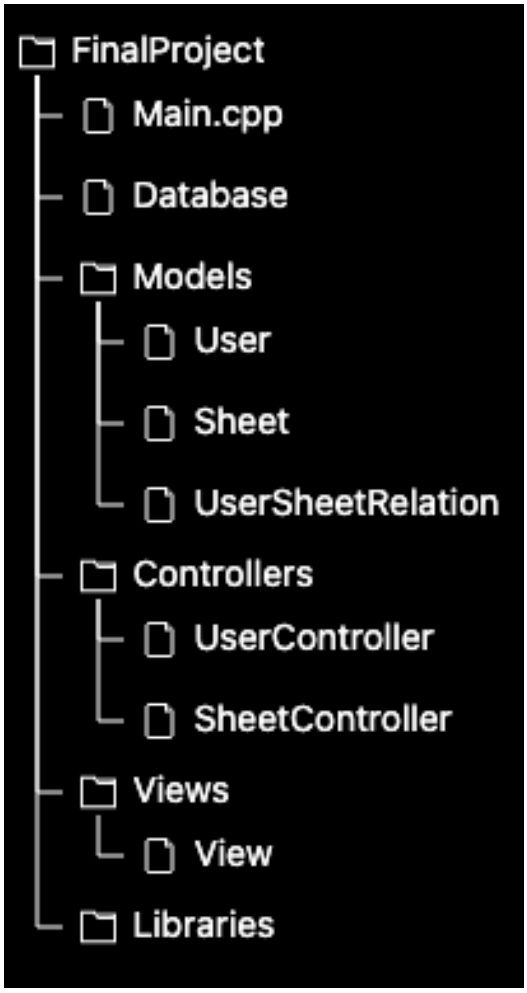
Final Project

程式語言設計 Programming Language Design



▲ Project 的 MVC 架構

How To Use

編譯：可以使用 makefile.sh 來編譯（用 g++）。

自己編譯的話，可以參考 makefile.sh 裡的指令，把所有 cpp 檔一起編譯即可。（如果是 intel 的 Mac 可以直接執行 FinalProject.out）

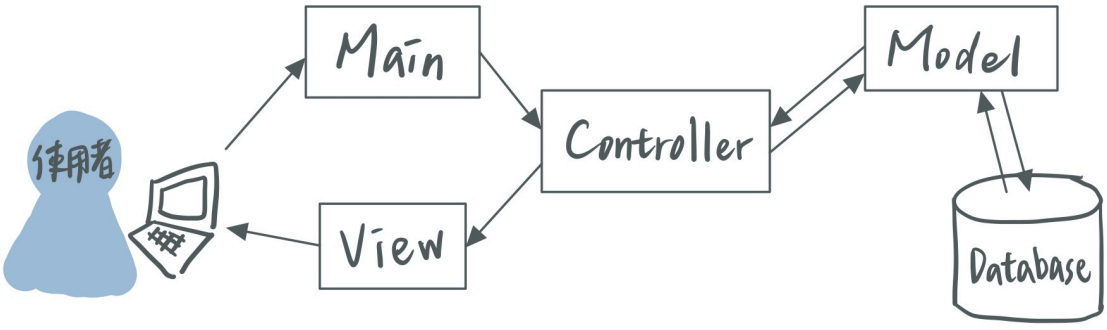
使用：和助教給的 Example 一模一樣，可以參考 example.txt（裡面放的是助教給的 Example）。

Overview

整個 project 是以 MVC 的架構模式切分，分為 Models、Controllers 以及 Views。

以 SQL 資料庫的概念設計和管理 User 和 Sheet 之間的關係。

控制第五項及第六項功能開關（5. Change a sheet's access right. & 6. Collaborate with an other user）的部份，使用 Feature Flags (a.k.a Feature Toggles) 的 Design Pattern 來控制。



Data structures

我使用 SQL Design Patterns 來設計儲存資料的形式，定義了 User、Sheet 和 UserSheetRelation 三種 model。

```
class User {
private:
    string name;

public:
    User(string name);
    string getName();
};

User *getUserByName(string name);
```

User 有一個 private attributes：name，以 name 視為 id 來區分不同的 User。

使用 Encapsulation 的概念，以 getName() 這個 public method 來拿到 private 的 name。

Sheet 有三個 private attributes：name、accessRights 和 sheetContent。name 視為 id 來區分不同的 Sheet；accessRights 紀錄現在的狀態是 editable 還是 read only；sheetContent 紀錄 sheet 裡面的值。

Sheet 一樣使用 Encapsulation 的概念來拿 private attributes。

```
class Sheet {
private:
    string name;
    string accessRights;  // "Editable" or "ReadOnly"
    float **sheetContent;

public:
    Sheet(string name);
    string getName();
    string getAccessRights();
    float **getContent();
    void setSheetContent(int row, int col, float newValue);
    void setAccessRights(string newAccessRights);
};

Sheet *getSheetByName(string name);
```

```
class UserSheetRelation {
private:
    User *editor;
    Sheet *sheet;

public:
    UserSheetRelation(User *user, Sheet *sheet);
    User *getEditor();
    Sheet *getSheet();
};

bool hasAccessTo(User *user, Sheet *sheet);
vector<Sheet *> getSheetsAccessibleByUser(string username);
vector<User *> getUsersHaveAccessToSheet(string sheetName);
```

