

Table of Contents

1. Introduction.....	2
1. Project Setup.....	2
1.1. Visual Studio	2
1.2. MySQL Database	3
1.3. MySQL Connector	6
2. Development.....	15
2.1. OOP in C++.....	15
2.2. Menu.....	25
2.3. Database Operation	29
2.4. Create (Insert).....	32
2.5. Simple Read (Login)	35
2.6. Update (Modify).....	36
2.7. Delete (Remove)	38
2.8. Read (Search): Dynamic Query	39
2.9. Formatting String	40
2.10. Get Last Generated ID.....	41
2.11. Bulk Insert	42
2.12. Read (Report): Query with Join statements.....	44
2.13. Input Validation	47
3. Additional Notes	51
3.1. Regular Expression	53
3.2. Arrow Menu	53
3.3. mysqlcpp-9-vs14.dll not found Error.	56

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.

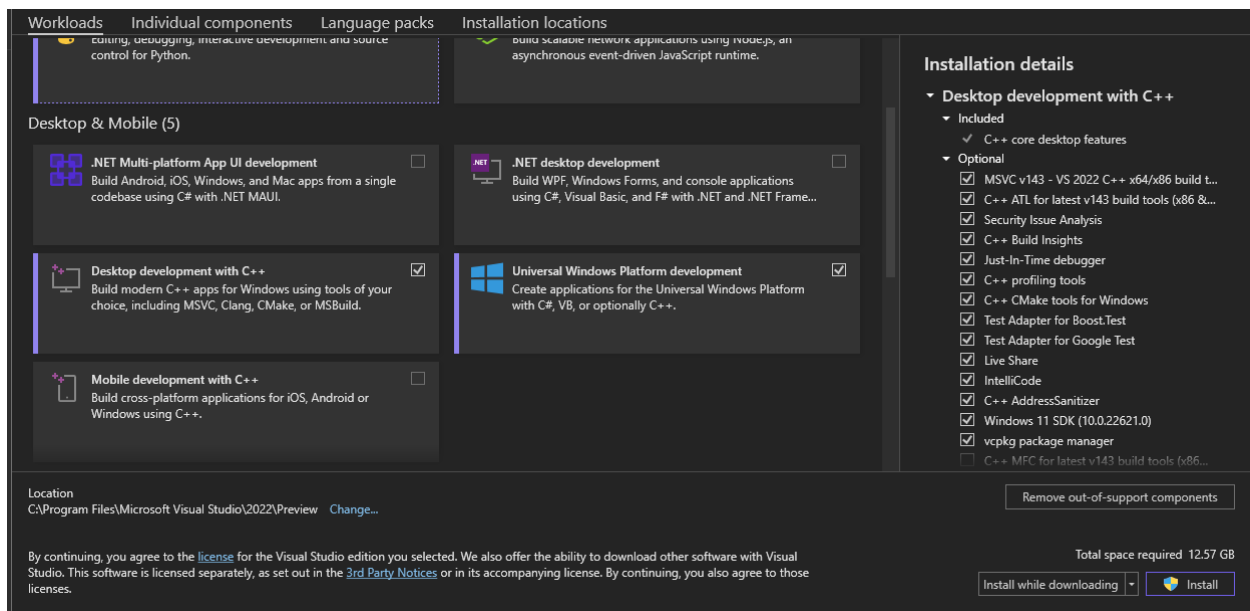
1. Project Setup

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.



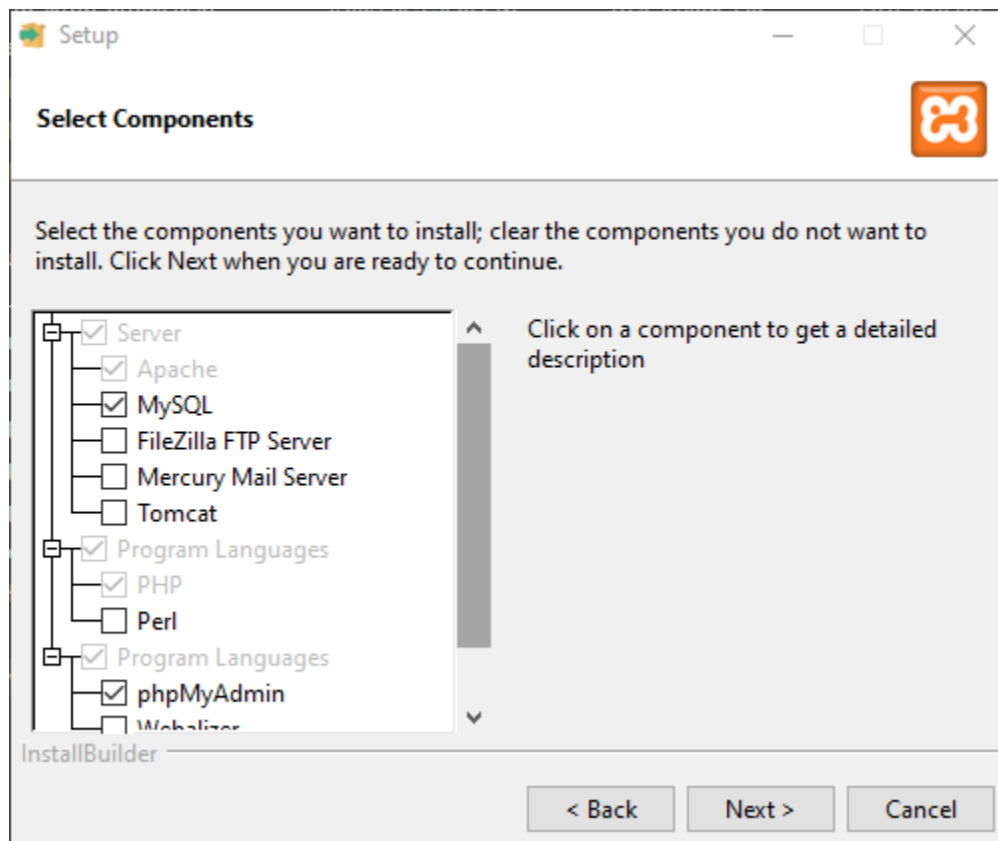
Other installation parameters, path, language etc. is up to your personal preferences. Proceed with the installation and wait until it finishes.

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.

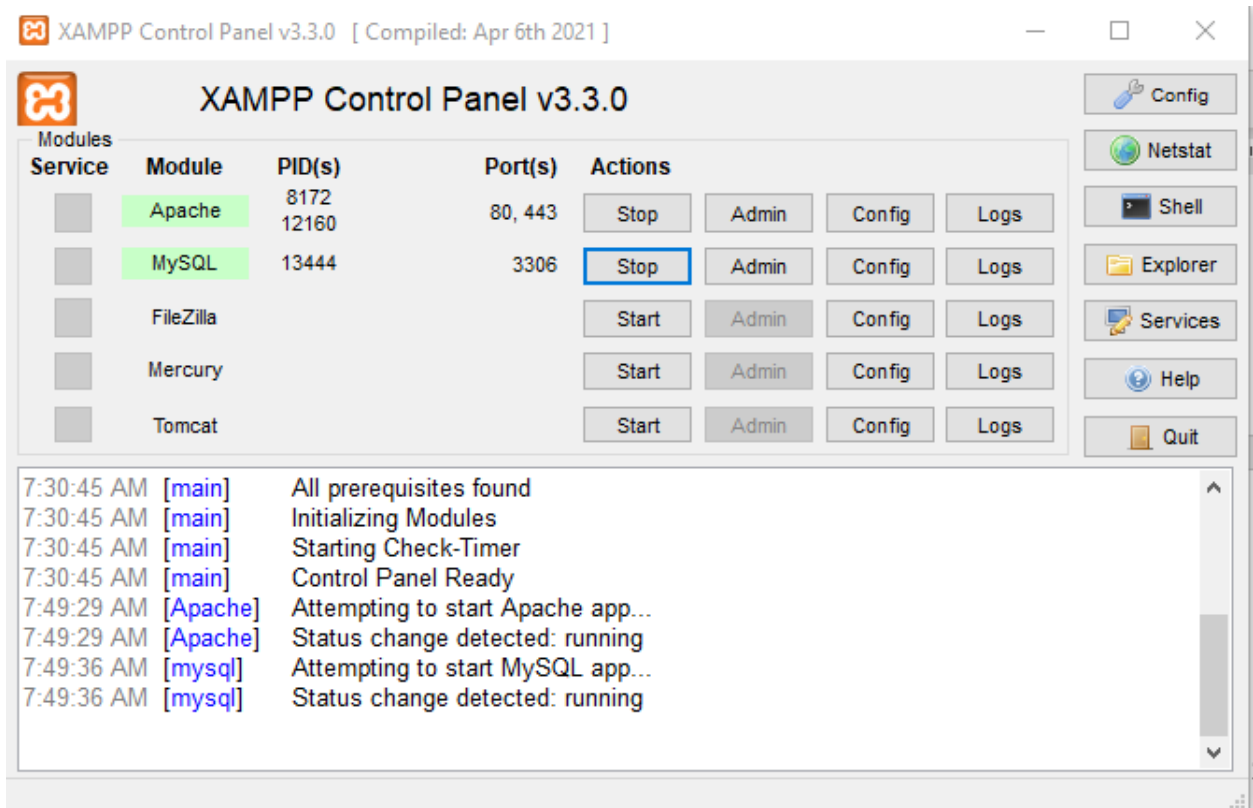
1.2.1. XAMPP

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.

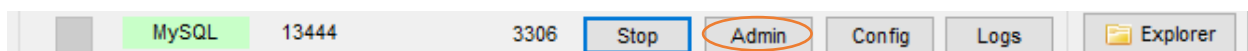


Again, there are multiple services provided by XAMPP. However, the most important service is MySQL which will be your database. Currently your database server will be running in your local server (localhost). You need to ensure that your MySQL is running whenever you want to run your C++ application 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.

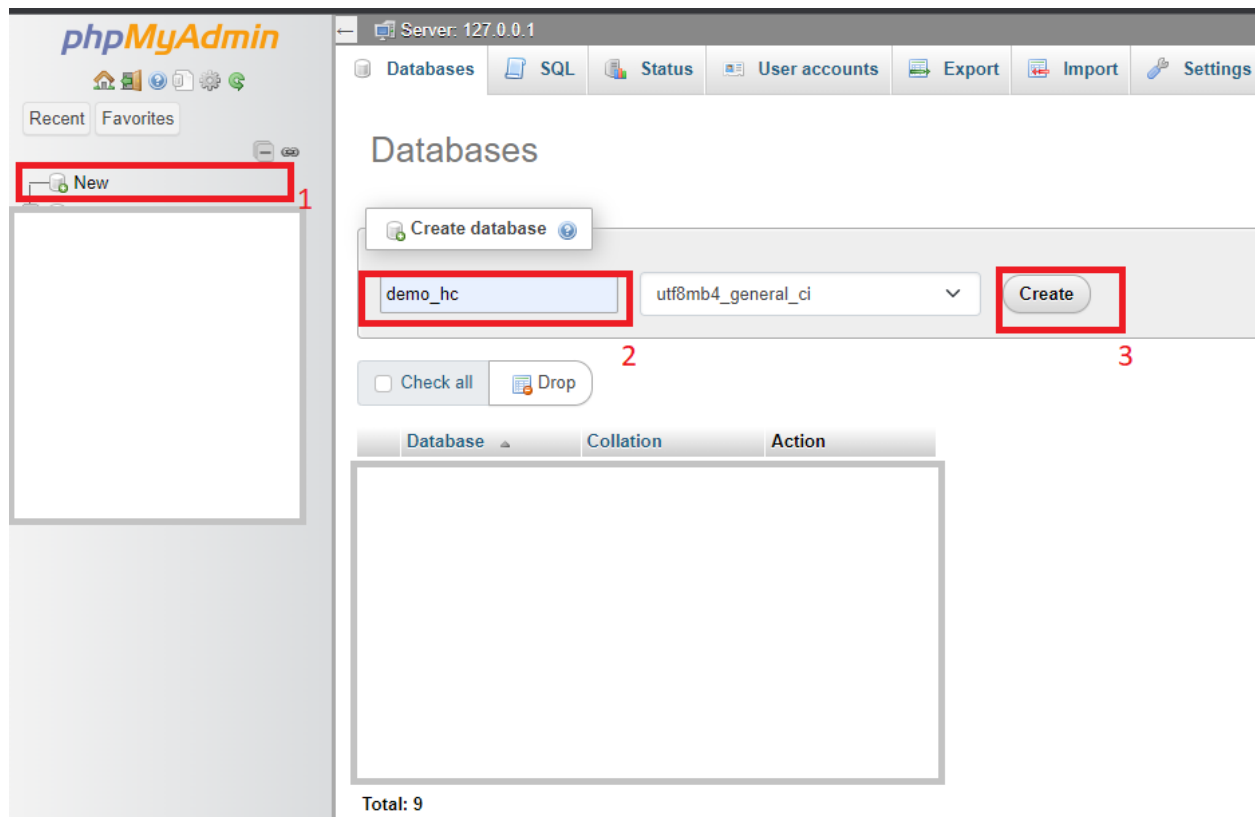
1.2.3. PHPMyAdmin

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.

