程式語言期末成果

使用物件導向開發單字記憶系統

VocaQuest - Vocabulary Learning System

第三組

資訊三甲楊孟憲 (組長)
學號: D1109023

資訊三甲李則瑞
學號: D1169908

資訊三甲陳宗佑
學號: D1158801

June 19, 2025

Abstract

本專案開發了一個名為 VocaQuest 的單字記憶系統，採用物件導向程式設計方法，使用 C++ 語言實作。系統提供完整的單字管理功能，包括新增、刪除、查詢單字，以及互動式測驗機制。透過模組化設計，實現了高內聚低耦合的軟體架構，展現了封裝、繼承、多型等物件導向核心概念的實際應用。本專案由三位成員共同合作完成，分工明確，展現良好的團隊協作能力。

Contents

1 專案組員與分工

1.1 組員資訊

Table 1: 專案組員名單

姓名	學號	職責
楊孟憲	D1109023	組長
李則瑞	D1169908	組員
陳宗佑	D1158801	組員

1.2 分工表

Table 2: 專案分工明細

組員	負責項目	貢獻比例
楊孟憲	專案架構設計與規劃 核心類別設計 (Vocabulary, VocaQuestApp) 物件導向架構實作 檔案 I/O 系統實作 主控制流程設計 程式碼整合與除錯 技術文件撰寫 專案管理與協調	50%
李則瑞	測驗系統設計 (Quiz 類別) 隨機演算法實作 選項生成機制 測驗流程控制	25%
陳宗佑	日誌系統設計 (Log, AlertMsg 類別) 時間戳記功能實作 訊息封裝機制 系統狀態追蹤	25%

1.3 協作流程

1. **需求分析**: 全組共同討論系統需求與功能規格
2. **架構設計**: 楊孟憲負責整體架構設計，其他成員提供意見
3. **模組開發**: 各成員負責指定模組的設計與實作
4. **整合測試**: 楊孟憲負責程式碼整合，全組共同進行測試
5. **文件撰寫**: 楊孟憲主筆技術文件，其他成員協助校對

Contents

2 專案概述

2.1 專案背景與目標

VocaQuest 是一個基於命令列介面的單字學習系統，旨在幫助使用者有效記憶與複習英文單字。系統採用物件導向設計原則，將複雜的功能分解為多個獨立且可重用的類別模組。

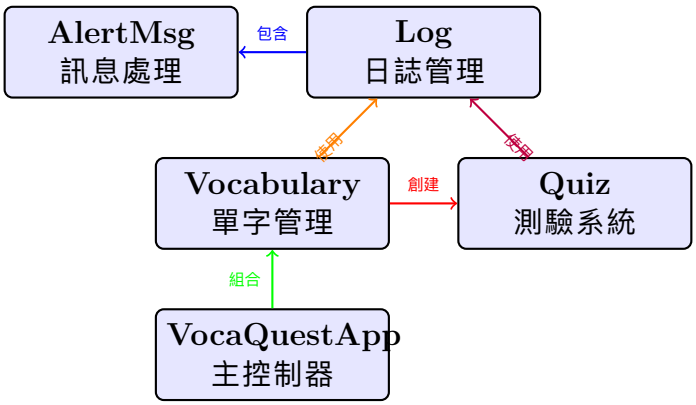
2.2 系統特色

- **完整的單字管理**: 支援單字的增刪查改操作
- **智能測驗系統**: 提供隨機選擇題測驗
- **日誌追蹤機制**: 記錄所有操作歷史
- **資料持久化**: 支援檔案讀寫功能
- **模組化設計**: 各功能模組獨立，易於維護擴充

3 物件導向設計架構

3.1 系統類別圖

本系統設計了五個核心類別，每個類別都有明確的職責分工:



類別關係說明:

- **包含關係**: Log 類別包含多個 AlertMsg 物件
- **組合關係**: VocaQuestApp 擁有 Vocabulary 物件
- **創建關係**: Vocabulary 創建 Quiz 物件進行測驗
- **依賴關係**: Vocabulary 使用 Log 記錄操作
- **依賴關係**: Quiz 使用 Log 記錄測驗過程

