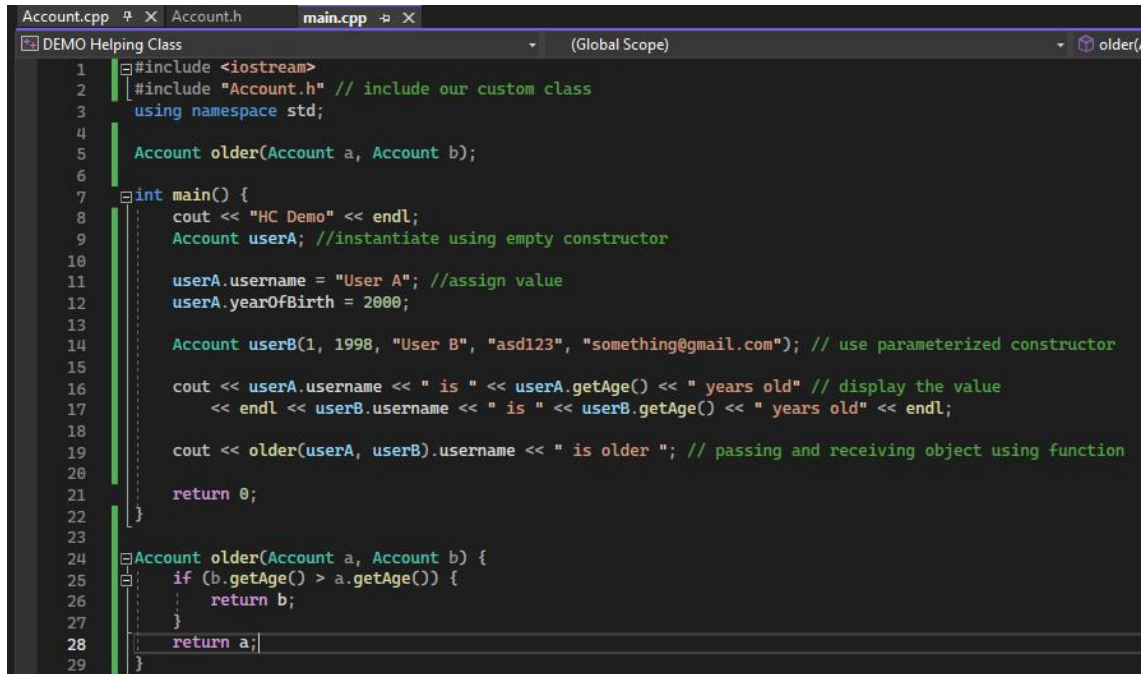
- Firstly, include the ctime library which is needed to get current time.
- Now in getAge() method, firstly we get the current epoch time user time(0)
- Then we convert it into manipulable struct/object tm using localtime_s
- Since the time(0) returns time since 1900, we need to add 1900 to the now.tm_year to get the current year.
- Finally, subtract the current year with the yearOfBirth attribute to get the age and return it.

  Technically, you can directly display the age in this method using cout, but it is **not recommended** to do so since you should delegate the responsibility properly among classes and methods.

  For example, getAge method should calculate user age and it should only do that not more or less. This way, we can **reuse** this method to display user's age or compare who is older since the method returns the data, the caller is free to do whatever is necessary without being bound to only display the data.

2.1.4. Using the class

Now that we have declared the class in the header file and define the methods in the cpp file. We can now use our class in our application. C++ applications must have main() function which is the entry point of the program. In this demonstration our main function is defined in the main.cpp file. The following shows the usage of class in main.cpp.

```cpp
#include <iostream>
#include "Account.h" // include our custom class
using namespace std;

Account older(Account a, Account b);

int main() {
    cout << "HC Demo" << endl;
    Account userA; //instantiate using empty constructor

    userA.username = "User A"; //assign value
    userA.yearOfBirth = 2000;

    Account userB(1, 1998, "User B", "asd123", "something@gmail.com"); // use parameterized constructor

    cout << userA.username << " is " << userA.getAge() << " years old" // display the value
        << endl << userB.username << " is " << userB.getAge() << " years old" << endl;

    cout << older(userA, userB).username << " is older "; // passing and receiving object using function

    return 0;
}

Account older(Account a, Account b) {
    if (b.getAge() > a.getAge()) {
        return b;
    }
    return a;
}
```