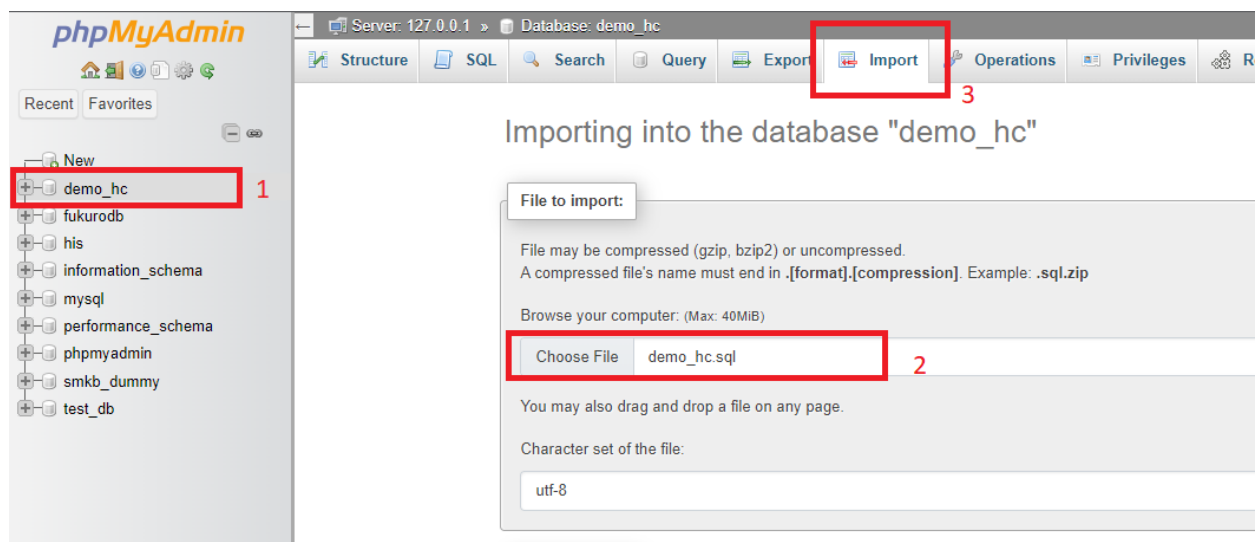
To run PHPMyAdmin you must ensure that Apache service is running in your XAMPP control panel since PHPMyAdmin is basically a php web application that runs on the Apache server locally. Then, click the admin button in the MySQL row in the control panel to start it.



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 .sql file. Finally scroll down and find the import button.



After you have imported the database, you can now proceed to follow through this guide.

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.

MySQL Community Downloads

Connector/C++

General Availability (GA) Releases Archives

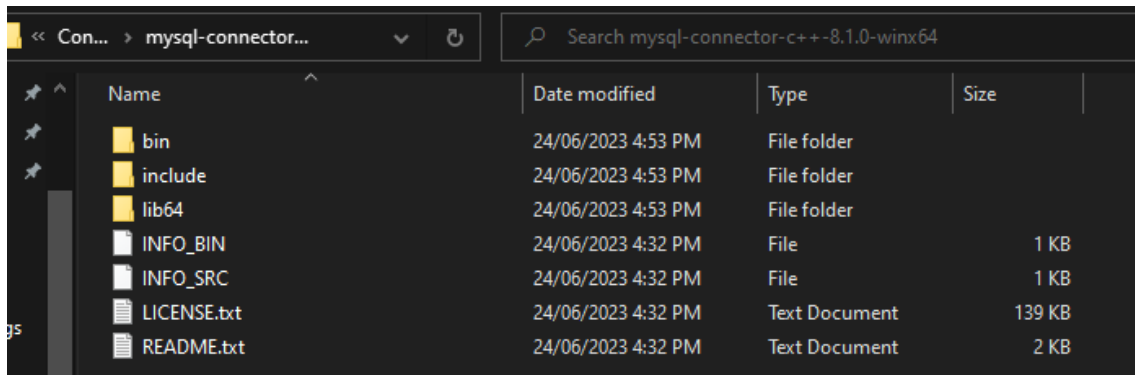
Connector/C++ 8.1.0

Select Operating System:
Microsoft Windows

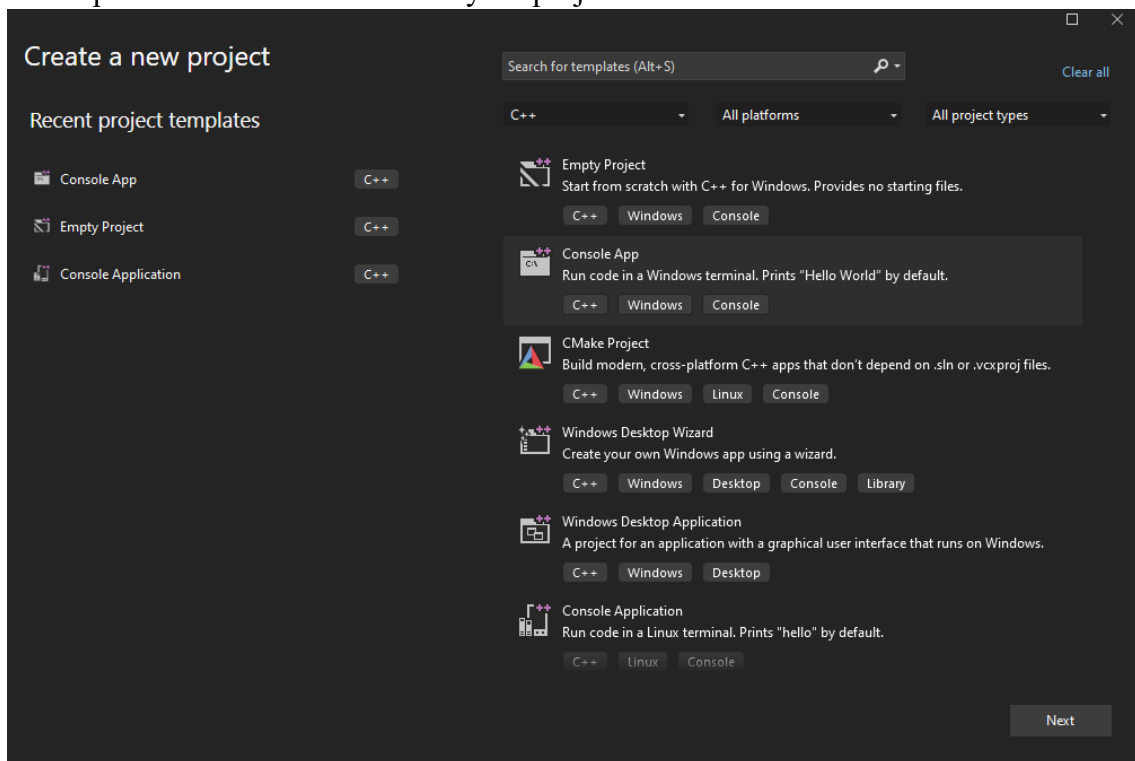
Windows (x86, 64-bit), MSI Installer (mysql-connector-c++-8.1.0-winx64.msi)	8.1.0	34.8M	Download
Windows (x86, 64-bit), ZIP Archive (mysql-connector-c++-8.1.0-winx64.zip)	8.1.0	45.2M	Download
Windows (x86, 64-bit), ZIP Archive Debug Binaries (mysql-connector-c++-8.1.0-winx64-debug.zip)	8.1.0	55.5M	Download

We suggest that you use the MD5 checksums and GnuPG signatures to verify the integrity of the packages you download.

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.

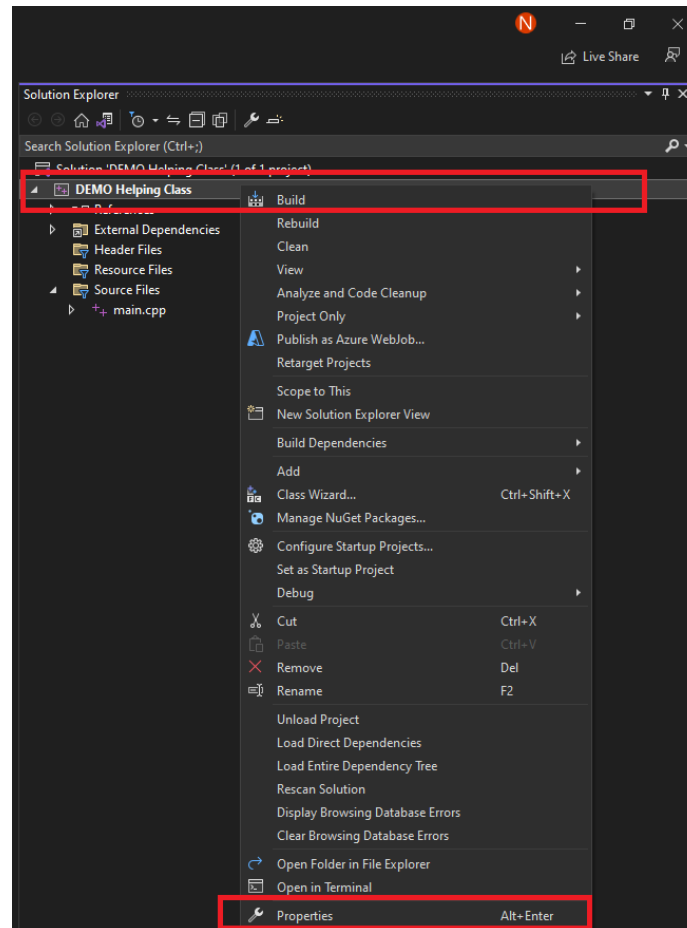


3. Now open Visual Studio and create your project.

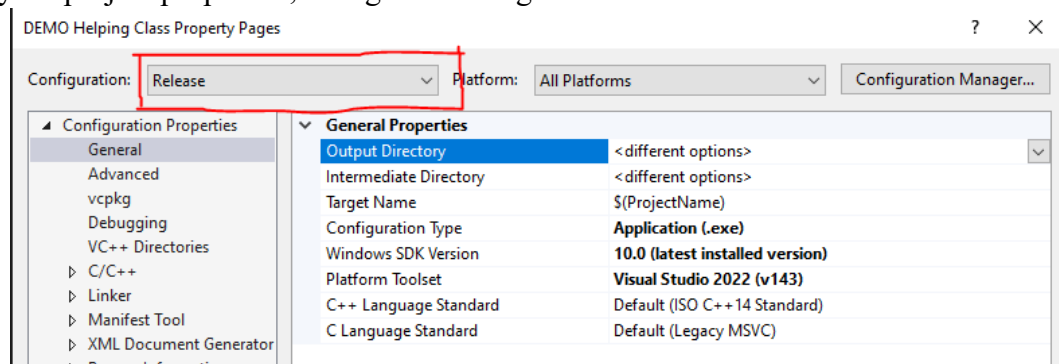


4. If you selected empty project, you might not see C/C++ option in your project property. In that case you need to create 1 .cpp file first in your project.

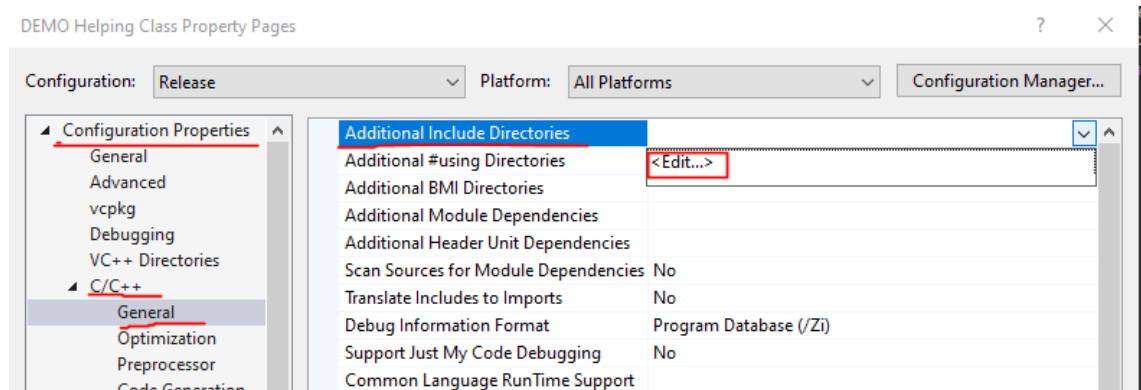
5. Go to your project properties via the solution explorer. Make sure you are right clicking the project name, not the solution since it will be different.



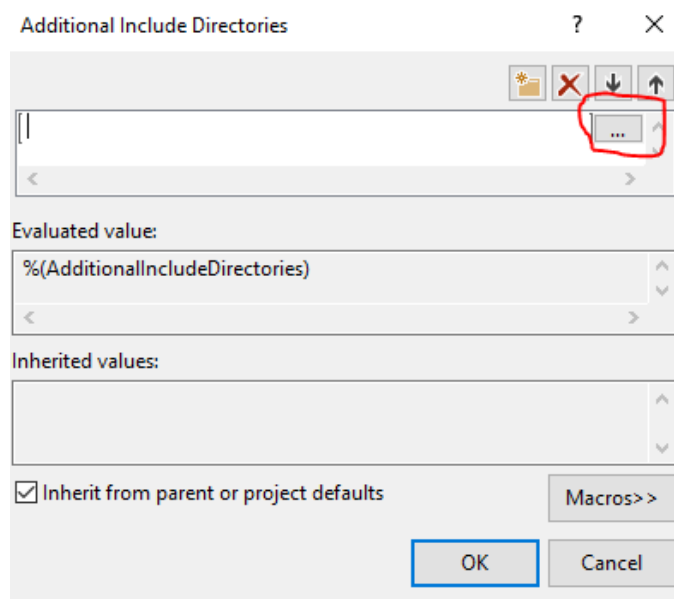
6. In your project properties, change the configuration to “release”



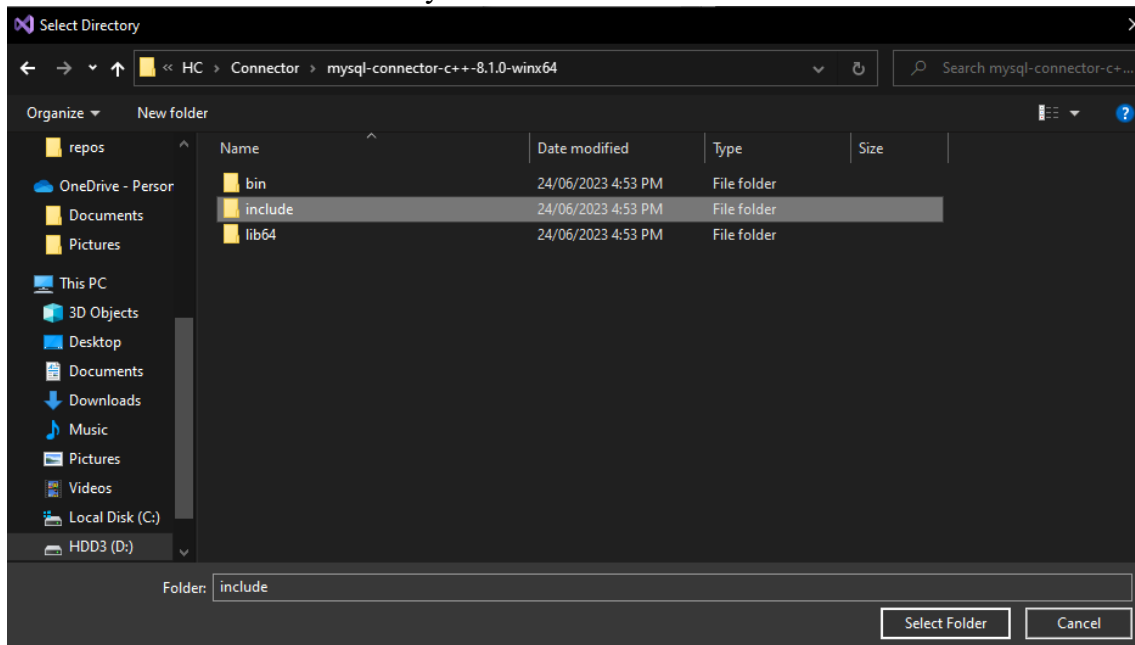
7. In your project properties, go to Configuration Properties > C/C++ > General. Then select the Additional Include Directories and click edit.