3.2 類別職責分析

3.2.1 AlertMsg 類別 - 訊息封裝

```
class AlertMsg {
private:
    std::string message;
    std::string curTime;

    std::string getTime();

public:
    AlertMsg(const std::string& message);

    std::string getMessage() const { return message; }
    std::string getCurTime() const { return curTime; }
};
```

Listing 1: AlertMsg 類別定義

物件導向特性展現:

- **封裝**: 將訊息內容與時間戳記封裝在一起
- **資料隱藏**: 私有成員變數保護資料完整性
- **介面設計**: 提供 getter 方法存取資料

3.2.2 Log 類別 - 日誌管理系統

```
class Log {
private:
    std::vector<AlertMsg> logs;

public:
    void push(const std::string& msg);
    void printAllLog() const;
    std::string getNewLog() const;
};

// 全域實例 - 單例模式概念
extern Log log;
```

Listing 2: Log 類別定義

設計模式應用:

- **組合關係**: Log 類別包含 AlertMsg 物件的容器
- **全域單例**: 使用全域變數實現類似單例模式的效果
- **封裝容器操作**: 隱藏 vector 的直接操作細節

3.2.3 Vocabulary 類別 - 核心業務邏輯

```
class Vocabulary {
private:
    std::string fileName;
    std::map<std::string, std::string> vocData;

public:
    Vocabulary(const std::string& fileName);

    void printCard() const;
    void addCard();
    void removeCard();
    void queryCard() const;
    void quiz();
    bool quitProgram();

    const std::map<std::string, std::string>& getVocData() const { return
        vocData; }
};
```

Listing 3: Vocabulary 類別定義

物件導向特性:

- **建構子初始化**: 透過建構子載入資料檔案
- **方法多樣性**: 提供完整的 CRUD 操作
- **const 正確性**: 區分修改性與非修改性方法
- **資料抽象**: 使用 map 容器抽象化單字資料結構

3.2.4 Quiz 類別 - 測驗邏輯封裝

```
class Quiz {
private:
    std::set<std::pair<std::string, std::string>> quiz;
    std::vector<std::pair<std::string, std::string>> vocVec;
    int amount;

    std::vector<std::string> getSelectOptions(const std::string& ans);

public:
    Quiz(int amount, const std::map<std::string, std::string>& vocData);
    void startQuiz();
    void printProblems() const;
};
```

Listing 4: Quiz 類別定義

設計特色:

- **依賴注入**: 透過建構子接收 Vocabulary 資料
- **演算法封裝**: 私有方法處理選項生成邏輯
- **資料轉換**: 將 map 轉換為適合測驗的資料結構

3.2.5 VocaQuestApp 類別 - 應用程式控制器

```
class VocaQuestApp {
private:
    Vocabulary vocabulary;

    void printRules() const;
    void processUserInput(int mode);

public:
    VocaQuestApp(const std::string& fileName);
    void run();
};
```

Listing 5: VocaQuestApp 類別定義

控制器模式:

- **組合模式**: 包含 Vocabulary 物件
- **介面統一**: 統一處理使用者互動
- **流程控制**: 管理整個應用程式生命週期

4 物件導向技術深度分析

4.1 封裝 (Encapsulation)

4.1.1 資料隱藏實作

系統中所有類別都嚴格遵循資料隱藏原則:

```
class AlertMsg {
private:
    std::string message;    // 私有資料成員
    std::string curTime;    // 私有資料成員

    std::string getTime();  // 私有輔助方法

public:
    // 公有介面方法
    std::string getMessage() const { return message; }
    std::string getCurTime() const { return curTime; }
};
```

Listing 6: 封裝示例 - AlertMsg 類別

4.1.2 介面設計原則

- **最小介面原則**: 只暴露必要的公有方法
- **const 正確性**: 讀取方法使用 const 修飾
- **返回值設計**: 適當使用 const 引用避免不必要的複製

4.2 組合與聚合關係

4.2.1 組合關係實例

```cpp
class Log {
private:
    std::vector<AlertMsg> logs;  // 組合: Log 擁有 AlertMsg 物件

public:
    void push(const std::string& msg) {
        AlertMsg newMsg(msg);    // 建立新的 AlertMsg 物件
        logs.push_back(newMsg);  // 加入容器
    }
};
```

