

# 杨欣渝-2022141650160-assignment5

## 1 unit test

```
root@730aa3418e90:/ws/assignment5/build# ./main
RUNNING TESTS ...
[-----] Running 4 tests from 1 test suite.
[-----] Global test environment set-up.
[-----] 4 tests from assignment5Test
[ RUN ] assignment5Test.ReadData
[ OK ] assignment5Test.ReadData (5 ms)
[ RUN ] assignment5Test.authors
[ OK ] assignment5Test.authors (7 ms)
[ RUN ] assignment5Test.publisher
[ OK ] assignment5Test.publisher (7 ms)
[ RUN ] assignment5Test.printTable
Title Author(s) Genre Price Publisher
Data Scientists at Work Sebastian Gutierrez data_science $23.00 Apress
Slaughterhouse Five Vonnegut, Kurt fiction $19.80 Random House
Birth of a Theorem Villani, Cedric mathematics $23.40 Bodley Head
Structure & Interpretation of Computer Programs Sussman, Gerald computer_science $24.00 MIT Press
Age of Wrath, The Eraly, Abraham history $23.80 Penguin
[ OK ] assignment5Test.printTable (6 ms)
[-----] 4 tests from assignment5Test (27 ms total)

[-----] Global test environment tear-down
[=====] 4 tests from 1 test suite ran. (27 ms total)
[ PASSED ] 4 tests.
<<<SUCCESS>>>
```

## 2 核心代码解释

### 2.1 生成makefile

```
#CMakeLists.txt节选
#链接到GoogleTest
target_link_libraries(main PRIVATE assignment5_lib GTest::gtest GTest::gmock)
include(GoogleTest)
add_test(GoogleTest main)
gtest_discover_tests(main)
```

### 2.2 assignment5.h

```
#ifndef ASSIGNMENT_H
#define ASSIGNMENT_H

#include<bits/stdc++.h>
#include"author.h"
#include"book.h"
#include"publication.h"
typedef std::vector<std::vector<std::string>> Dataframe;

Dataframe read_csv(std::string filename);//从.csv中读取
Book add_book(Book x);//加入总书单
Author* get_author(std::string _name);//判断作者是否已经出现过了，有就直接返回Author指针，没有则先创建再返回
std::shared_ptr<Publisher> get_pub(std::string _name);//判断出版社是否已经出现过了，有就直接返回指针，没有则先创建再返回
std::vector<Book> defineBooks(Dataframe* Table);
```

```
void sortBooksByPrice(std::vector<Book*> list_of_books );//用书的价格排序，升序（注意排序的类对象的成员变量中不能有其他类的引用）
void showTable(Dataframe* table,int start, int stop);
#endif
```

## 2.3 author.h

```
#ifndef AUTHOR_H
#define AUTHOR_H

#include<bits/stdc++.h>
class Book;
class Author{
public:
    Author(std::string _name);//初始化
    void setListOfBooks(std::vector<Book*> all_books);//对作者的书单进行初始化
    bool operator ==(const Author& y);//这里重载==是用于判断是否出现过该作者
    void add_one_book(Book* x);//往这个作者的书单里加一本书
    std::string getAuthorName();//返回作者名字

private:
    friend Author* get_author(std::string _name);//判断作者是否出现过的函数，需要访问private

    std::string name;//作者名字
    std::vector<Book*> list_of_books;//作者的书单
};

#endif
```

## 2.4 book.h

```
#ifndef BOOK_H
#define BOOK_H

#include<bits/stdc++.h>
#include "author.h"
#include "publication.h"
typedef std::vector<std::vector<std::string>> Dataframe;
class Book{
public:
    Book();
    Book(std::string _title,
        std::string _genre,
        double _price,
        Author* _author,
        std::shared_ptr<Publisher> _publisher);//书的初始化
    std::string getAuthorName();
    std::string getPublisherName();
    bool operator ==(Book& y);
```

```

    bool operator<(Book& y); //排序sort的重载<

private:
    friend Book add_book(Book* x);
    friend void sortBooksByPrice(std::vector<Book> & list_of_books );

    std::string title;
    std::string genre;
    double price;

    Author* author;
    std::shared_ptr<Publisher> publisher;
};

#endif

```

## 2.5 publisher.h

```

#ifndef PUBLICATION_H
#define PUBLICATION_H

#include<bits/stdc++.h>
class Book;
class Publisher{
public:
    Publisher(std::string _name);
    void setListOfBooks(std::vector<Book*> all_books);
    bool operator ==(const Publisher& y);
    void add_one_book(Book* x);
    std::string getPublisherName();

private:
    friend std::shared_ptr<Publisher> get_pub(std::string _name); //判断出版社是否出
    现过的函数，需要访问private

    std::string name;
    std::vector<Book*> list_of_books;
};

#endif

```