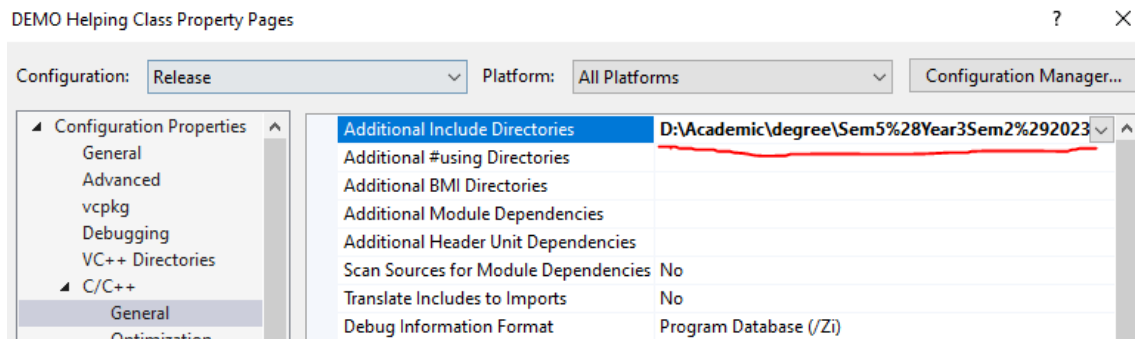
8. Then click on the empty row and lick the button at the right of it. Locate the folder which you extracted the ZIP files in step 2 previously.



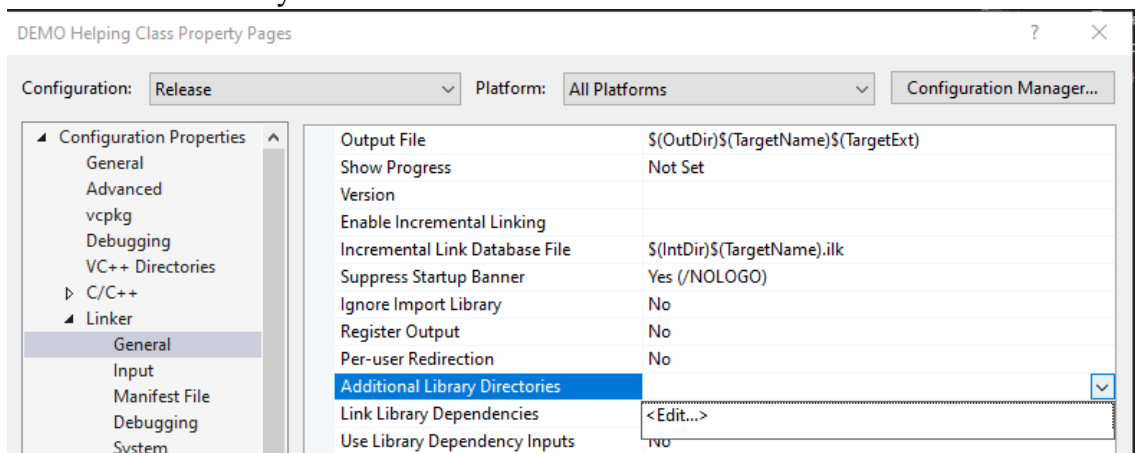
9. Select the “include” folder from your connector folder.



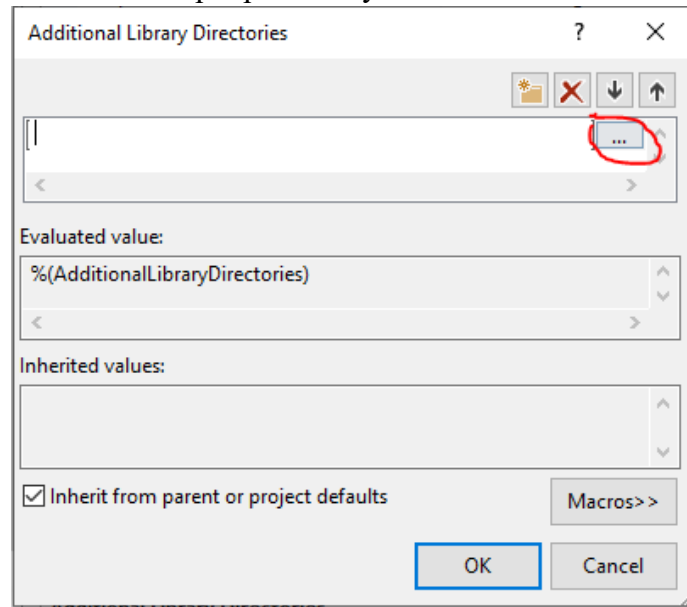
10. Press Oks. Now your project properties should look like this. Ensure that there is a value in the additional include directories.



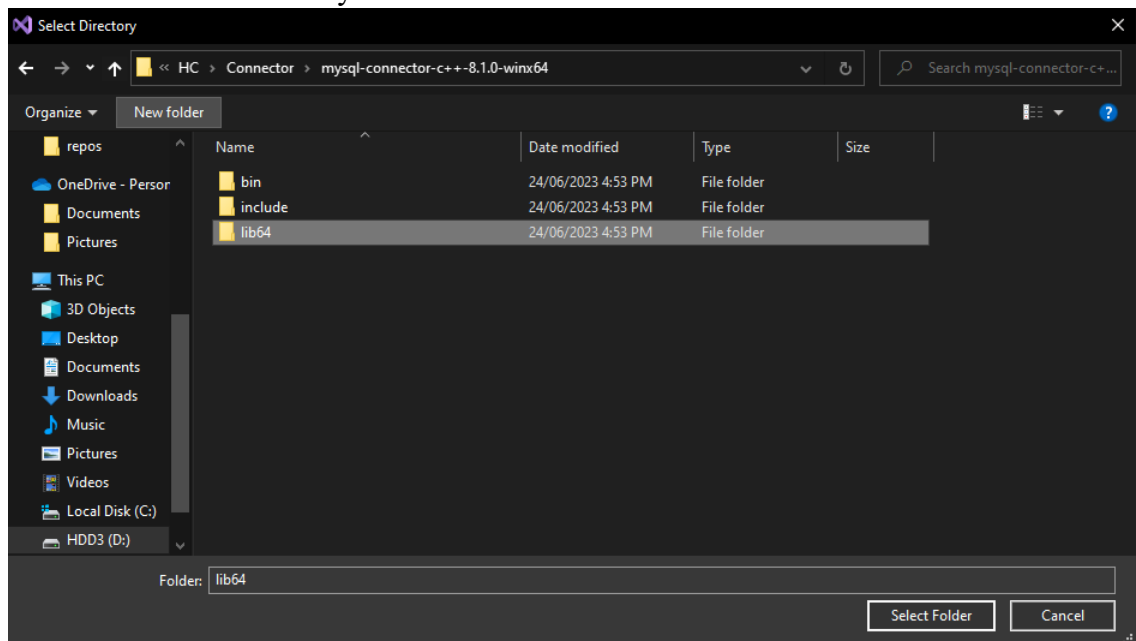
11. In your project properties go to Configuration Properties > Linker > General. Then select the Additional Library Directories to edit it.



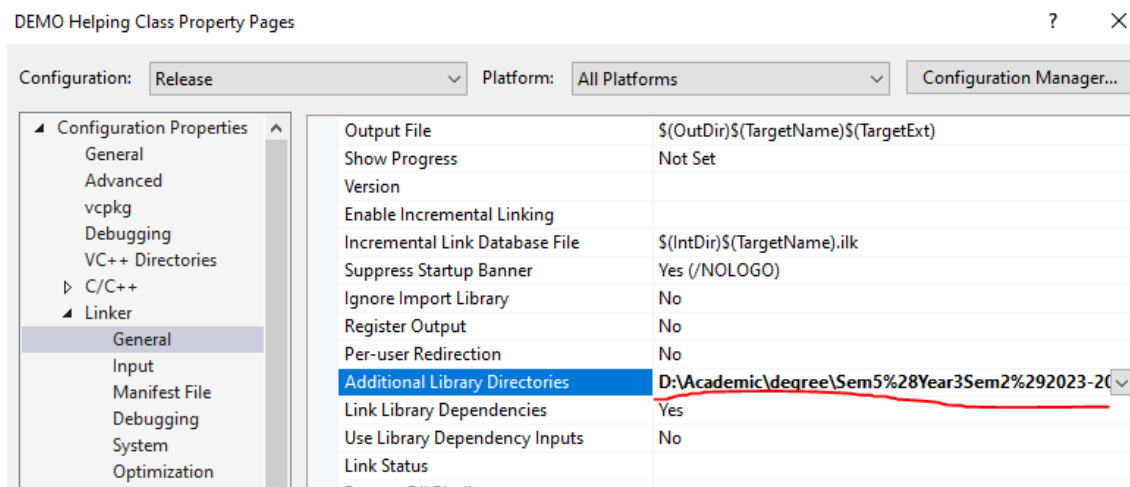
12. Then click on the empty row and lick the button at the right of it. Locate the folder which you extracted the ZIP files in step 2 previously.



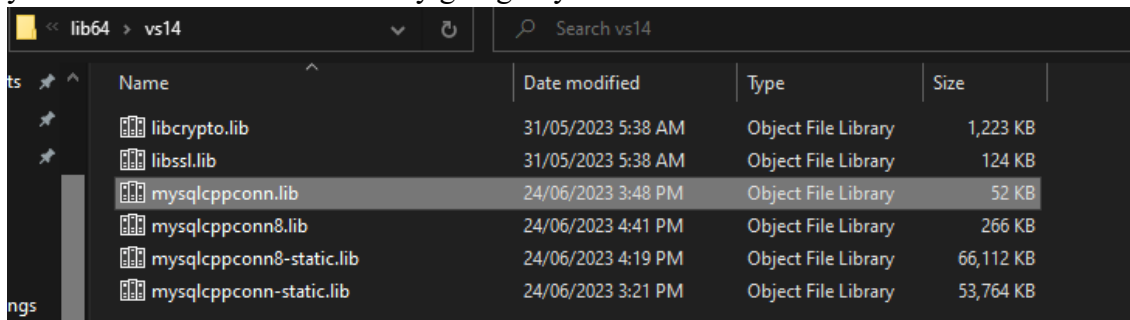
13. Select the lib folder from your connector folder.



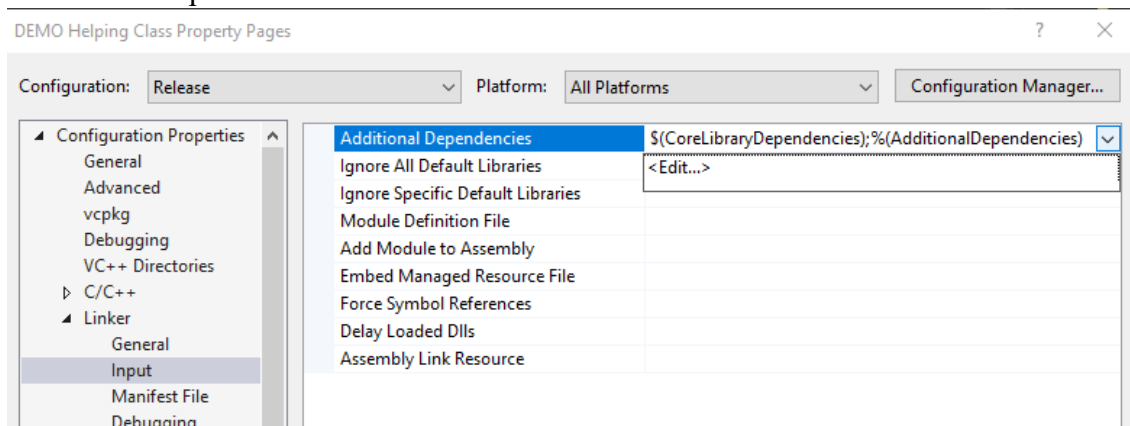
14. Click Oks. Now your project properties should look like this. Ensure that there is value in the Additional Library Directories.