Listing 7: Log 類別中的組合關係

4.2.2 聚合關係實例

```cpp
class Quiz {
public:
    // 聚合: Quiz 使用但不擁有 Vocabulary 的資料
    Quiz(int amount, const std::map<std::string, std::string>& vocData) {
        this->vocVec = std::vector<std::pair<std::string, std::string>>(
            vocData.begin(), vocData.end());
    }
};
```

Listing 8: Quiz 類別中的聚合關係

4.3 建構子與解構子設計

4.3.1 建構子初始化

```cpp
Vocabulary::Vocabulary(const std::string& fileName) {
    this->fileName = fileName;

    std::fstream ifs(fileName, std::ios::in);

    if (!ifs.good()) {
        log.push("檔案開啟失敗!");
        return;
    }

    std::string eng, zh;
    while(ifs >> eng >> zh) {
        if (vocData.count(eng) && vocData[eng] != zh) {
            log.push("資料庫資料異常");
            return;
        }
        vocData[eng] = zh;
    }
    ifs.close();
}
```

Listing 9: Vocabulary 建構子實作

建構子設計特色:

- **資源初始化**: 自動載入資料檔案
- **錯誤處理**: 處理檔案讀取異常
- **資料驗證**: 檢查資料一致性
- **日誌記錄**: 整合日誌系統

4.4 STL 容器的物件導向應用

4.4.1 容器選擇策略

```
class Vocabulary {
private:
    // map: 快速查找，鍵值對應
    std::map<std::string, std::string> vocData;
};

class Quiz {
private:
    // set: 避免重複，自動排序
    std::set<std::pair<std::string, std::string>> quiz;
    // vector: 隨機存取，順序儲存
    std::vector<std::pair<std::string, std::string>> vocVec;
};

class Log {
private:
    // vector: 順序儲存，支援尾端插入
    std::vector<AlertMsg> logs;
};
```

Listing 10: 不同容器的使用場景

5 核心功能實作分析

5.1 單字管理系統

5.1.1 新增功能

```
void Vocabulary::addCard() {
    std::string eng, zh;
    std::cout << "請輸入英文: ";
    std::cin >> eng;
    std::cout << "請輸入中文: ";
    std::cin >> zh;

    if (vocData.count(eng) && vocData[eng] != zh) {
        log.push("插入失敗( 資料庫已有資料，無法對應 )");
        std::cout << "插入失敗( 資料庫已有資料，無法對應 )";
        return;
    }

```

```
    vocData[eng] = zh;
    log.push("插入成功");
}
```

Listing 11: 單字新增功能實作

5.1.2 查詢功能

```
void Vocabulary::queryCard() const {
    std::string eng;
    std::cout << "請輸入英文單字: ";
    std::cin >> eng;
    std::cout << "查詢結果:\n";

    if (!vocData.count(eng)) {
        log.push("查無 " + eng + " 此單字!");
        return;
    }

    std::cout << "英文: " << eng << "\n";
    std::cout << "中文: " << vocData.at(eng) << "\n";
    log.push("成功查詢 " + eng + " 單字");
}
```

Listing 12: 單字查詢功能實作

5.2 測驗系統設計

5.2.1 題目生成演算法

```
std::vector<std::string> Quiz::getSelectOptions(const std::string& ans) {
    std::set<std::string> options{ans};  // 包含正確答案
    const int amountOfOptions = 4;

    while(options.size() < amountOfOptions) {
        int randomIdx = rand() % vocVec.size();
        if (!options.count(vocVec[randomIdx].second)) {
            options.insert(vocVec[randomIdx].second);
        }
    }

    std::vector<std::string> optionVec(options.begin(), options.end());
    return optionVec;
}
```

Listing 13: 選擇題選項生成

5.2.2 測驗流程控制

```
void Quiz::startQuiz() {
    int problemId = 1;
    int correctAmount = 0;

    for(auto it = quiz.begin(); it != quiz.end(); it++, problemId++) {
```

```
        system("clear");
        printf("題號:  %d  |  (答對: %d | 已完成: %d)\n\n\n",
               problemId, correctAmount, problemId);

        std::vector<std::string> options = getSelectOptions(it->second);
        int userChoose;

        std::cout << "請選擇 " + it->first + " 的中文:\n\n";

        for(int i = 0; i < options.size(); i++) {
            std::cout << "(" << i + 1 << "): " << options[i] << "\n\n";
        }

        std::cout << "你的選擇: ";
        std::cin >> userChoose;

        if (options[userChoose - 1] != it->second) {
            std::cout << "答錯了! 正確答案為: " << it->second << "\n";
        } else {
            std::cout << "答對了!!\n";
            correctAmount++;
        }

        getchar(); getchar();  // 等待使用者按鍵
    }
}
```