- Firstly, we need to include the class we have created by using its name.h. notice that unlike normal library such as iostream, for custom class we use "" symbol instead of <>.
- In line 9 we **declare/instantiate** an object of Account class and name the object as userA. Since we do not specify any parameter, it will automatically instantiate the object using the first empty constructor we have declared and defined.
- As explained earlier, class is a blueprint. In the previous section we declared and defined the behavior of the class which all the objects will have. Since we declared that account class have username, yearOfBirth, getAge() etc. so the instantiated object will surely have these attributes and method which can be **accessed using . (dot)**.
  Although, usage of pointer will not be emphasized in this guide, do take not that if you are using pointer of object instead you need to access it using → for example userA→username.
- You can **assign value** to public attributes directly as shown in line 10, 11.
- In line 13 we instantiate another object, but we passed multiple values as parameters which will make the program call the second **parameterized constructor** instead.
- Line 16 and 17 demonstrate how you can **access the attribute and method** to display it using cout.
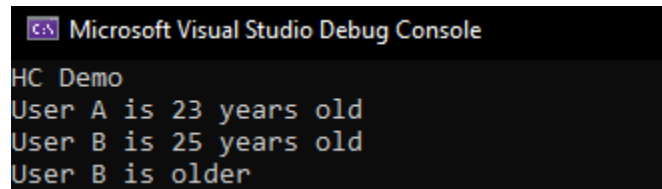
- Line 19 shows how you can use objects with function as parameter and return type. Passing object into function as parameter is as simple as passing a normal variable provided that the function is declared to receive object of that class.
- In the older() function we directly uses the return value of the getAge() method to compare which object has greater age and returns it.
- Since older() function is declared to return object of Account class in line 5. As shown in line 19, we can directly treat the function call as an object where we use dot (.) to directly access the attribute of the returned object.
Do note that, this way of handling returned object from function results to one time use of the returned object. If you wish to use it multiple times, then you can declare another object to store the returned object. For example:
`Account olderAcc = older(userA, userB);`

Executing the program now will produce the following output:
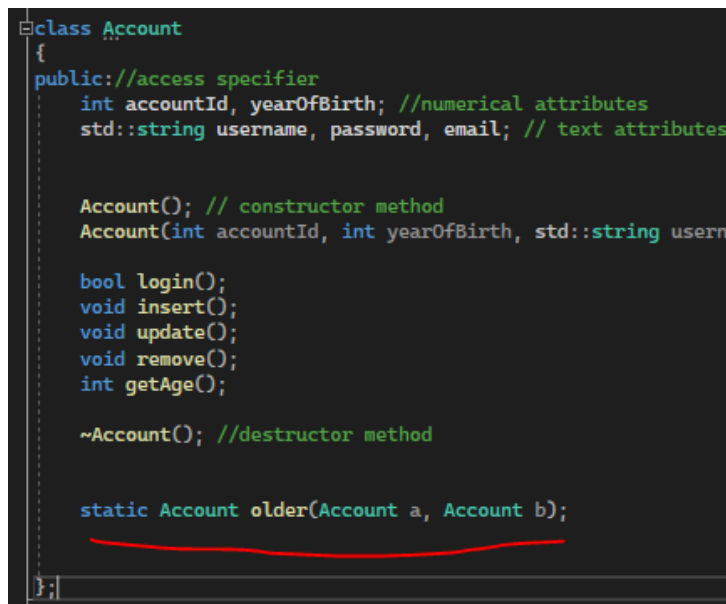
```
Microsoft Visual Studio Debug Console
HC Demo
User A is 23 years old
User B is 25 years old
User B is older
```

2.1.5. Static method

Notice that the function "older()" returns object of class Account and receive two object of class Account in its parameter. This function will not be able to operate without the class "account" since it is completely reliant on it. Thus, we can move this function into its associated class as a static method.

- In Account.h declare the method

```cpp
class Account
{
public://access specifier
    int accountId, yearOfBirth; //numerical attributes
    std::string username, password, email; // text attributes


    Account(); // constructor method
    Account(int accountId, int yearOfBirth, std::string userna

    bool login();
    void insert();
    void update();
    void remove();
    int getAge();

    ~Account(); //destructor method


    static Account older(Account a, Account b);


};
```

- Move the function definition into Account.cpp as method definition. Do not forget to add Account:: to let the compiler know that you are defining/implementing a method.

```cpp
int Account::getAge() {
    time_t current = time(0); // get time in epoch seconds (since 1900)
    tm now; // create a struct/object of tm to hold data
    localtime_s(&now, &current); //populate the now object with data from current

    return (now.tm_year + 1900) // +1900 to the epoch year to get current year
        - yearOfBirth; // minus user year of birth to get their age
}
```

- Try using it by calling the class name Account::methodname().

```cpp
cout << Account::older(userA, userB).username << " is older "; // passing and receiving object using function

Account olderAcc = Account::older(userA, userB);
```

- The primary difference between normal method and static is that normal method is bound to object. It cannot be called without instantiating an object.
  On the other hand, you can call static method directly from class name.
- Few of the basic characteristic of a static method:
  o It does not use any of the instance attributes. (cannot uses this keyword)
  o It processes something outside of the scope of single instance of object. (e.g.: older() method compare 2 object)
  o It is related to the class it belongs to similar to how this older() method is reliant on the Account class.

2.2. Menu

This section basically explains how to modularize your menu to make it reusable and standardized.