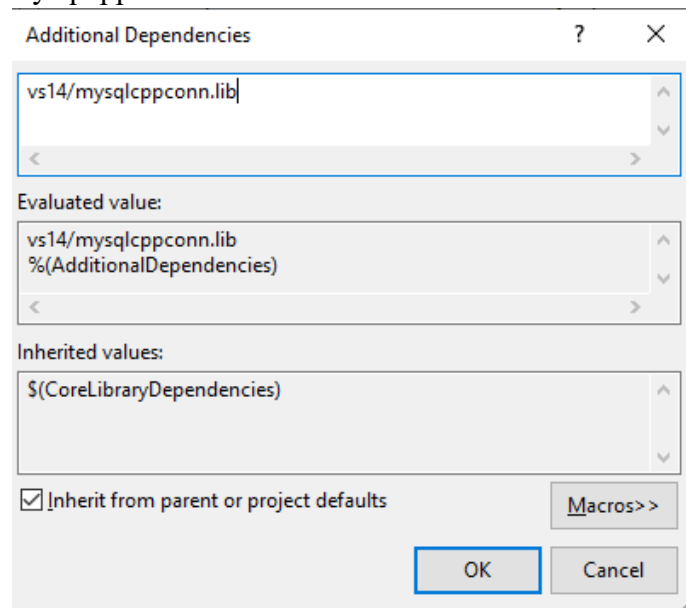
15. (optional) In this part we want to check the library dependency file name. By default, it will most likely be "mysqlcppconn.lib". But in case of MySQL themselves make changes you can confirm the file name by going to your connector folder/lib64/vs14



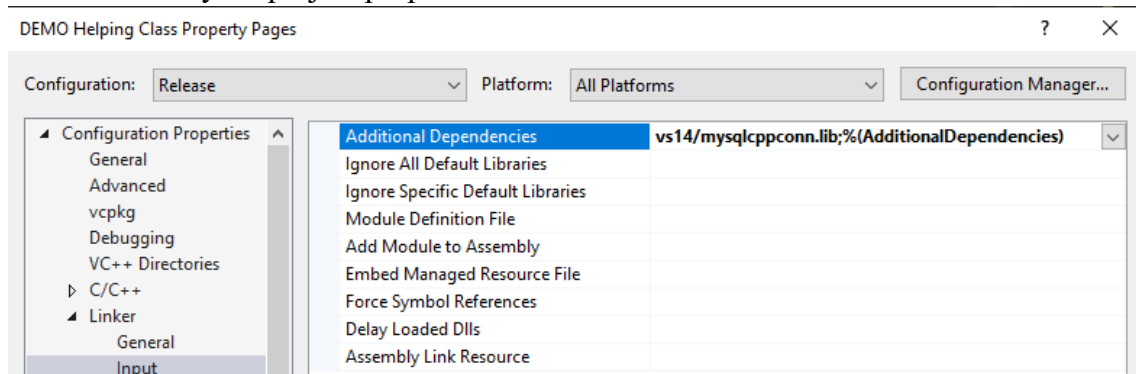
16. In your project properties. Go to Configuration Properties > Linker > Input. Select the Additional Dependencies to edit it.



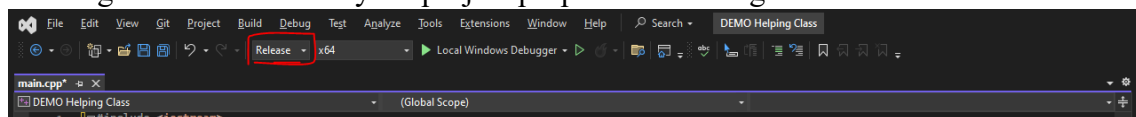
17. Then put “vs14/mysqlcppconn.lib”. This basically tells your IDE which component of the library that you added into the linker just now you would like to use in your project which in this case is the mysqlcppconn.lib



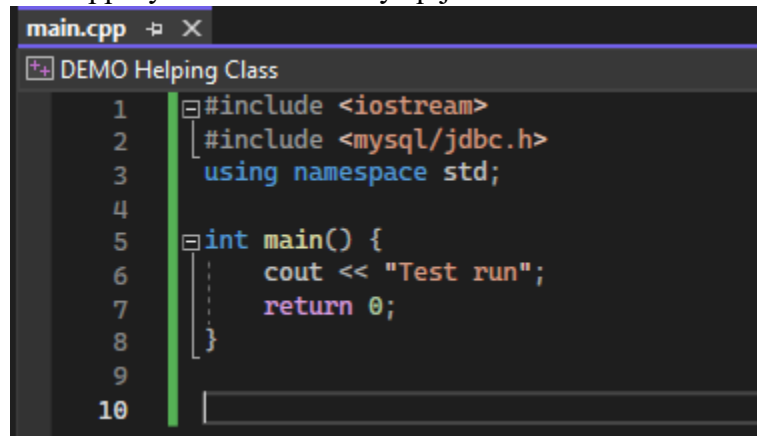
18. Press ok. Now your project properties should look like this.



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
- C/C++ > General > Additional Include Directories
 - Linker > General > Additional Library Directories
 - 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



```
main.cpp [X]
DEMO Helping Class
1  #include <iostream>
2  #include <mysql/jdbc.h>
3  using namespace std;
4
5  int main() {
6      cout << "Test run";
7      return 0;
8  }
9
10
```

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.

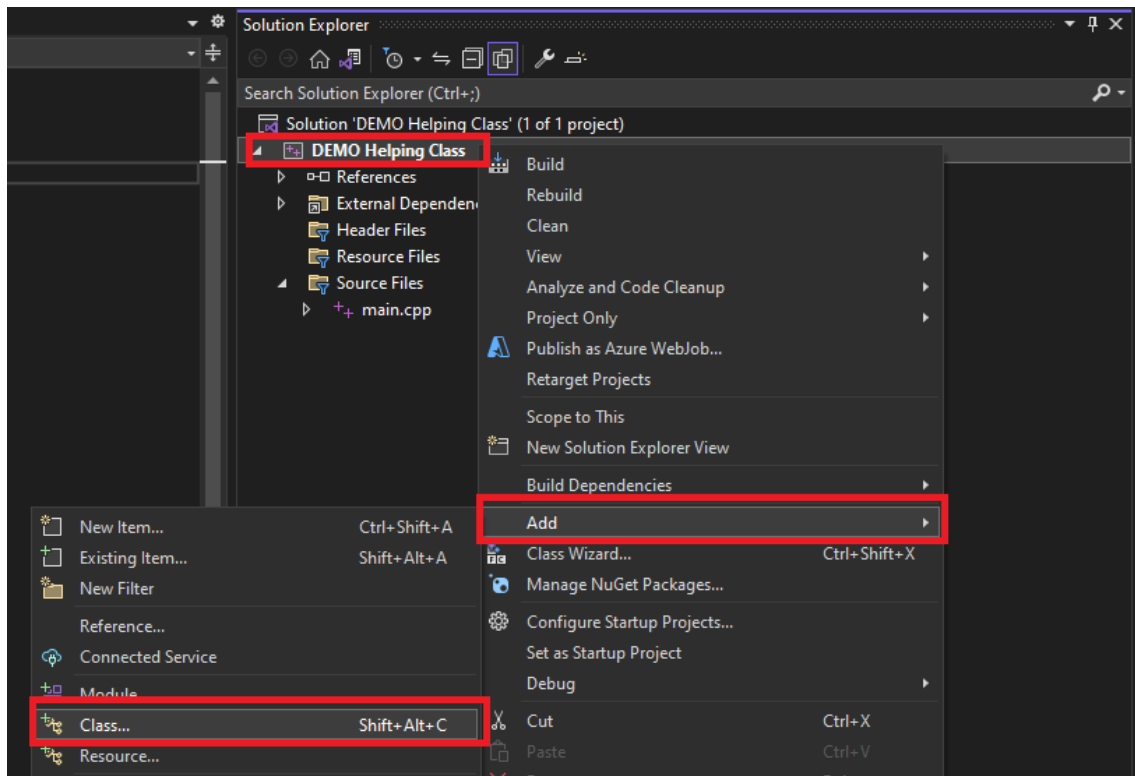
2.1.1. Class

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.

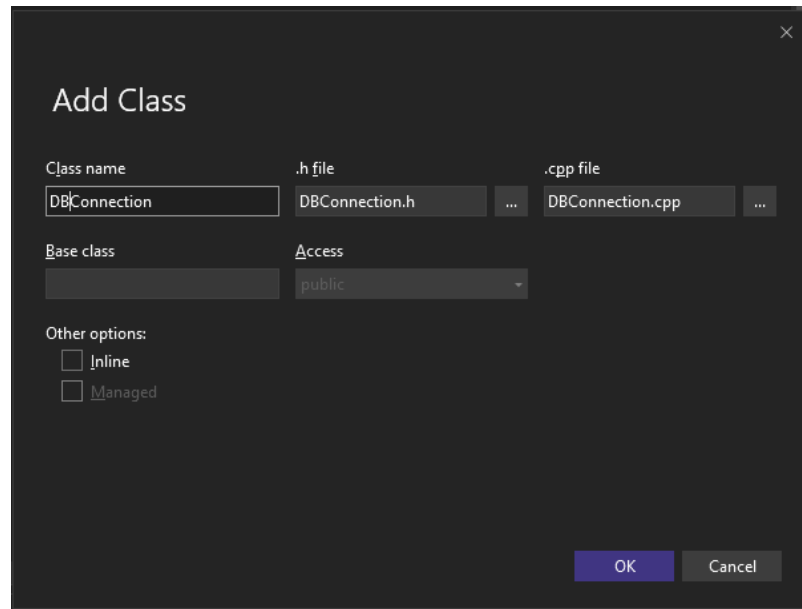
Analogically if the blueprint (**class**) of a house shows that the house will have 5 doors. Then when a house (**object**) is built (**instantiated**) according to the blueprint (**class**), all of the houses (**object**) will have the same characteristics of having 5 doors.

In C++ project you can create class by right clicking the project in the solution explorer and choose Add > class.



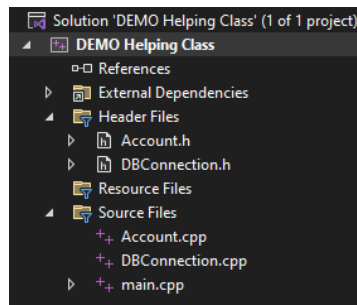
Next fill in your class name according to your needs. In this example we are creating the DBConnection class which will handle the database connection and transaction later. Although not compulsory, it is recommended to practice standardization and naming conventions while naming your classes. Basically, 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

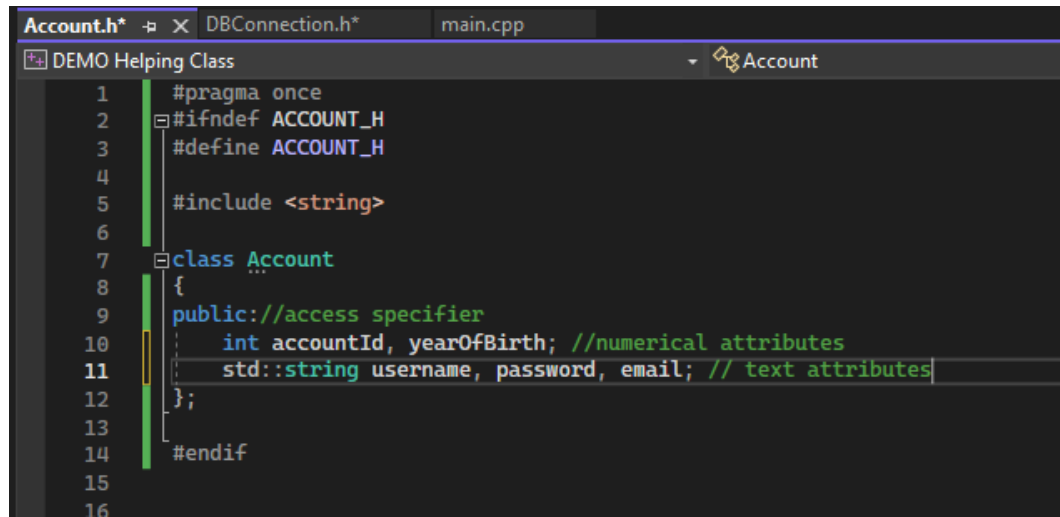
The .h file extension indicates that the file is **a header file** which in the context of C++, contains the declaration of the class members (what it should have). This is where you **declare** your class attributes and methods.

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:



```
Account.h*  DBConnection.h*  main.cpp
DEMO Helping Class  Account
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 };
13
14 #endif
15
16
```

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

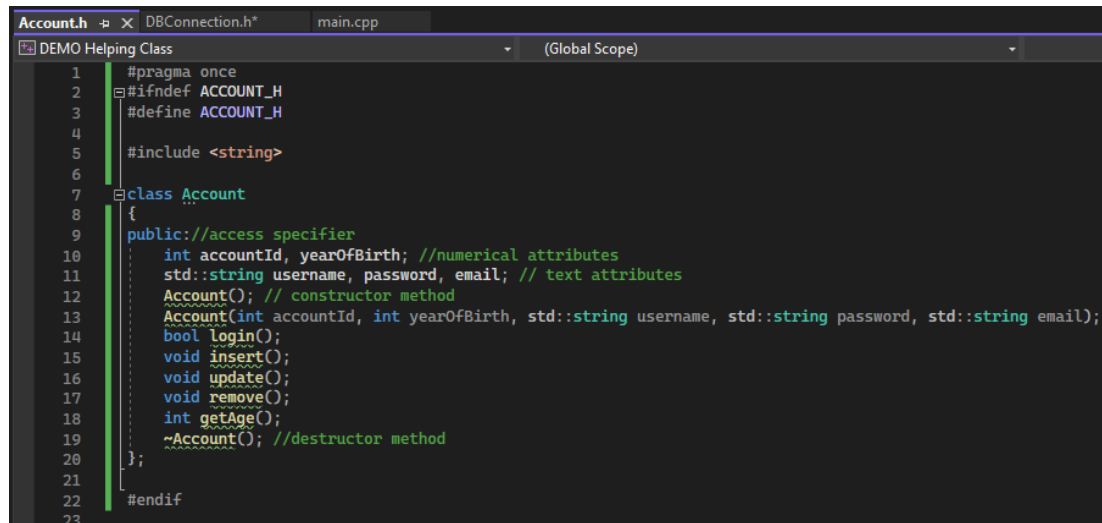
- The #pragma once (1) #ifndef CLASSNAME_H (2), #define CLASSNAME_H (3) and #endif (14) are the necessary code which tells the compiler to only compile the class once. This is necessary since there might be multiple files which include your custom class later on which may cause ambiguity error.
- public: 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.