Listing 14: 測驗主要流程

5.3 檔案持久化機制

5.3.1 資料載入

系統啟動時自動從檔案載入單字資料，展現建構子的資源管理能力。

5.3.2 資料儲存

```
bool Vocabulary::quitProgram() {
    std::string ss;
    std::cout << "確定要退出程式嗎?(yes/no): ";
    std::cin >> ss;

    if (ss == "no" || ss == "No" || ss == "nO" || ss == "NO")
        return false;

    std::fstream ofs(fileName, std::ios::out | std::ios::trunc);
    if (!ofs.good()) {
        log.push("結束失敗!");
        return false;
    }

    for(const auto& card : vocData) {
        ofs << card.first << " " << card.second << "\n";
    }

    ofs.close();
```

```
    std::cout << "退出成功!\n";
    return true;
}
```

Listing 15: 程式結束時儲存資料

6 系統使用說明

6.1 編譯與執行

6.1.1 編譯環境需求

- C++11 或更新版本的編譯器
- 支援標準庫 STL
- Make 工具（可選）

6.1.2 編譯指令

```
make
```

Listing 16: 使用 Makefile 編譯

```
g++ -std=c++11 -Wall -Wextra -o vocaquest \
    main.cpp AlertMsg.cpp Log.cpp Quiz.cpp \
    Vocabulary.cpp VocaQuestApp.cpp
```

Listing 17: 手動編譯指令

6.2 程式執行

```
./vocaquest
```

Listing 18: 執行程式

6.3 功能操作說明

6.3.1 主選單

程式啟動後會顯示以下選項:

1. 新增單字卡片
2. 打印所有卡片
3. 刪除單字卡片
4. 查詢單字卡片
5. 開始考試
6. 結束程式（保存資料 | 退出）
7. 查詢系統狀態

6.3.2 測驗模式

- 可自訂題目數量
- 四選一選擇題形式
- 即時答題回饋
- 答題統計功能

7 程式效能分析

7.1 時間複雜度分析

Table 3: 主要操作的時間複雜度

操作	資料結構	時間複雜度
單字查詢	std::map	O(log n)
單字新增	std::map	O(log n)
單字刪除	std::map	O(log n)
測驗題目生成	std::set + std::vector	O(k log n)
日誌記錄	std::vector	O(1)

7.2 空間複雜度分析

- **單字儲存**: O(n)，其中 n 為單字數量
- **日誌系統**: O(m)，其中 m 為操作次數
- **測驗系統**: O(k)，其中 k 為題目數量

8 設計模式應用

8.1 單例模式

雖然沒有嚴格實作單例模式，但透過全域 log 變數實現了類似效果:

```
// Log.cpp
Log log; // 全域實例定義

// Log.h
extern Log log; // 外部宣告
```

Listing 19: 全域日誌實例

8.2 工廠模式概念

AlertMsg 的建立展現了簡單工廠的概念:

```
void Log::push(const std::string& msg) {
    AlertMsg newMsg(msg);  // 自動建立帶時間戳的訊息
    logs.push_back(newMsg);
}
```

Listing 20: AlertMsg 物件建立

9 錯誤處理機制

9.1 檔案操作錯誤

```
std::fstream ifs(fileName, std::ios::in);

if (!ifs.good()) {
    log.push("檔案開啟失敗!");
    std::cout << "檔案開啟失敗!\n";
    return;
}
```

Listing 21: 檔案讀取錯誤處理

9.2 資料一致性檢查

```
if (vocData.count(eng) && vocData[eng] != zh) {
    log.push("資料庫資料異常");
    std::cout << "資料庫資料異常";
    return;
}
```

Listing 22: 資料驗證機制

10 學習心得與反思

10.1 物件導向設計收穫

透過此專案的開發，深刻體會到物件導向程式設計的優勢:

10.1.1 模組化的好處

- **職責明確**: 每個類別都有清楚的功能定位
- **易於維護**: 修改某個功能不會影響其他模組
- **可重用性**: AlertMsg 類別可在不同地方使用
- **擴展性**: 可輕易添加新的測驗模式或單字格式

10.1.2 封裝的重要性

- **資料保護**: 私有成員避免外部直接修改
- **介面穩定**: 公有方法提供穩定的操作介面
- **實作隱藏**: 內部演算法變更不影響使用者

10.2 C++ STL 的活用

- **容器選擇**: 根據需求選擇最適合的資料結構
- **演算法效率**: 充分利用 STL 提供的高效演算法
- **迭代器概念**: 統一的元素存取方式

10.3 程式設計技巧

- **const 正確性**: 適當使用 const 增加程式安全性
- **RAII 原則**: 透過建構子和解構子管理資源
- **錯誤處理**: 完善的例外狀況處理機制

11 未來改進方向

11.1 功能擴充

- **多種測驗模式**: 填空題、拼字測驗等
- **學習進度追蹤**: 記錄使用者學習歷程
- **單字分類功能**: 依主題或難度分組
- **圖形化介面**: 使用 Qt 或其他 GUI 框架

11.2 技術優化

- **資料庫整合**: 使用 SQLite 替代文字檔
- **網路功能**: 線上同步與分享功能
- **多執行緒**: 提升大量資料處理效能
- **設計模式**: 導入更多設計模式提升架構品質

11.3 使用者體驗

- **命令列優化**: 更友善的互動介面
- **設定檔支援**: 個人化設定功能
- **多語言支援**: 國際化介面設計

12 結論

VocaQuest 單字記憶系統成功展現了物件導向程式設計的核心概念與實作技巧。透過適當的類別設計、封裝機制、以及模組化架構，創造了一個功能完整且易於維護的應用程式。

本專案不僅實現了預期的功能需求，更重要的是在開發過程中深入理解了物件導向的設計哲學。從需求分析、類別設計、到程式實作，每個階段都體現了軟體工程的重要原則。

未來將持續改進系統功能，並探索更多進階的程式設計技術，使 VocaQuest 成為更加優秀的學習工具。這個專案為後續的軟體開發學習奠定了堅實的基礎，也讓我對程式設計有了更深層的認識與熱忱。

A 完整程式碼清單

A.1 專案檔案結構

```
VocaQuest/
|-- src/                        # 源碼目錄
|   |-- main.cpp                # 程式進入點
|   |-- AlertMsg.h/.cpp         # 訊息處理類別
|   |-- Log.h/.cpp              # 日誌管理類別
|   |-- Quiz.h/.cpp             # 測驗系統類別
|   |-- Vocabulary.h/.cpp       # 單字管理類別
|   |-- VocaQuestApp.h/.cpp     # 主應用程式類別
|   |-- Makefile                # 編譯設定檔
|-- report/                     # 報告目錄
|   |-- report.tex              # LaTeX 報告
|   |-- Makefile               # 報告編譯設定
|-- vocData.txt                # 單字資料檔
|-- README.md                  # 專案說明文件
```

A.2 編譯設定檔

```
CXX = /opt/homebrew/bin/g++-14
CXXFLAGS = -std=c++14 -Wall -Wextra -isysroot /Applications/Xcode.app/
    Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX.sdk
TARGET = vocaquest
SOURCES = main.cpp AlertMsg.cpp Log.cpp Quiz.cpp Vocabulary.cpp
    VocaQuestApp.cpp

$(TARGET): $(SOURCES)
    $(CXX) $(CXXFLAGS) -o $(TARGET) $(SOURCES)

clean:
    rm -f $(TARGET)

.PHONY: clean
```

Listing 23: Makefile 內容

B 參考資料

1. Bjarne Stroustrup. *The C++ Programming Language*. 4th Edition. Addison-Wesley, 2013.
2. Scott Meyers. *Effective C++*. 3rd Edition. Addison-Wesley, 2005.
3. Herb Sutter and Andrei Alexandrescu. *C++ Coding Standards*. Addison-Wesley, 2004.
4. ISO/IEC 14882:2011. *Information technology —Programming languages —C++*.
5. C++ Reference Documentation. `https://cppreference.com/`