The menu is basically a list of options users can choose from which is the very basic form of user interaction with your system. The general idea of the menu here is to prompt users with a list of possible actions and take their decision as input. Most common implementation of menu in CLI (Command Line Interface) is by using user key input to identify which option they are choosing. For example:
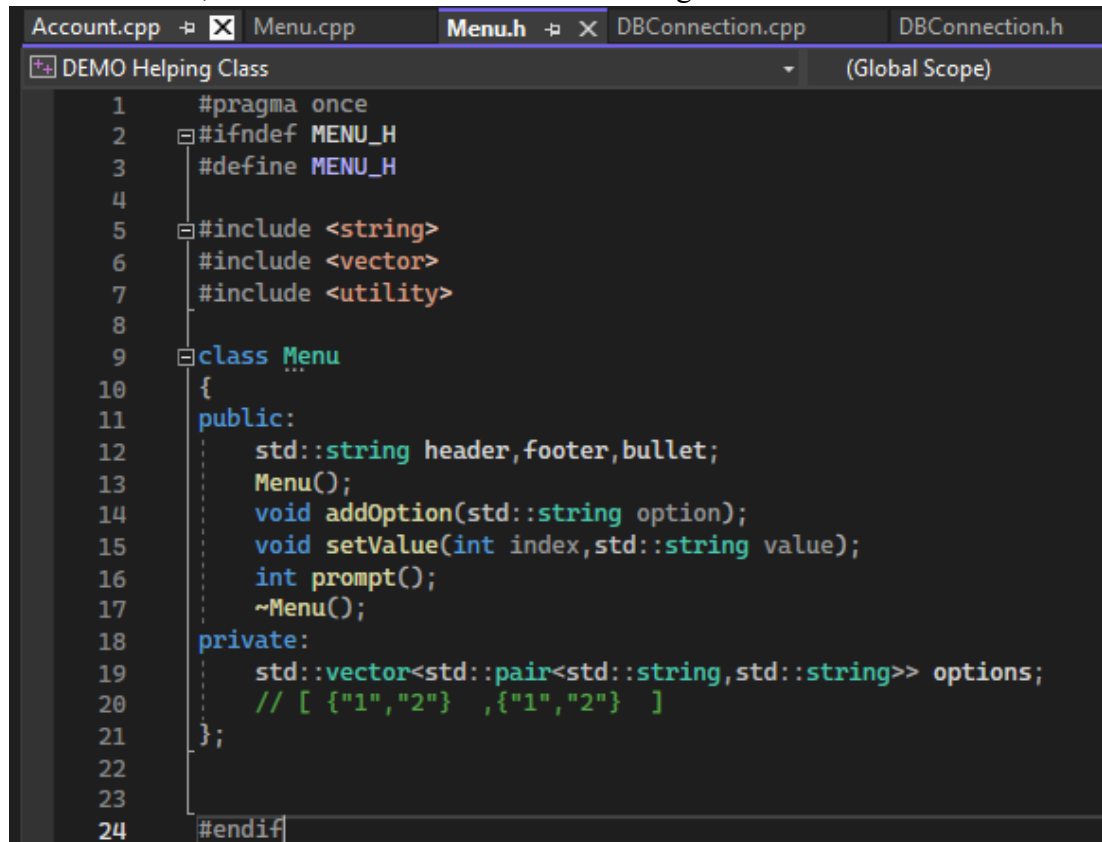
```
Welcome
1. Profile
2. Find User
3. Find Shop
Insert the number to choose:
```

```cpp
int option;
cout << endl << "Welcome" << endl << "1. Profile"
    << endl << "2. Find User" << endl << "3. Find Shop"
    << endl << "Insert the number to choose: ";
cin >> option;
```

The diagram shows a simple implementation of menu which shows option to user and takes their input. This implementation is indeed correct and works but throughout your system, you will have multiple menus and variety of options which all share the same logic "prompt and wait for user input option". Since we have multiple processes that share the same logic, then we can modularize this to make a reusable component. Furthermore, using basic menu, your application might have bad navigability, users cannot go back from your menu which can be annoying for them. Besides that, when you prompt users to input data consecutively, they can't go back to edit the previous data and some bad implementation also does not allow users to cancel or go back unless they finish inserting.