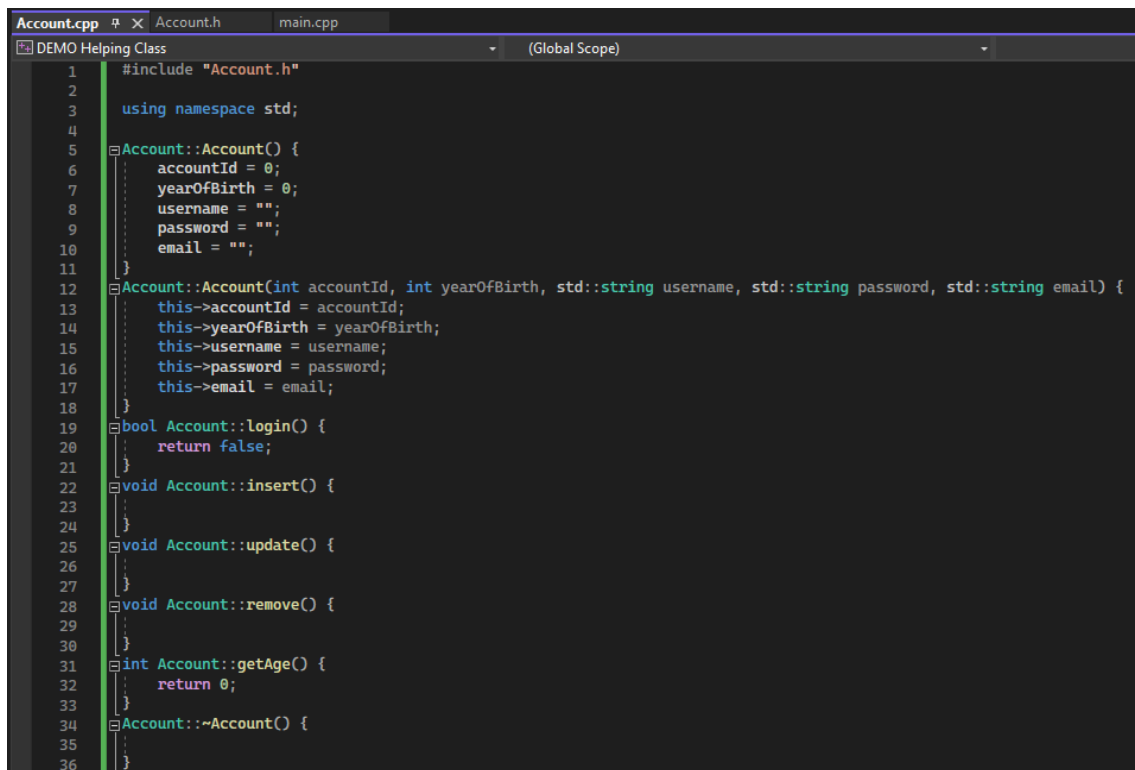
```
1 #pragma once
2 #ifndef ACCOUNT_H
3 #define ACCOUNT_H
4
5 #include <string>
6
7 class Account
8 {
9 public://access specifier
10     int accountId, yearOfBirth; //numerical attributes
11     std::string username, password, email; // text attributes
12     Account(); // constructor method
13     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:

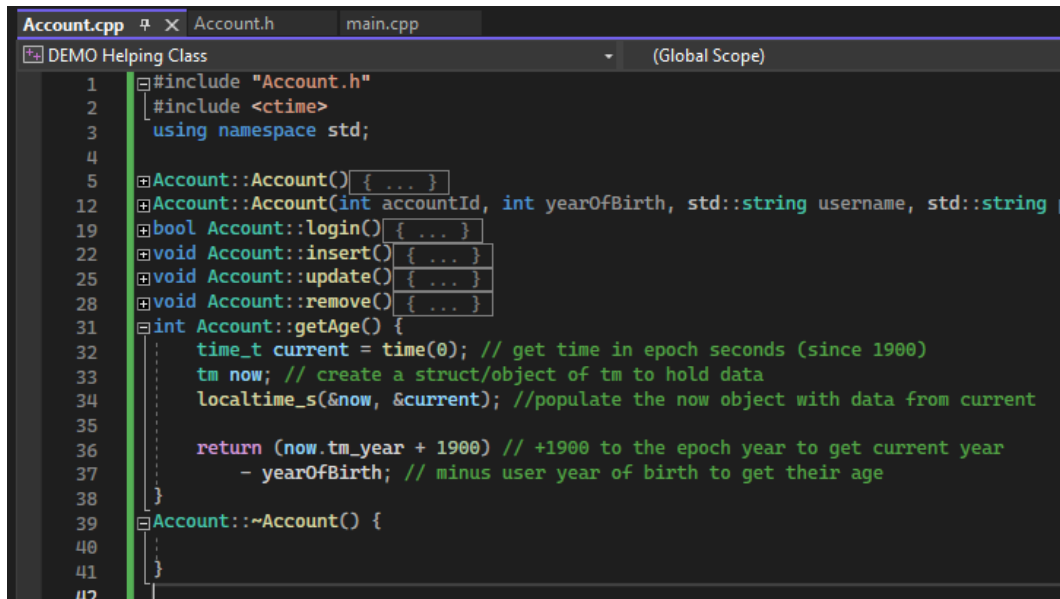


```
1  #include "Account.h"
2
3  using namespace std;
4
5  Account::Account() {
6      accountId = 0;
7      yearOfBirth = 0;
8      username = "";
9      password = "";
10     email = "";
11 }
12 Account::Account(int accountId, int yearOfBirth, std::string username, std::string password, std::string 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 void Account::update() {
25 }
26 void Account::remove() {
27 }
28 int Account::getAge() {
29     return 0;
30 }
31 Account::~Account() {
32 }
```

- #include “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 simple method 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 }
```

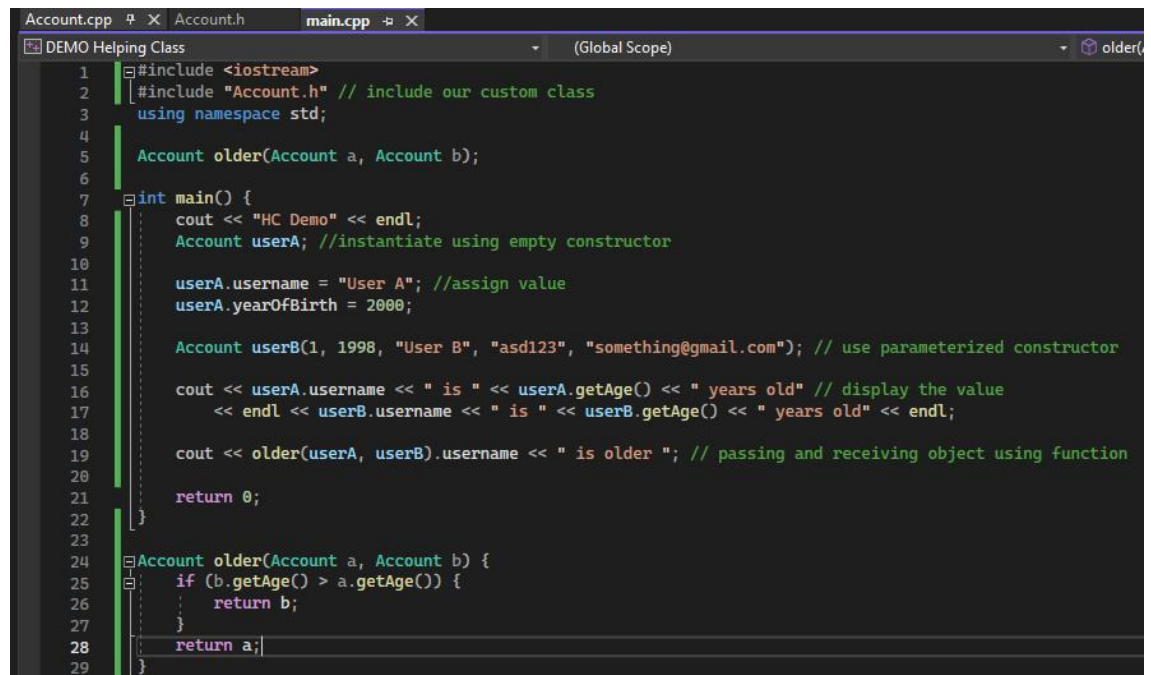
- Firstly, include the ctime library which is needed to get current time.
- Now in getAge() method, we get the current epoch time using time(0)
- Then we convert it into manipulable struct/object tm using localtime_s
- Since the time(0) returns time since 1900, we need to add 1900 to the now.tm_year 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.



```
1 #include <iostream>
2 #include "Account.h" // include our custom class
3 using namespace std;
4
5 Account older(Account a, Account b);
6
7 int main() {
8     cout << "HC Demo" << endl;
9     Account userA; //instantiate using empty constructor
10
11     userA.username = "User A"; //assign value
12     userA.yearOfBirth = 2000;
13
14     Account userB(1, 1998, "User B", "asd123", "something@gmail.com"); // use parameterized constructor
15
16     cout << userA.username << " is " << userA.getAge() << " years old" // display the value
17         << endl << userB.username << " is " << userB.getAge() << " years old" << endl;
18
19     cout << older(userA, userB).username << " is older "; // passing and receiving object using function
20
21     return 0;
22 }
23
24 Account older(Account a, Account b) {
25     if (b.getAge() > a.getAge()) {
26         return b;
27     }
28     return a;
29 }
```

- 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 note that if you are using pointer of object instead you need to access it using → for example userA→username.

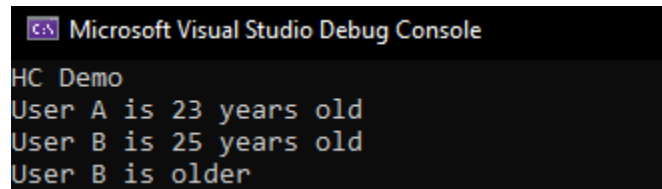
- You can **assign value** to public attributes directly as shown in line 10, 11.
- In line 13 we instantiate another object, but we passed multiple values as parameters which will make the program call the second **parameterized constructor** instead.
- Line 16 and 17 demonstrate how you can **access 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:

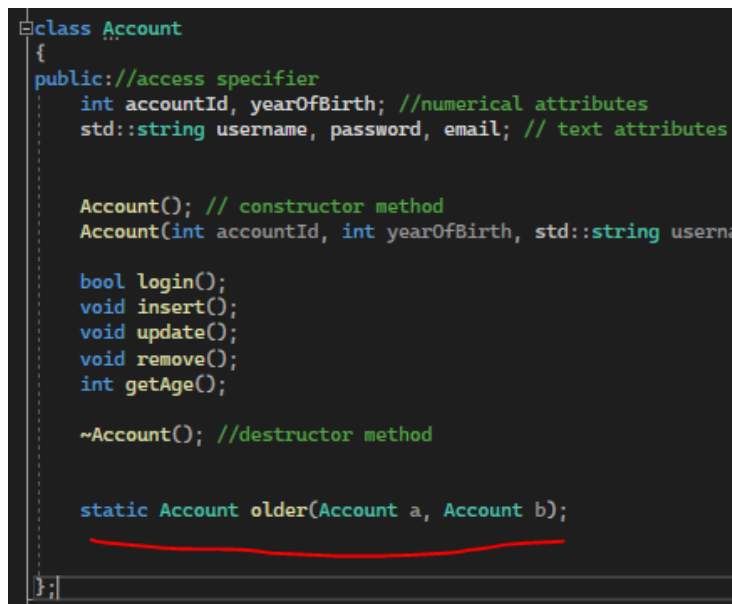


```
Microsoft Visual Studio Debug Console
HC Demo
User A is 23 years old
User B is 25 years old
User B is older
```

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



```
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 username, std::string password, std::string email);

    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.

```
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().

```
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:
 - It does not use any of the instance attributes. (cannot uses this keyword)
 - It processes something outside of the scope of single instance of object. (e.g.: older() method compare 2 object)
 - 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:

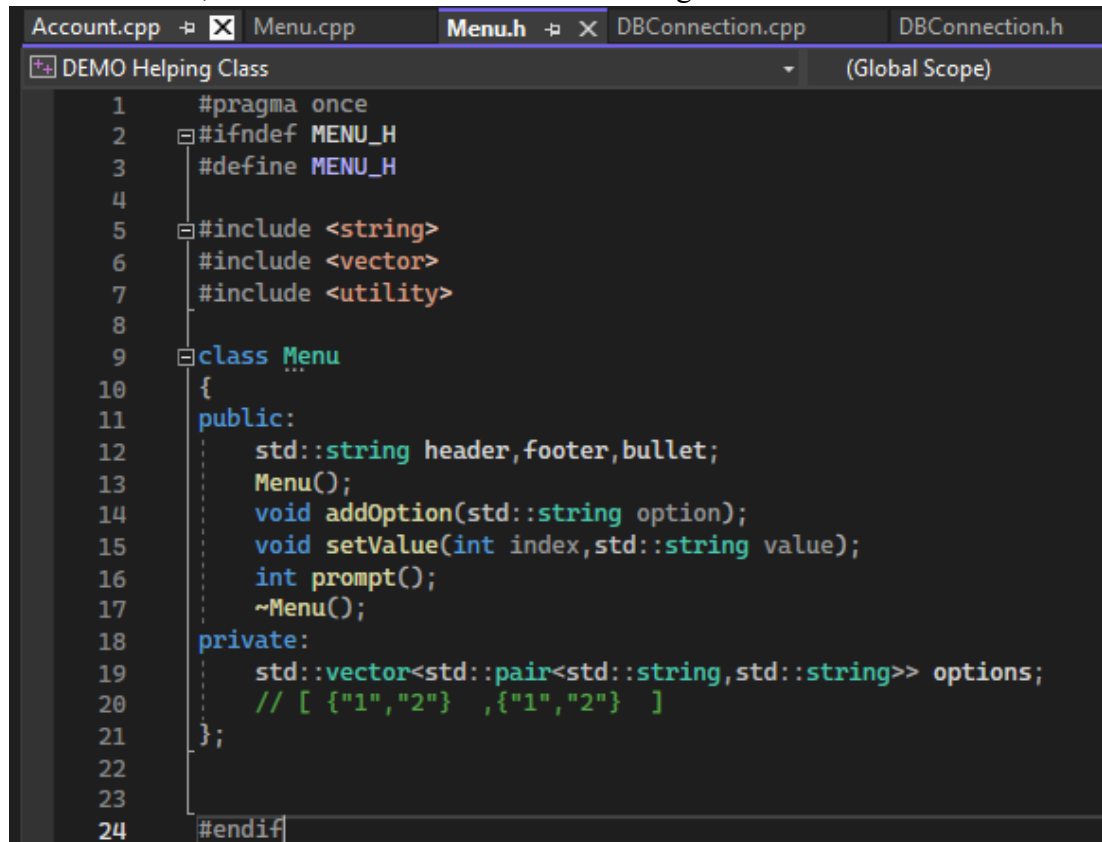
```
Welcome
1. Profile
2. Find User
3. Find Shop
Insert the number to choose:
```

```
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 within our console application.