最後一個是 UserSheetRelation，有兩個 private attributes： editor 和 sheet。這個 model 是用來管理哪些 User 可以存取哪些 Sheet。因為有 Collaborate 的功能，所以可以有很多個 User 都能存取同一個 Sheet，而一個 User 也同時可以存取很多個 Sheet，是一個多對多的關係。因此我用這個 UserSheetRelation 來記錄每一筆權限（例如有一個紀錄 editor 是 Kevin，sheet 是 SheetA，那就表示 Kevin 可以存取 SheetA）。

Controllers 定義了 UserController、SheetController 兩種。

UserController 負責 User 相關功能的操作。

```
class UserController {
private:
public:
    void create();
};
```

create() 實作 1. Create a user

```
class SheetController {
private:
    float evaluate(string expression_string);

public:
    void create();
    void check();
    void changeValue();
    void changeAccessRights();
    void collaborateWithUser();
};
```

SheetController 負責 Sheet 相關功能的操作。

create() 實作 2. Create a sheet

check() 實作 3. Check a sheet

changeValue() 實作 4. Change a value in a sheet

changeAccessRights() 實作 5. Change a sheet's access right.

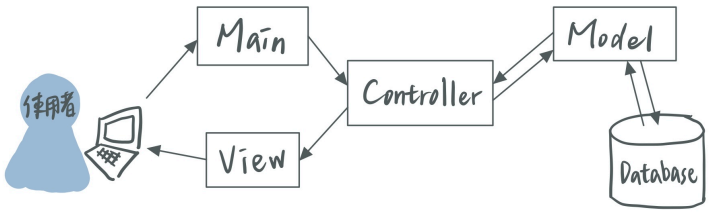
collaborateWithUser() 實作 6. Collaborate with an other user

儲存 Models 資料的地方是 Database，包含三個 Table，分別儲存三種 Models 的資料。

```
vector<User *> UserTable;
vector<Sheet *> SheetTable;
vector<UserSheetRelation *> UserSheetRelationTable;
```

程式架構

我使用 Model–View–Controller (MVC) 架構來組織這個 project。一開始使用者會先從 Main 輸入選項，之後根據選項，Main 會呼叫相對應的 Controller 來執行。而 Controller 如果需要 Database 的資料（新增、修改或查詢都算）時，Controller 會透過 Model 來操作。而 Controller 執行結束時，就會呼叫 View 來印出對應的畫面，最後再回到 Main。



根據 MVC 架構，Controllers 負責執行商業邏輯，會動到 Database 裡的資料時，一定是由 Models 來進行操作。所以 Controllers 需要 Database 的資料（新增、修改或查詢）時，Controller 會透過 Model 來操作。例如：

```
8  void UserController::create() {
9      cout << "> ";
10     string newUserName;
11     cin >> newUserName;
12
13     bool alreadyExists = true;
14     if (getUserByName(newUserName) != NULL) {
15         return printCreateUser(newUserName, alreadyExists);
16     }
17     User *newUser = new User(newUserName);
18     return printCreateUser(newUser->getName(), (!alreadyExists));
19 }
```

透過 User Model 來讀資料

透過 User Model 來新增 User

而 Controllers 執行完商業邏輯後，如果需要印出執行結果的話，就會 return 一個 View，並傳送整理好的資料給 View，由 View 來印出畫面。例如：

```
8  void UserController::create() {
9      cout << "> ";
10     string newUserName;
11     cin >> newUserName;
12
13     bool alreadyExists = true;
14     if (getUserByName(newUserName) != NULL) {
15         return printCreateUser(newUserName, alreadyExists);
16     }
17     User *newUser = new User(newUserName);
18     return printCreateUser(newUser->getName(), (!alreadyExists));
19 }
```

View

View

```
23 void printCreateUser(string newUserName, bool alreadyExists) {
24     if (alreadyExists) {
25         cout << "User \"" << newUserName << "\" already exist." << endl;
26         cout << "User name must be unique." << endl;
27     } else
28         cout << "Create a user named \"" << newUserName << "\"." << endl;
29 }
```

View

Switch On/Off Functionalities

使用 Feature Flags (a.k.a Feature Toggles) 的 Design Pattern 來控制。 Feature Flags 允許團隊或開發人員在不用修改整個程式架構的情況下改變系統的行為。

我用 canChangeAccessRights 和 canCollaborate 兩個布林值來控制是否保留「改變 Sheet 權限」的功能和「和其他 User 共用」的功能。canChangeAccessRights 是 true 的話就有「改變 Sheet 權限」的功能；canCollaborate 是 true 的話就有「和其他 User 共用」的功能。

如果要關掉特定功能的話，只要把 true 改成 false 就可以了。

```
14    bool canChangeAccessRights = true;
15    bool canCollaborate = true;
```

```
33          case 5:
34              if (canChangeAccessRights)
35                  sheetController.changeAccessRights();
36              else
37                  cout << "Unknown choice." << endl;
38              break;
39          case 6:
40              if (canCollaborate)
41                  sheetController.collaborateWithUser();
42              else
43                  cout << "Unknown choice." << endl;
44              break;
```

Design Patterns Summarization

- 使用 MVC 來作為整個程式的架構。
- 使用 SQL Design Patterns 來設計儲存資料的形式。（使用第三個 Table 來實現多對多的關係）
- 使用 Feature Flags (a.k.a Feature Toggles) 來控制是否開啟某項功能。