In this, section we will create a simple menu class which can solves the stated problem to increase navigability of our application.
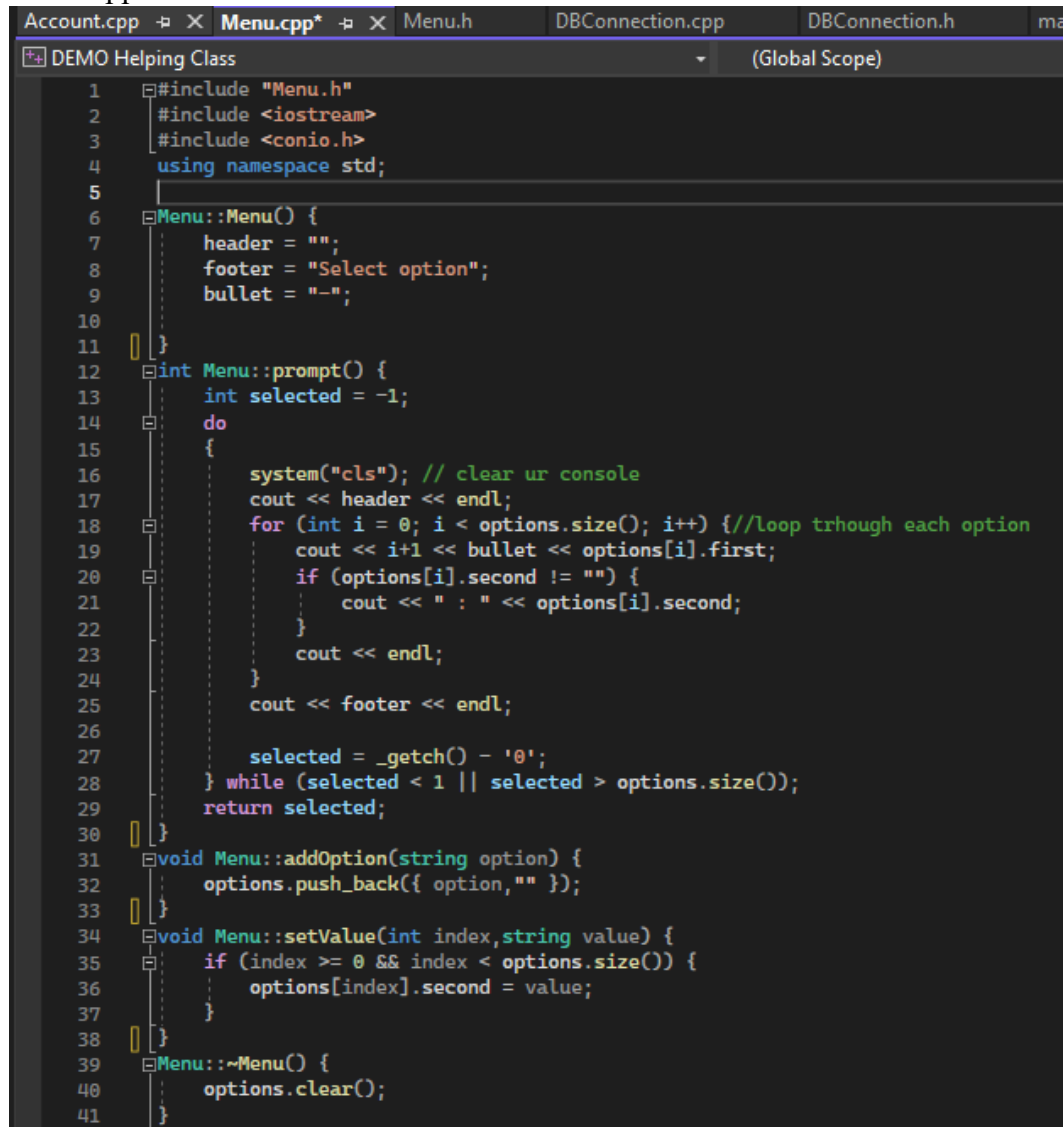
a) Create a new class named "Menu".
b) The header file, Menu.h should contain the following declarations.

```cpp
#pragma once
#ifndef MENU_H
#define MENU_H

#include <string>
#include <vector>
#include <utility>

class Menu
{
public:
    std::string header,footer,bullet;
    Menu();
    void addOption(std::string option);
    void setValue(int index,std::string value);
    int prompt();
    ~Menu();
private:
    std::vector<std::pair<std::string,std::string>> options;
    // [ {"1","2"}  ,{"1","2"}  ]
};

#endif
```

- The options attribute is a vector of pair.
- Pair is a data structure in C++ which store pair of data of defined type. In this case we declared that the pair is pair<string,string> which means that it will hold 2 string data as a pair accessible using first and second keyword. In the context of this menu class, the first string in the pair is the option text, and the second string in the pair is the value associated with the option.
- We put the pair<string,string> into vector<?> to declare that this attribute options will store collections of data which is pair of string.
- Vector is a dynamic array which works almost the same with a typical array but can grow and shrink in size.
- The purpose of options attribute is to store the option in the menu and its associated data if any.