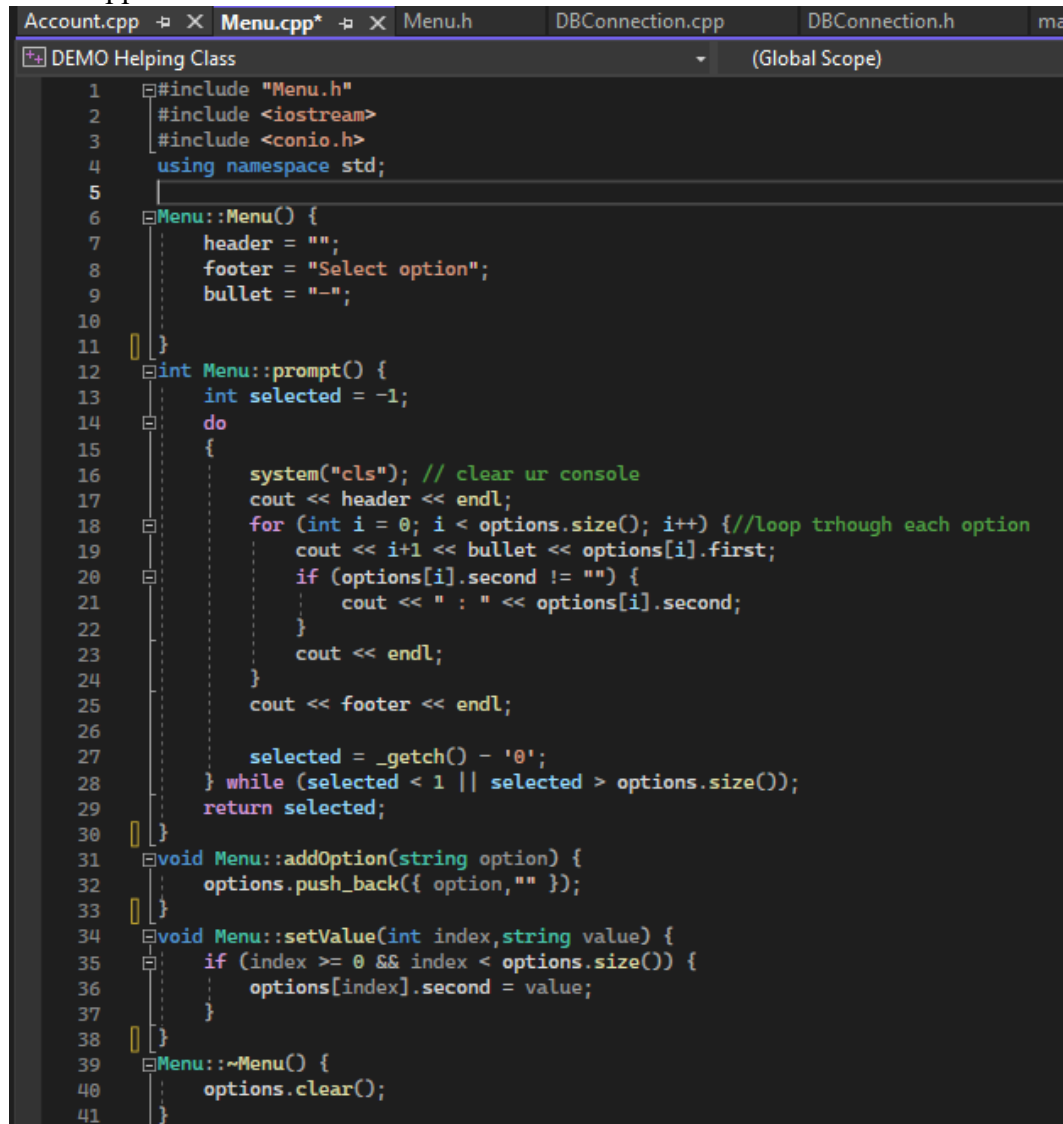
- a) Create a new class named “Menu”.
- b) The header file, Menu.h should contain the following declarations.



```
1  #pragma once
2  #ifndef MENU_H
3  #define MENU_H
4
5  #include <string>
6  #include <vector>
7  #include <utility>
8
9  class Menu
10 {
11 public:
12     std::string header, footer, bullet;
13     Menu();
14     void addOption(std::string option);
15     void setValue(int index, std::string value);
16     int prompt();
17     ~Menu();
18 private:
19     std::vector<std::pair<std::string, std::string>> options;
20     // [ {"1", "2"} , {"1", "2"} ]
21 };
22
23
24 #endif
```

- The options attribute is a vector of pair.
- **Pair is a data structure in C++** which stores pairs 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 the 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 as a pair inside a collection so that we can access it by index and accurately identify which value is for what option.

c) Menu.cpp



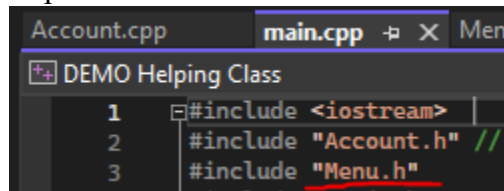
```

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 trthrough 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
32 void Menu::addOption(string option) {
33     options.push_back({ option, "" });
34 }
35
36 void Menu::setValue(int index, string value) {
37     if (index >= 0 && index < options.size()) {
38         options[index].second = value;
39     }
40 }
41
42 Menu::~Menu() {
43     options.clear();
44 }

```

- In Constructor, 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.



```
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

```
D:\Academic\degree\Sem5
LOGIN
1-username : user01
2-password : asd123
3-Login
4-Back
Select option
```

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 our user.

2.3. Database Operation

To perform 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

```
#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 the destructor method.

b) Implements the class in DBConnection.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.

```
DBConnection::~DBConnection() {
    try {
        if (connection) {
            connection->close();
        }
        if (stmt) {
            stmt->close();
        }
        if (res) {
            res->close();
        }
    }
    catch (sql::SQLException& e) {
        cout << "# ERR: SQLException in " << __FILE__ << endl;
        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 have 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. If 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.

```
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 must instantiate an object of the DBConnection class which will automatically call its constructor which establishes 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** as 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.


```

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()

```
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)

```
C:\> 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 PHPPMyAdmin

2.5. Simple Read (Login)

This section will explain 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:

```
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->rowCount() == 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)

```
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.

```
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): Dynamic Query

Refer to the source code of the demo project for complete code.

One of the most must-have features in an information system is searching. In this section, we will cover how to create a dynamic prepared statement and use it in database query.

The Product class has a constructor which receives MySQL query result set as parameter. Basically, it will directly extract the data from the result set into the object attributes during instantiation.

```
Product::Product(sql::ResultSet* data) {  
    productId = data->getInt("productId");  
    name = data->getString("name");  
    description = data->getString("description");  
    price = data->getDouble("price");  
    category = data->getInt("category");  
}
```

Next, the find product method demonstrates how you can dynamically create a query as a prepared statement.

```
vector<Product> Product::findProduct(int category, string keyword, double minPrice,  
double maxPrice, string sortColumn, bool ascending) {  
    string query = "SELECT * FROM `product` WHERE "  
        " (name LIKE ? OR description LIKE ?) AND price >= ? AND price <= ? AND category = ? "  
        " ORDER BY " + sortColumn;  
    if (ascending) {  
        query += " ASC";  
    }  
    else {  
        query += " DESC";  
    }  
  
    DBConnection db;  
    db.prepareStatement(query);  
    db.stmt->setString(1, "%" + keyword + "%");  
    db.stmt->setString(2, "%" + keyword + "%");  
    db.stmt->setDouble(3, minPrice);  
    db.stmt->setDouble(4, maxPrice);  
    db.stmt->setInt(5, category);  
  
    vector<Product> products;  
  
    db.QueryResult();  
  
    if (db.res->rowCount() > 0) {  
        while (db.res->next()) {  
            Product tmpProduct(db.res);  
            products.push_back(tmpProduct);  
        }  
    }  
  
    db.~DBConnection();  
    return products;  
}
```

- Firstly, we declared and assigned a variable of string with the query text with question marks placeholders.
- To use LIKE inside a prepared statement we do not put %?% but only ? since the percent symbol (%) itself is also considered as value.
- Then, we determine the ordering direction either ascending or descending using the value of the ascending Boolean.
- After constructing the query string, the subsequent process is similar to using regular prepared statement where we establish connection, set value and execute it.
- In the while loop where we read the result set, notice that the db.res which is the result pointer is passed into the constructor of Product class to use the constructor explained previously.
- The object instantiated in the loop is then appended to the vector of products to be stored and returned to the caller.

2.9. Formatting String

Continuing from the previous section where we have the list of products from the database, now we need to display it to the user. The most common way of displaying a list of data is in a table format which can be achieved in CLI by utilizing the feature of <iomanip> library.

```
while (1)
{
    productMenu.setValue(3, sortColumn);
    if (ascending) {
        productMenu.setValue(4, "Ascending");
    }
    else {
        productMenu.setValue(4, "Descending");
    }

    if (displayString == "") {
        displayString = "\nSearch Result:\n";
        stringstream tmpString;
        tmpString << fixed << setprecision(2) << setw(5) << "ID" << "|" << setw(20) << "Name"
        << "|" << setw(10) << "Price" << "|" << setw(20) << "Description" << "|" << endl;
        for (int i = 0; i < products.size(); i++) {
            tmpString << setw(5) << products[i].productId << "|" << setw(20) << products[i].name
            << "|" << setw(10) << products[i].price << "|" << setw(20) << products[i].description << "|" << endl;
        }
        displayString += tmpString.str();
    }
    productMenu.footer = displayString;
}
```

- Iomanip provides various features which allow you to modify behavior of a stream. You might be already familiar with using iomanip features such as setw with cout to directly format your output. But in this case, the cout doesn't happen here but inside the menu class. So, we need to assign the formatted string into menu.footer.
- Iomanip can be used to construct a formatted string by first declaring a string stream which works similarly to cout in this context but it is only data, not display.
- setPrecision is used to set the decimal place to be fixed to 2 in this stream. This works for all value that comes after it in the same stream.
- setw is used to assign a fixed length of a string. Setw(20) will reserve 20 space for only one, the first value which comes after the setw.

- Then `stringstream.str()` is called to convert whatever you put into the stream into a string which then can be used and manipulated as normal string.

2.10. Get Last Generated ID

When dealing with table that has auto incremented field, the generation of the id value for each record is up to the database system. However, sometimes we do need to know what is the id that the MySQL generated for our data.

Some implementations might opt to querying using simple select on the inserted table. This could work if we have only 1 user at a time. But **AVOID** doing it this way. Although it works now realistically our system will have multiple concurrent users. Thus, it is better to adopt the more proper method to get the generated id which will be explained in this section.

The idea is to use the “`SELECT LAST_INSERT_ID()`” query which is a SQL command available on most MySQL variants. Since this process is not bound to any specific table, we will add it as a method in the `DBConnection` class.

```
int DBConnection::getGeneratedId() {
    prepareStatement("SELECT LAST_INSERT_ID();");
    QueryResult();
    int lastInsertId = -1;
    if (res->rowCount() > 0) {
        while (res->next()) {
            lastInsertId = res->getInt64("LAST_INSERT_ID()");
        }
    }
    return lastInsertId;
}
```

Since we are inside the `DBConnection` class itself, there is no need to instantiate an object, we can use its own instance attribute and method. It works similarly with normal queries and it will return a result with 1 row and 1 column named `LAST_INSERT_ID()`. We then store this value and return it.

2.11. Bulk Insert

Throughout your project, you might have data that needs to be inserted in bulk. For example, we have a cart data consisting of multiple product id to be inserted into the `transaction_item` table. You may opt to insert it one by one using a loop, but it is much more proper if you insert it all at once, which will be demonstrated in this section.

The query for insert is still same but with multiple rows of values. The main concern is how do you put your value into the correct index of the prepared statement place holder.

Firstly, we have a new class which is `Transaction` to store transaction data.

```
#pragma once
#ifndef TRANSACTION_H
#define TRANSACTION_H

#include <string>
#include <vector>
#include "Product.h"

class Transaction
{
public:
    std::string dateTime;
    int transactionId, user;
    std::vector<std::pair<Product,int>> items; // pair of product and its quantity

    Transaction();
    void addProduct(Product product, int quantity);
    void insert();
    double total();
    int count();
};

#endif
```

The class has similar attributes to the database table it represents which has the same name. However, in this implementation we add another additional attribute which is `items`.

In cases like this you actually have the option to either create a whole different class for `transaction_item` or as shown in example just represent the `transaction_item` table as an attribute in the `transaction` class. This is **only applicable to** weak entity such as bridge table such as the `transaction_item` table. In some cases where bridge table connects more than 2 tables this might not be applicable, and you need to create another entity class for it.

Since `transaction_item` is bridge table connecting only 2 table `transaction` and `product`, we don't have to store the transaction data in `items` attribute since the attribute itself is inside a `transaction` class and wouldn't exist without a `transaction` so we already have access to

transaction data there. Now we have product and quantity data inside transaction_item. In implementation, we represent this as a pair of Product object and integer for the quantity. This way, we can accurately store transaction_item data inside items attribute.

Next, the process of saving a transaction into database can be done as shown in the following diagram.

```
11 void Transaction::insert() {
12
13     DBConnection db;
14     db.prepareStatement("INSERT INTO transaction(user) VALUES (?)");
15     db.stmt->setInt(1, user);
16     db.QueryStatement();
17     transactionId = db.getGeneratedId();
18     // get back the generated id to be used during insertion of transaction items
19
20     string query = "INSERT INTO transaction_item(transactionId,productId,quantity) VALUES ";
21     for (int i = 0; i < items.size(); i++) {
22         query += "(?, ?, ?), ";
23         // 3 place holder per item/row/record
24         // P 1,      2,      3
25         // transactionId,productId,quantity
26     }
27     query.erase(query.size() - 1); // remove the extra comma at the end
28     db.prepareStatement(query);
29
30     for (int i = 0; i < items.size(); i++) {
31         // formula for inserting the value into the right index
32         // i * N + P
33         // N is number of place holder you have per row/item in your prepared statement
34         // P is the position of the value in each row
35         // Example:
36         // i=0, 0 * 3 + 1 = 1 || 0 * 3 + 2 = 2 || 0 * 3 + 3 = 3
37         // i=1, 1 * 3 + 1 = 4 || 1 * 3 + 2 = 5 || 1 * 3 + 3 = 6
38         // i=2, 2 * 3 + 1 = 7 || 2 * 3 + 2 = 8 || 2 * 3 + 3 = 9
39
40         db.stmt->setInt(i * 3 + 1, transactionId);
41         db.stmt->setInt(i * 3 + 2, items[i].first.productId);
42         db.stmt->setInt(i * 3 + 3, items[i].second);
43     }
44     db.QueryStatement();
45     db ~DBConnection();
46 }
```

- Firstly, we insert the transaction data first which is simple and similar to previous example on inserting data. The user id is taken from the class attribute.
- Next, we get the generated id of the inserted transaction which is essential since the transaction items to be inserted next relies on this id.
- Then, we create the prepared statement to insert transaction_item
- Each row of transaction_item consists of 3 values. Thus, we use the for loop to iterate through the vector items which store the transaction_item data to create sufficient placeholders(?) in our prepared statement.
- In line 27 we use erase to remove the last character in the string since inside our loop, the string we appended will result in an additional comma at the end.
- The, we call the prepareStatement method before setting the value into the preparedStatement
- Since we might have an unknown number of placeholders, like how we generated the placeholder in preparedStatement, we also use loop to insert value into it.

- Since we have 3 placeholders (?) for each transaction_item, we multiply the iterator index, I by 3 so that we get the correct index on each iteration.
- Then we add the position of the value to the multiplied index.
- The formula and example visualization of the index are shown in the diagram.
- Now that we have prepared the statement and set the values, the execution is same as normal insert.

2.12. Read (Report): Query with Join statements.

Another usage of reading data from database is to generate reports. Producing a report using the data stored in your database often involves querying from multiple tables using JOIN statements which will be the focus of this section to explain usage of JOIN statements with prepared statements and using the data to be displayed to user.

The objective in this example is to develop a method which is reusable for multiple report generation for sales report by month.

Since we have multiple categories, we will use WHERE IN () conditional statement to include only necessary categories or include all depending on user choice.

In the salesReport method we have parameter categoryIds which is vector of integer consisting of the id of category to included in the report.

```
string categoryString = "";
if (categoryIds.size() > 0) {
    categoryString = " AND p.category IN (";
    for (int i = 0; i < categoryIds.size(); i++) {
        categoryString += "?,";
    }
    // since we add , after each placeholder we now have extra comma at the end
    categoryString.erase(categoryString.size() - 1); //remove the character at the end
    categoryString += ") "; // close bracket
}
```

In the example above, if the categoryId is empty which mean user does not specify any category, we assume that the report to be generated is for all category in which case all the process inside if will be skipped and the categoryString will simply be an empty string that doesn't affect the query. When categoryids has value, it will generate a string similar to "AND p.category IN (?,?)".

```

// construct our query for the table and joins part first
string sql = " SELECT t.dateTime as date, pc.name as categoryName, SUM(p.price * ti.quantity) as value "
    " FROM transaction t "
    " JOIN transaction_item ti ON t.transactionId = ti.transactionId "
    " JOIN product p ON p.productId = ti.productId "
    " JOIN product_category pc ON pc.categoryId = p.category "
    + categoryString; // append category string which will be our join condition if cateogryId vector is not empty
    // whereby if the categoryIds is empty this will simply be appending empty string that changes nothing

// add the where clause
sql += " WHERE t.dateTime >= ? AND t.dateTime <= ? ";

// now construct our grouping
sql += " GROUP BY ";
if ( ! categoryIds.empty() ) {
    // if categoryids is not empty only we group by category
    sql += " p.category, ";
}
// otherwise we skip the p.category to only groups it by its year and month
sql += " CAST(MONTH(t.dateTime) AS VARCHAR(2)) + '-' + CAST(YEAR(t.dateTime) AS VARCHAR(4)) ";

// now construct the sorting clause
sql += " ORDER BY ";
if (sortByDate) {
    // we have bool sortByDate parameter, if this value is true then we use date column for ordering
    sql += " t.dateTime ";
}
else {
    // otherwise we use the result column sale for ordering
    sql += " value ";
}

```

- Next, we create the query for only the select and joins part and append the category string which will be part of the join condition with product_category table.
- Then, we add where clause which is the dateTime range to be included in the report.
- Next, we have dynamic grouping. By default, this query will use the cast MONTH and cast YEAR to generate MM-YYYY string from each record dateTime and use this value to group the result. This way, we will get results grouped by month. The year part is necessary in case of user asking for report for duration that exceeds 1 year in which case we have same month but different year.
- As for the grouping by p.category, it will only happen if user specified categories to be included.

```

db.prepareStatement(sql);
// since we have non-fixed number of placeholder ? in our prepared statment we need to use a varaible to keep track of the index
int index = 1; // start from 1
// load the value for category ids
while (index <= categoryIds.size()) {
    // remember our index starts from 1 to follow prepared statement indexing
    // thus, our exit condition is until the vector size

    db.stmt->setInt(index, categoryIds[index - 1]); // and we -1 to access the item in the vector since vector index starts from 0
    index++;
}
// after the loop we have the latest index value ready to be use for our where condition placeholders (?)
// if categoryIds is empty, size 0 the previous loop will not occur and index here is still 1 so it will still be correct
db.stmt->setString(index, start);
index++; //move index forward
db.stmt->setString(index, end);
db.QueryResult();

if (db.res->rowCount() > 0) {
    while (db.res->next()) {
        Sale tmpSale(db.res);
        salesReport.push_back(tmpSale);
    }
}

```

Since we have a dynamic number of categories which user may choose, we also use looping here to put the value into its correct position in the prepared statement.

2.13. Input Validation

Input validation is a common thing to do. As a developer we have to increase the usability of our system to not only make it easier to use for a larger group of users by providing error messages as feedback to notify users of their mistake. Besides, input validation will also ensure the integrity of our data where no invalid data should be saved into the database. In this section, the two most basic types of validation will be explained.

2.13.1. Type validation

Input type validation is essential, and it concerns whether user inserted the correct input type or not. For example, let's look at the current application without validation. Users can insert invalid data into their year of birth with wrong type such as text when it supposed to be number. This causes unexpected behavior that might even crash the application. For that, we must ensure correct input before proceeding.

The most versatile data type that can hold almost anything is string, which is why it would be much safer to temporarily store our input data into a string first so that whatever user inserts it would not cause any error. Then only we check and validate the string whether it meets our requirements.

The following function will loop through the string and return false if any of the characters in the string is not digit. Although this method does work, it is not the best. There are other alternatives way which are mentioned in the additional note section.

```
bool isNumeric(string input) {
    for (int i = 0; i < input.length(); i++) {
        // loop through the string and if the character at index is not digit return false
        if (!isdigit(input.at(i))) {
            return false;
        }
    }
    // if loop finishes means all is digit so return true
    return true;
}
```

Then we can use it on our input:

```
case 4:
    cout << "Insert year of birth:";
    // cin >> temp.yearOfBirth;
    cin >> tmpInput;
    if (isNumeric(tmpInput)) {
        temp.yearOfBirth = stoi(tmpInput);
    }
    else {
        cout << "Input for year of birth must be numeric";
        _getch();
    }
    break;
case 5:
```

- Since the function returns Boolean, we can directly call it inside if ().
- When the string is numeric, we proceed to convert the string into integer using stoi and stores it into yearOfBirth attribute of our newacc object.
- Also, do not forget to inform users regarding the error to guide them on how to correct it.
- _getch() is used here to wait for user to press any key so they have time to read the error message.

2.13.2. Format validation

The format validation is more to logical requirement rather than technical. For example, a system might have rules on its password minimum 6 characters, or username minimum 10 characters or email must be in valid format etc. In this section we only will cover the length format validation as the structural format will require using regular expression (Regex) that will be covered in the additional notes.

In this example we will be checking to ensure that the user input for password must be at least 6 characters long.

```
case 2:
    cout << "Insert password:";
    cin >> tmpinput;
    if (tmpinput.length() < 6) {
        cout << "Password must be at least 6 character long";
        _getch();
    }
    else {
        newacc.password = tmpinput;
        rgMenu.setValue(1, newacc.password);
    }
    break;
```

The implementation is rather simple since we can just use the. length () method of string to validate in. Besides text input like password, we can also apply the same logic for numerical input by using temporary input string.

For example, year of birth must be exactly 4 digit long. Then we can combine the checking with our previous checking as shown in the following diagram.

```
case 4:
    cout << "Insert yearOfBirth:";
    cin >> tmpinput;
    if (isNumeric(tmpinput) && tmpinput.length() == 4) {
        newacc.yearOfBirth = stoi(tmpinput);
        rgMenu.setValue(3, to_string(newacc.yearOfBirth));
    }
    else {
        cout << "Input for year of birth must be number with 4 digit";
        _getch();
    }
    break;
```

So now the year of birth must be number with exactly four digit.

2.13.3. Range Validation

Similar to format validation, range validation is not up to technical requirements but rather your logical requirements on your data. For example, for price user must insert price greater than 0. In this example we will be doing range validation using the user age.

In the registration menu, we have validated to year of birth to ensure that it should be 4 digits of numbers only. But is this enough? Is 1000 a valid year? No, it's not since it is not logical to have user who was born at year 1000. What about when user inserts "0010"? using our simple validation which only checks type and length "0010" will pass as valid string, even "0000".

Thus, we will implement another checking on the submission/registration to ensure that user are at least 20 years old to proceed.

Since we already have `getAge()` method in account class, to check user age is fairly simple as shown in the following diagram.

```
case 5:
    valid = true;

    // 20 years old to register,
    if (newacc.getAge() < 20) {
        valid = false;
        cout << endl << "You must be at least 20 years old to register" << endl;
    }
    if (valid) {
        newacc.insert();
        cout << "Registered";
        _getch();
        return;
    }
    else {
        cout << "Please re-check your informations";
        _getch();
    }
    break;
case 6:
    return;
```

- Note that our menu is in a loop, so we declared this valid Boolean variable outside of the loop.
- On case 5 which is the "register" option, we first re-initialize this valid value to true.
- Then we perform checking using the `getAge()` method, and if its less than 20 which doesn't meet our requirements, we set the valid Boolean to false and display an error message.
- Finally, we check if the valid Boolean is true or not before proceeding to save user data using `insert()` or display error footer.
- Between re-initialization of `valid = true` and the final process, you can have any number of validation process which should change valid to false and display error message when the data does not meet the logical criteria.

- Thus, regardless of how many times you check if at least one error, the process will not proceed. Otherwise, when all validation is not error, the code will proceed to save the data into database.

This concludes the section for validation explanation. In this example, the validation is very minimal since the purpose is only to demonstrate and explain. Do note that in an actual application you will have to validate all input from user to ensure accuracy of your data and protect your system from unexpected behaviors.

3. Additional Notes

3.1. Validation using Exception handling (Try Catch)

Previously we have done basic validation using simple if. For the input type validation, we loop through each of the characters in the string to check if any of it is not digit. This approach definitely works but its implementation is limited. Not all data types have similar functions like isDigit to be used in checking each of the characters.

For example, what if the input is price with decimal place? Our validation using loop will reject the input the moment it encounters the dot “.”. Should we add is digit or character == “.” To allow dot? Then what if the user wrongly inserted string with multiple dots. There are so many things to consider when doing input validation since users’ stupidity and carelessness is unmeasurable.

Thus, this section will explain an alternative method to perform input validation using exceptions as shown in the following code.

```
// extra example validating using try catch with pointers
bool toInteger(string * input, int * valueholder) {
    // our parameter here is pointer instead of value,
    // which mean any changes done to this pointer will applies to whatever variable address we pass into it
    // so when we store stoi result to valueholder pointer we are storing the value into the memory address of variable passed into this function

    try
    {
        *valueholder = stoi(*input); //if the string fails to be converted into integer it will throw error otherwise converted integer is stored into valueholder
    }
    return true; // return true after successful conversion from string to int stored into valueholder
    catch (exception ex) {
        return false; // catch error and return false, error means string failed to be converted to integer
    }
}
```

- The purpose of this function is to convert a string into integer while also checking if the string is actually numeric or not.
- Note that the parameters are pointers (*), not value. Thus, when we refer to a pointer, we are referring to the original variable passed to this function. Basically, any changes you make to the pointer here will apply to the variable being passed, for example if you change value of valueholder, the variable passed into this parameter will have that value even though it's in different functions.
- Try block contain codes that may produce/throw an exception which is the stoi() in this case. If the string passed in the parameter is not numeric and cannot be converted into integer stoi will throw an exception which will crash and stop your application if not handled. Thus, if stoi causes error, the return true statement will not be executed.
- The catch block will handle the exception thrown from try block to prevent crashing. Additionally, we can add codes of what we want to do in case of error. In this case we only return false when error happens.
- The expected behavior of this function is, it will return true or false whether the string is numeric or not and it will also store the integer value into the variable passed as second parameter on successful conversion in case of valid string.