c) Menu.cpp

⊞ DEMO Helping Class   ▼   (Global Scope)
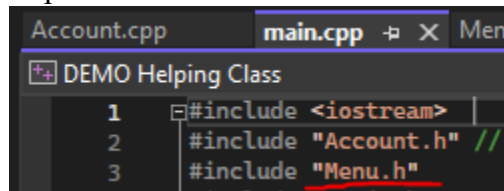
```cpp
1   #include "Menu.h"
2   #include <iostream>
3   #include <conio.h>
4   using namespace std;
5
6   Menu::Menu() {
7       header = "";
8       footer = "Select option";
9       bullet = "-";
10
11  }
12  int Menu::prompt() {
13      int selected = -1;
14      do
15      {
16          system("cls"); // clear ur console
17          cout << header << endl;
18          for (int i = 0; i < options.size(); i++) {//loop trhough each option
19              cout << i+1 << bullet << options[i].first;
20              if (options[i].second != "") {
21                  cout << " : " << options[i].second;
22              }
23              cout << endl;
24          }
25          cout << footer << endl;
26
27          selected = _getch() - '0';
28      } while (selected < 1 || selected > options.size());
29      return selected;
30  }
31  void Menu::addOption(string option) {
32      options.push_back({ option,"" });
33  }
34  void Menu::setValue(int index,string value) {
35      if (index >= 0 && index < options.size()) {
36          options[index].second = value;
37      }
38  }
39  Menu::~Menu() {
40      options.clear();
41  }
```
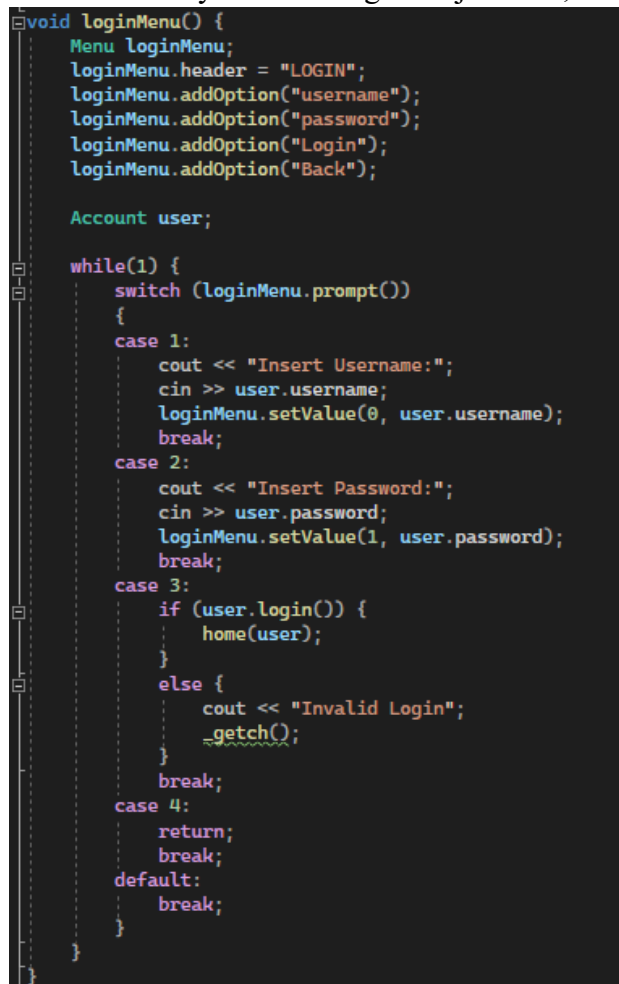
- Initialize the default header, footer and bullet in constructor.
- Prompt method will display the menu starting with header, option until footer and wait for user input using _getch() which values are converted into integer by – '0'.
- The loop inside the prompt will keep displaying the menu until user pressed a number which is included in the option. This will prevent from your menu abruptly closed when user miss pressed something.
- During the loop it will first display the first part of the option pair which is the option text, while the second part, value is only displayed if it is not empty.
- addOption() method is used to pass the option text to be added into the vector of string pairs. Curly braces {} is used to form a new pair variable which then appended to the vector via push_back
- setValue() method is to allow assigning value to the second part of the string pair in the option vector at specific index.

d) Import the custom header inside the file you want to use it in.

```
Account.cpp          main.cpp  ⊟ ✕  Men
⊞ DEMO Helping Class
1      ⊟#include <iostream>
2       #include "Account.h" //
3       #include "Menu.h"
```

e) Use the class by instantiating an object of it, assign values and call the prompt method.

```cpp
void loginMenu() {
    Menu loginMenu;
    loginMenu.header = "LOGIN";
    loginMenu.addOption("username");
    loginMenu.addOption("password");
    loginMenu.addOption("Login");
    loginMenu.addOption("Back");

    Account user;

    while(1) {
        switch (loginMenu.prompt())
        {
        case 1:
            cout << "Insert Username:";
            cin >> user.username;
            loginMenu.setValue(0, user.username);
            break;
        case 2:
            cout << "Insert Password:";
            cin >> user.password;
            loginMenu.setValue(1, user.password);
            break;
        case 3:
            if (user.login()) {
                home(user);
            }
            else {
                cout << "Invalid Login";
                _getch();
            }
            break;
        case 4:
            return;
            break;
        default:
            break;
        }
    }
}
```

- while(1) is an infinite loop since 1 is always true. The loop will only exit when user press 3 which makes the program goes into case 3 to return 0 and exit the main() function.
- Menuobject.prompt() will return the integer of valid option. By putting the method call inside switch it will automatically evaluate the returned value from the prompt which is the option user selected in integer.
- Menuobject.setValue() used to display back the value that user has inserted to imitate form interface.

f) Sample output

Even though username is inserted, user can still edit their input for username if they select 1. They also can cancel the process by selecting 4 without having to key in username and password. This provides much better navigability for your user.

2.3. Database Operation

To make database operation in your C++ application we will have to utilize the third-party library which we have imported in the previous section. This section will explain about the class "DBConnection" which handles the database operation.

a) Declare the followings inside DBConnection.h

```cpp
#pragma once
#ifndef DBConnection_H
#define DBConnection_H
#include <mysql/jdbc.h>
#include <string>
class DBConnection
{

public:
    DBConnection();
    ~DBConnection();
    void prepareStatement(std::string query);
    sql::PreparedStatement* stmt;
    sql::ResultSet* res;
    void QueryStatement(); // have no result
    void QueryResult(); // it has result


private:
    sql::Connection* connection;

};

#endif // !DBConnection_H
```

- The stmt attribute will store our prepared statement.
- The res attribute will store result of the query if it has any.
- The connection will hold the connection object to our database.
- All of these attributes are a pointer and have to be closed in destructor method.

b) Implements the class in DBConnection.cpp

```cpp
DBConnection::DBConnection() {

    try {
        mysql::MySQL_Driver* driver = mysql::get_mysql_driver_instance();

        connection = driver->connect("tcp://127.0.0.1:3306", "root", "");
        //database address, username, password

        connection->setSchema("demo_hc"); //database name
        stmt = nullptr;
        res = nullptr;
    }
    catch (sql::SQLException& e) {
        if (e.getErrorCode() == 0) {
            system("cls");
            cout << "Unable to connect to database";
            _getch();
            exit(0);
            return;
        }
        cout << "# ERR: SQLException in " << __FILE__;
        cout << "(" << __FUNCTION__ << ") on line " << __LINE__ << endl;
        cout << "# ERR: " << e.what();
        cout << " (MySQL error code: " << e.getErrorCode();
        cout << ", SQLState: " << e.getSQLState() << " )" << endl;
        _getch();
    }
}
```

- Ensure that the parameters in connect() is correct. 127.0.0.1 is your localhost address and :3306 is the port number MySQL running on. You can double check the port on which MySQL run in your XAMPP. Username root with empty password is the default configuration of MySQL if you changed something in your own MySQL server then uses those username and password in the parameters.

c) Implement the rest of the methods.

```cpp
DBConnection::~DBConnection() {
    try {
        if (connection) {
            connection->close();
        }
        if (stmt) {
            stmt->close();
        }
        if (res) {
            res->close();
        }
    }
    catch (sql::SQLException& e) {
        cout << "# ERR: SQLException in " << __FILE__;
        cout << "(" << __FUNCTION__ << ") on line " << __LINE__ << endl;
        cout << "# ERR: " << e.what();
        cout << " (MySQL error code: " << e.getErrorCode();
        cout << ", SQLState: " << e.getSQLState() << " )" << endl;
        _getch();
    }
}

void DBConnection::prepareStatement(string query) {
    stmt = connection->prepareStatement(query);
}

void DBConnection::QueryStatement() {
    try {
        stmt->executeUpdate();
    }
    catch (sql::SQLException& e) { ... }
}

void DBConnection::QueryResult() {
    try {
        res = stmt->executeQuery();
    }
    catch (sql::SQLException& e) { ... }
}
```

- Try and catch block will display error message when sql error occurred. The minimized catch block { … } all hase the same code as the one in catch block of ~DBConnection() method.

2.4. Create (Insert)

The process of inserting data from user input into database will be explained. Assuming that you have already created the Account class. We have already declared the insert() method in its header file previously. Now we need to add its implementation in the Account.cpp as follows.

```cpp
void Account::insert() {

    DBConnection db;//instantiate
    db.prepareStatement("Insert into account (username,password,email,yearOfBirth) VALUES (?,?,?,?)");
    db.stmt->setString(1, username);
    db.stmt->setString(2, password);
    db.stmt->setString(3, email);
    db.stmt->setInt(4, yearOfBirth);
    db.QueryStatement();
    db.~DBConnection();
}
```

- To perform database operation, we have to instantiate an object of the DBConnection class which will automatically calls its constructor which establish connection to our system database.
- Pass your prepared statement into the prepareStatement() method.
- Prepared statement is safe way to perform a query. It has the exact same syntax with normal SQL query except that the values are replaced with placeholders which is the question marks.
- Make sure that your placeholders in prepared statement (?) matches with the value being set at that particular index.
- The index of prepared statements here starts from 1, not 0.
- To set value into the prepared statement, call the set? Method depending on the data type. For numerical setInt, if with decimal setDouble and if text use setString.
- Ensure that the index is accurate for which ? you want the value to be set to. In the example, the first ? is for username so username data from the attribute should be set into the first position. You can change the order of the code but the first parameter of the setString must be the accurate index.
- When you set the value into prepared statement placeholder, it will automatically escapes your value so you don't need to think about whether to put ' " or not in your query. Just leave it to the prepared statement. Further explanation on prepared statements will be appended in the additional note.

Now create a new menu for registration using the menu class as shown in the following example.

```cpp
void registerAccount() {
    Account newacc;

    Menu rgMenu;
    rgMenu.header = "Registration";
    rgMenu.addOption("Username");
    rgMenu.addOption("Password");
    rgMenu.addOption("Email");
    rgMenu.addOption("Year of Birth");
    rgMenu.addOption("Register");
    rgMenu.addOption("Back");

    while (1) {

        switch (rgMenu.prompt()) {
        case 1:
            cout << "Insert Username:";
            cin >> newacc.username;
            rgMenu.setValue(0, newacc.username);
            break;
        case 2:
            cout << "Insert password:";
            cin >> newacc.password;
            rgMenu.setValue(1, newacc.password);
            break;
        case 3:
            cout << "Insert email:";
            cin >> newacc.email;
            rgMenu.setValue(2, newacc.email);
            break;
        case 4:
            cout << "Insert yearOfBirth:";
            cin >> newacc.yearOfBirth;
            rgMenu.setValue(3, to_string(newacc.yearOfBirth));
            break;
        case 5:
            newacc.insert();
            return;
        case 6:
            return;
        default:
            break;
        }

    }

}
```

- Similarly, instantiate an object of Menu and add the options.
- Instantiate an object of account to store the new account data.
- Use the while(1) to create infinite loop so that it will only go back when user choose to. Calls the prompt method and put it inside switch case.
- Inside each case, instruct user to input the field associated to the option.
- Store the input value into appropriate attribute of the object.
- For case 5 which is the register option, call the insert method which will save the data into database.