Example usage:

```
break;
case 4:
    cout << "Insert yearOfBirth:";
    cin >> tmpinput;
    if (toInteger(&tmpinput, &newacc.yearOfBirth) && tmpinput.length() == 4) {
        //newacc.yearOfBirth = stoi(tmpinput);
        rgMenu.setValue(3, to_string(newacc.yearOfBirth));
    }
    else {
        cout << "Input for year of birth must be number with 4 digit";
        getch();
    }
    break;
```

- Notice that we added (&) before the variable we pass into this function because the required parameters are pointed to variable address, not value. & symbol means address of. For example, &tmpinput means address of tmpinput. So, when we pass the address instead of value, the function will know where the data is actually stored and changes the value inside that same memory where the variable stores the data. Generally, this is called passing by reference which basically passing the exact same variable into the parameter.

Now applying the same logic, change the stoi to stod(string to double) to create a validation function for number with decimal places.

```
bool toDecimal(string* input, double* value) {
    try {
        *value = stod(*input);
        return true;
    }
    catch(exception ex){
        return false;
    }
}
```

You can even use stof with float with the same logic for float value.

3.2. Regular Expression

Regex is a commonly used validation technique which is not really that advanced, but it does involve learning something new for those unfamiliar. Regex is basically defining a pattern in which you want to string to match using a specific symbol and syntax to form an expression. This expression is then used to check against user input to see if it matches.

3.3. Arrow Menu

Code for this part is only available in “Additional” branch in the GitHub Repository

One of the challenges in creating a menu in CLI (command line interface) is the limitation of the CLI itself. Most implementations use simple menu selected using numbered option like what we did in the other part of the guide. However, this way of creating a menu has a clear limitation which is the number of options that you can add. In our example menu class, we use 1 digit for the option which means the menu can work for maximum of 10 options from 0 to 9.

In this section, as an additional improvement, we will implement a new type of menu which will use arrow keys to navigate through the options instead of selecting using user input value. The primary purpose of this menu is to limit the display appropriately and allow any number of options to be shown.

The following diagram is an example output of what we are trying to develop in this section. The general idea is to display the menu while highlighting the option with different color then wait for user to press arrow keys, enter or escape.

```
Welcome asraf
-----
>Profile
>Shop
>Sale Report
-----
Use Up/Down key to move selection and press enter to select
Press Escape to go back
```

Most of the other behavior of this class is very similar to the Menu class. The only significant difference is in the prompt method. Refer to the source code in the GitHub repository in “Additional” branch to get clearer view.

```
12 int ArrowMenu::prompt(int selected) {
13     char option = '\0';
14     while (1) {
15         system("cls");
16         cout << header << endl << separator << endl;
17         for (int i = 0; i < items.size(); i++) {
18             if (selected == i) {
19                 cout << "\u001b[33m"; //if it is selected option we set the console text color to yellow/gold
20             }
21             cout << bullet << items[i].first;
22             if (items[i].second.length() > 0) {
23                 cout << " : " << items[i].second;
24             }
25             cout << endl;
26             if (selected == i) {
27                 cout << "\u001b[0m"; // this code reset backs the color to default colouring
28             }
29         }
30         cout << separator << endl << footer << endl ;
31         option = _getch();
32         if (int(option) == -32) {
33             option = _getch();
34             switch (option)
35             {
36                 case 72: // 72 is the ASCII code for up arrow
37                     if (selected > 0) { // iff selected is greater than 0 and up arrow is pressed we decrease the selected index
38                         selected--;
39                     }
40                     else {
41                         selected = items.size() - 1; //if we can't decreament when user already at first item we move selection to last instead
42                     }
43                     break;
44                 case 80: // 80 is the ASCII code for down arrow
45                     if (selected < items.size() - 1) { // if selected is less than lass index we increment
46                         selected++;
47                     }
48                     else {
49                         selected = 0; // if we can't increment means that it is last item, we move selected back to the top
50                     }
51                     break;
52             }
53         }
54         else { // if the first character sent to buffer after getch is not -32 means that it is normal character so we can process accordingly
55             if (option == 27) {
56                 return -1;
57             }
58             else if (option == '\r') {
59                 return selected;
60             }
61         }
62     }
63 }
64
65
```

- Firstly, we initialize a char variable to store the user input.
- Then we loop through the items to display each of them.
- If the index of the item equals the current index, we highlight it with yellow color.
- `\u001b[33m` tells the console to start printing text in yellow.
- `\u001b[0m` tells the console to start printing text in default color (reset).
- Getch is used to wait and read user keypress. The difference here is we utilized more ASCII codes. To understand more about ASCII you can refer to other sources but basically it is the code that represents each character.
- -32 is the escape character which we use to identify if user pressing normal or special characters.
- 72 and 80 are the ASCII code of up and down arrow key. The actual characters sent to input buffer when arrow keys are pressed are like -32,72 and -32,80 because in

ASCII table, the code for 72 and 80 overlaps with normal characters. Hence the needs of -32 scape character identifier.

- 27 is ASCII code for the Escape button on keyboard which we use to break the loop and return -1 to indicate that user cancel selection.
- Option == '\r' will detect user key press on Enter key which indicates that user would like to proceed with current selection.

Then by improving the code with the following addition, we can limit the menu display so that not all the data will be displayed at once. Instead the data will be limited and user can scroll through the data using the selection.

```
    }
    limit = 10;
}

int ArrowMenu::prompt(int selected) {
    // selected is the optional paramter, by default if no value is passed we assume the initial selection is at index 0, the first item
    int before = limit / 2;
    int after = limit - before;
    char option = '\0';
    while (1) {

        system("cls");
        cout << header << endl << separator << endl;

        for (int i = 0; i < items.size(); i++) {

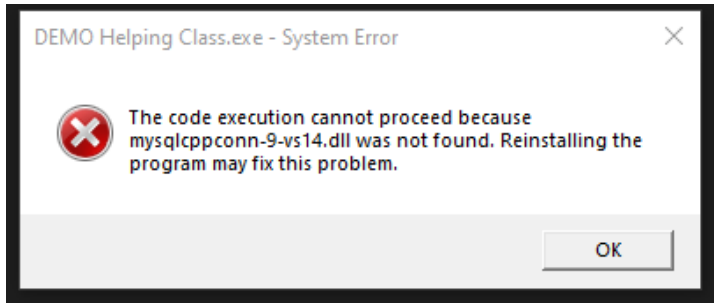
            if (selected < before) {
                if (i >= limit) {
                    continue;
                }
            }
            else if (selected >= items.size() - limit) {
                if (i < (items.size() - limit)) {
                    continue;
                }
            }
            else if (((items.size() - 1 - selected) >= limit)) {
                if (i > selected + after || i <= selected - before) {
                    continue;
                }
            }

            if (selected == i) {
```

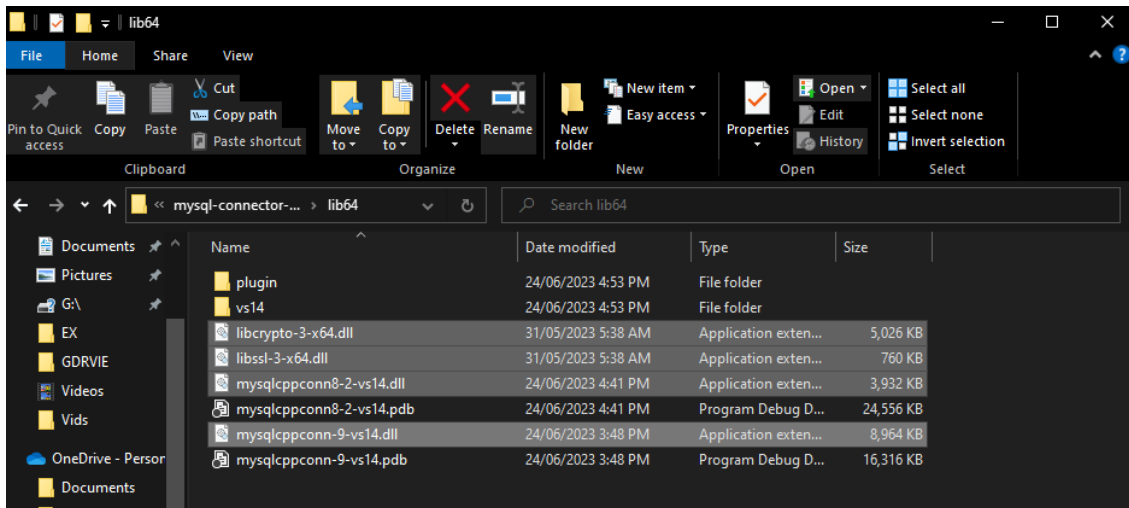
- A new attribute limit is added.
- Before displaying we calculate the number of items to display before and after the selected index.
- Then 3 if conditions are added inside the loop.
- The first if handles when selection is at the beginning of the items list.
- Second if handle when selection is at the end of the item list.
- Last if handle when selection is somewhere else in the middle.
- The idea is to use continue, to skip items at indexes that are outside of the limit that we want to display.
- For example, if selection is at index 50 of total 100 item, we only want to display 45 until 65.

3.4. mysqlcpp-9-vs14.dll not found Error.

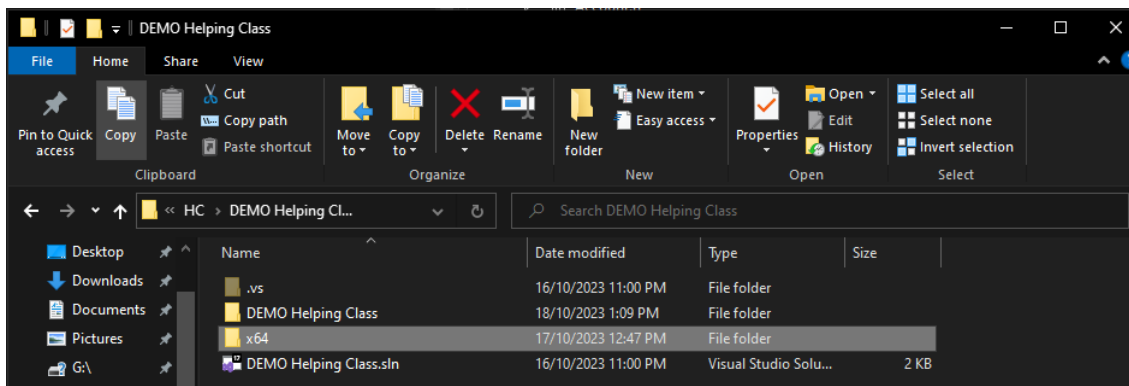
In case you encounter a similar error as shown in following diagram, you can continue to read this section which explains how to solve it.



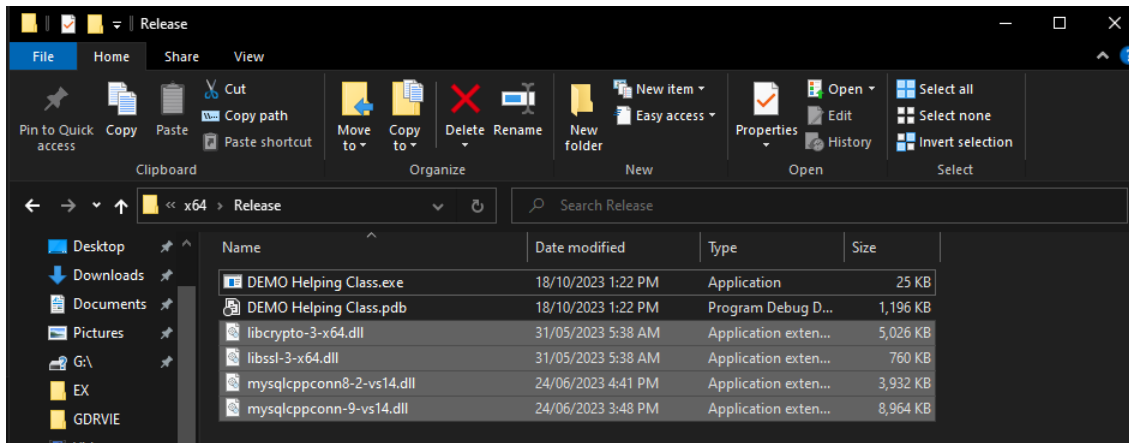
Locate the folder in which you extracted the downloaded MySQL connector and go into the lib64 folder to copy the .dll files.



Go to your project folder root directory where you can see the project name.sln file and go into the x64 folder in that same directory.

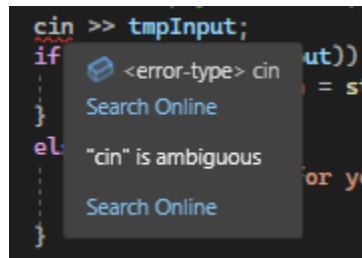


There should be a Release folder in the x64 directory, if not you can create one with the same name. Paste the .dll files you have copied here.



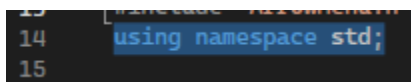
Now when you compile your program it should be able to detect the necessary .dll files.

3.5. Cin / Cout ambiguous error



This error is a very common error when developing C++ applications using visual studio. It usually happens at random. This section will show one of the possible fixes which is simple.

- In your file, find the line where you put using namespace std;



- Cut this line.
- Save your file.
- Now paste it back.
- Should've fixed the ambiguous error.

If this method does not work, you might have to find an alternative on your own for your specific issue.