- Link your menus via a main menu in main()

```cpp
Menu mainmenu;
mainmenu.header = "Welcome to Demo";
mainmenu.addOption("Register");
mainmenu.addOption("Login");
mainmenu.addOption("Exit");

while (1) {
    switch (mainmenu.prompt())
    {
    case 1:
        registerAccount();
        break;
    case 2:
        loginMenu();
        break;
    case 3:
        return 0;
    default:
        break;
    }
}
```

- You should get the similar output. (this example already inserted username and email)

```
D:\Academic\degree\Sem5(Year3Sem2)2023-2024-1\
Registration
1-Username : asd123
2-Password
3-Email : asd@gmail.com
4-Year of Birth
5-Register
6-Back
Select option
```

- Try to fill in all information and select register option.
- Verify that the data you inserted now exist in the database using MySQL cmd or PHPMyAdmin

2.5. Simple Read (Login)

This section will explains about how to process login in your application. The interface code has already been exposed previously in section Menu (e). the implementation of the login() method in Account.cpp is as follows:

```cpp
bool Account::login() {

    DBConnection db;
    db.prepareStatement("SELECT * FROM account WHERE username=? AND password=?");
    db.stmt->setString(1, username);
    db.stmt->setString(2, password);
    db.QueryResult();
    if (db.res->rowsCount() == 1) {
        while (db.res->next()) {
            accountId = db.res->getInt("accountId");
            username = db.res->getString("username");
            password = db.res->getString("password");
            email = db.res->getString("email");
            yearOfBirth = db.res->getInt("yearOfBirth");
        }
        db.~DBConnection();
        return true;
    }
    else {

        db.~DBConnection();
        return false;
    }
}
```

- Instantiate object of database connection to establish connection.
- Prepare statement and pass values into it appropriately.
- Call the QuerResult() which will execute the query and stores the result into result set inside res attribute.
- Check the row count from res attribute. Since this is for login, there must be exactly 1 match for the username with matching password. Else it will return false.
- If there is a result, use the while(db.res->next()) to access each of the data in the result set.
- Read the data from result set using getInt or getString or getDouble depends on your data type using the column name as parameter and store it into its corresponding attribute.
- Define the home menu function in main.cpp.
- Pass the logged in user object into all subsequent menu functions so you can maintain the user data which will be necessary later. For example, you will need to have the user id to record any activities that the user do for example when they make a purchase etc.

```
void home(Account user) {
    Menu homeMenu;
    homeMenu.addOption("Profile");
    homeMenu.addOption("Shop");
    homeMenu.addOption("History");
    homeMenu.addOption("Logout");
    while (1) {
        homeMenu.header = "Welcome " + user.username;
        switch (homeMenu.prompt())
        {
        case 1:
            user = profile(user);
            break;
        case 2:
            break;
        case 3:
            break;
        case 4:
            return;
            break;
        default:
            break;
        }
    }
}
```

- Try your login function, see if it works with valid and invalid credentials. On successful login it should display the home menu with options stated in above diagram.

2.6. Update (Modify)

Previously we have successfully verified user credential for log in. In this section we will proceed to the U in the CRUD which is update where we going to make a profile change feature. Firstly, add the implementation of update method in Account.cpp. The querying process is similar to insertion process where we user prepared statement and set value into it accordingly.

```
void Account::update() {

    DBConnection db;
    db.prepareStatement("UPDATE account SET username=?, password=?, email=?,yearOfBirth=? WHERE accountId=?");
    db.stmt->setString(1, username);
    db.stmt->setString(2, password);
    db.stmt->setString(3, email);
    db.stmt->setInt(4, yearOfBirth);
    db.stmt->setInt(5, accountId);
    db.QueryStatement();
    db.~DBConnection();

}
```

Now create a new profile menu. (the picture omits some minor code, refer to the demo project files if necessary)

```cpp
Account profile(Account user) {

    Account temp = user; // copy the object

    Menu profileMenu;
    profileMenu.header = "Your profile";
    profileMenu.addOption("username");
    profileMenu.addOption("password");
    profileMenu.addOption("email");
    profileMenu.addOption("yearOfBirth");
    profileMenu.addOption("Reset");
    profileMenu.addOption("Save");
    profileMenu.addOption("Back");
    profileMenu.addOption("Delete Account");

    while (1) {
        profileMenu.setValue(0, temp.username);
        profileMenu.setValue(1, temp.password);
        profileMenu.setValue(2, temp.email);
        profileMenu.setValue(3, to_string(temp.yearOfBirth));
        profileMenu.footer = "You are " + to_string(temp.getAge()) + " Years old\nSelect Option";

        switch (profileMenu.prompt())
        {
        case 1:
            cout << "Insert Username:";
            cin >> temp.username;
            break;
        case 2:
            cout << "Insert password:";
            cin >> temp.password;
            break;
        case 3:
            cout << "Insert email:";
            cin >> temp.email;
            break;
        case 4:
            cout << "Insert year of birth:";
            cin >> temp.yearOfBirth;
            break;
        case 5:
            temp = user;
            break;
        case 6:
            user = temp;
            user.update();
            cout << "Updated";
            _getch();
        case 7:
            return user;
            break;
        case 8:
            cout << "Delete your account? (y/n)";
            char confirm;
            confirm = _getch();
            if (confirm == 'Y' || confirm == 'y') {
                user = temp;
                user.remove();
                main();
            }
```

- Configure the menu object similar to previous examples.
- Create infinite loop with switch case and prompt() method call in it.
- Instantiate a temporary Account object which will be used to display changes.
- Copy value from the user object in the parameter into the temp object by assigning it as shown in the example.
- For input cases save user input into the temporary Account object.
- For case 5 which is reset option, copy back the data from Account object "user" in the parameter to the temp object.
- For case 6 save, copy the values from the temp object into user object and call the update method.
- Notice that in case 6 there is no break, which mean the code will proceed down to case 7 after case 6 automatically to return the saved user object to the calling function which is the main menu.
- Try your profile menu features.

2.7. Delete (Remove)

Deleting or removing data from database will also be handled by Account class. The menu option for delete example has already been included in the profile menu previously. The following are the implementation of the remove() function in Account.cpp.

```cpp
void Account::remove() {
    DBConnection db;
    db.prepareStatement("DELETE FROM account WHERE accountId=?");
    db.stmt->setInt(1, accountId);
    db.QueryStatement();
    db.~DBConnection();
}
```

- Query execution for DELETE is similar to update and insert since it does not return any result.
- Again, instantiate database object, prepare query, set value, execute and close connection.
- The option in the profile menu should now works.

Now, you have basically developed full CRUD functionality for 1 table "Account". You can apply similar logic to any of your database table by creating its entity class. Processes for insert, update and delete are mostly similar. However, select process might be different and involve more complex queries which will be explained in the upcoming section.

2.8. Read (Search)

One of the most must-have features in an information system is searching. In this section, we will cover about how to create a dynamic prepared statement and use it in database query. In the previous section we only use DBConnection, Menu and Account classes. In this section we will have 2 more additional classes which is Product and Transaction. Please refer to the source codes of the demo project for complete code since from now on only important part will be explained.

2.9. Read (Report)

Another usage of reading data from database is to generate report.

3. Improvement

3.1. Input Validation

3.2. Error Handling

4. Additional Notes

4.1. Regular Expression

4.2. Arrow Menu

4.3. GitHub

DEMO Helping Class.exe - System Error

The code execution cannot proceed because mysqlcppconn-9-vs14.dll was not found. Reinstalling the program may fix this problem.

OK

---

lib64

File   Home   Share   View

Pin to Quick access   Copy   Paste   Cut   Copy path   Paste shortcut   Move to   Copy to   Delete   Rename   New folder   New item   Easy access   Properties   Open   Edit   History   Select all   Select none   Invert selection

Clipboard   Organize   New   Open   Select

« mysql-connector-... > lib64   Search lib64

Name	Date modified	Type	Size
plugin	24/06/2023 4:53 PM	File folder	
vs14	24/06/2023 4:53 PM	File folder	
libcrypto-3-x64.dll	31/05/2023 5:38 AM	Application exten...	5,026 KB
libssl-3-x64.dll	31/05/2023 5:38 AM	Application exten...	760 KB
mysqlcppconn8-2-vs14.dll	24/06/2023 4:41 PM	Application exten...	3,932 KB
mysqlcppconn8-2-vs14.pdb	24/06/2023 4:41 PM	Program Debug D...	24,556 KB
mysqlcppconn-9-vs14.dll	24/06/2023 3:48 PM	Application exten...	8,964 KB
mysqlcppconn-9-vs14.pdb	24/06/2023 3:48 PM	Program Debug D...	16,316 KB

DEMO Helping Class

File   Home   Share   View

Pin to Quick access   Copy   Paste   Cut   Copy path   Paste shortcut   Move to   Copy to   Delete   Rename   New folder   New item   Easy access   Properties   Open   Edit   History   Select all   Select none   Invert selection

Clipboard   Organize   New   Open   Select

« HC > DEMO Helping Cl...   Search DEMO Helping Class

Name	Date modified	Type	Size
.vs	16/10/2023 11:00 PM	File folder	
DEMO Helping Class	18/10/2023 1:09 PM	File folder	
x64	17/10/2023 12:47 PM	File folder	
DEMO Helping Class.sln	16/10/2023 11:00 PM	Visual Studio Solu...	2 KB

File   Home   Share   View

Pin to Quick access   Copy   Paste   Cut   Copy path   Paste shortcut   Move to   Copy to   Delete   Rename   New folder   New item   Easy access   Properties   Open   Edit   History   Select all   Select none   Invert selection

Clipboard   Organize   New   Open   Select

« x64 > Release   Search Release

Desktop
Downloads
Documents
Pictures
G:\
EX
GDRVIE

Name	Date modified	Type	Size
DEMO Helping Class.exe	18/10/2023 1:22 PM	Application	25 KB
DEMO Helping Class.pdb	18/10/2023 1:22 PM	Program Debug D...	1,196 KB
libcrypto-3-x64.dll	31/05/2023 5:38 AM	Application exten...	5,026 KB
libssl-3-x64.dll	31/05/2023 5:38 AM	Application exten...	760 KB
mysqlcppconn8-2-vs14.dll	24/06/2023 4:41 PM	Application exten...	3,932 KB
mysqlcppconn-9-vs14.dll	24/06/2023 3:48 PM	Application exten...	8,964 KB