Student Name: 戈天齐
Student ID: 2022141460202

C++
Assignment

头文件

author.h

```cpp
#ifndef HEADER_AUTHOR_H
#define HEADER_AUTHOR_H

#include <iostream>
#include <vector>

class Book;
class Author {
public:
    Author(std::string _name);//此为一个构造函数，并且借此可以实现访问private变量
    void setListOfBooks(std::vector<Book *> all_books);

    std::string getAuthorName();

private:
    std::string name;
    std::vector<Book *> list_f_Books;

};
#endif // HEADER_AUTHOR_H
```

pulication.h

```cpp
#ifndef HEADER_PUBLICATION_H
#define HEADER_PUBLICATTION_H
#include <iostream>
#include <vector>
class Book;
class Publisher{
public:
    Publisher(std::string _name);
    void setListOfBooks(std::vector<Book*> all_books);
```

```
    std::string getPublisherName();
private:
    std::string name;
    std::vector<Book*> list_of_books;
};
#endif // HEADER_PUBLICATION_H
```

book.h

```
#ifndef HEADER_BOOk_H
#define HEADER_BOOK_H

#include <iostream>
#include <memory>
#include "author.h"
#include "publication.h"

class Book {
public:
    Book(std::string _title,
         std::string _genre,
         std::string _price,
         Author& _author,
         std::string _publisher);
    std::string getAuthorName();
    std::string getPublisherName();

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

    Author& author;
    std::shared_ptr<Publisher> publisher;
};
#endif
```

这三个头文件涉及了互相引用的问题，如果直接互相引用头文件会在构建时无限循环。这里的处理方式采用的是使用前向声明，在 author.h 和 pubication.h 里面声明 class Book 在 book.h 里面引用前两个头文件

0.1 assignment5.h

```
#include "Book.h"
#include <vector>
```

```
#include <string>
#include <fstream>
#include <regex>
#include <sstream>
#include <iomanip>
typedef std::vector<std::vector<std::string>> Dataframe;
Dataframe read_csv(const std::string& filename);
void sortBooksPrice(std::vector<Book> &books);
void showTable(Dataframe* table,int start, int stop);
std::vector<Book> defineBooks(Dataframe* Table);
```

源文件

author.cpp

```
#include "../include/author.h"

Author::Author(std::string _name)
{
    name = _name;
}
void Author::setListOfBooks(std::vector<Book*> all_books)
{
    list_f_Books = all_books;
}
std::string Author::getAuthorName() {
    return name;
}
```

publication.cpp

```
#include "../include/publication.h"
Publisher::Publisher(std::string _name)
{
    name = _name;
}
void Publisher::setListOfBooks(std::vector<Book*> all_books)
{
    list_of_books = all_books;
}
std::string Publisher::getPublisherName() {
    return name;
}
```

book.cpp

Constructor for 'Book' must explicitly initialize the reference member 'author'

在这里提示我们时不能创造一个空的 Author&，所以不能按寻常的方式完成，这里采用的是初始化列表，在构建的时候初始化

```
#include "../include/book.h"
Book::Book(std::string _title,std::string _genre,std::string _price,Author & _author,std::string
    _publisher):author(_author)//这两个author有什么区别
{
    title = _title;
    genre = _genre;
    price = std::stod(&_price[1]);
    publisher = std::make_shared<Publisher>(_publisher);
}
std::string Book::getAuthorName() {
    return author.getAuthorName();
}
std::string Book::getPublisherName() {
    return publisher->getPublisherName();

}
```

assignment5.cpp

这里存在一个问题，book 对象里面的元素 Author& author，这里采用了一个引用，而引用的对象如果只在 defineBooks 里面声明定义，那么它的生命周期就与该函数有关，该函数结束的时候，该对象就被销毁，而 author 的引用被销毁了，编译器就会报错

```
#include "../include/assignment5.h"

Dataframe read_csv(const std::string& filename){
    std::ifstream file(filename);
    if(!file.is_open()){
        std::cout << "File not found" << std::endl;
        exit(1);
    }
    std::string line;
    std::getline(file,line);
    std::vector<std::vector<std::string>> data;
    while(std::getline(file,line)){
        std::stringstream ss(line);
        std::string cell;
        std::vector<std::string> row;
        while(std::getline(ss,cell,',')){
            row.push_back(cell);
```

```
        }
        data.push_back(row);
    }
    for(auto & i : data){
        int k = 0;
        for(int j = 0; j < i.size(); j++){
            if(i[j][0] == '"'){
                i[j].erase(0,1);
                k = j;
            }
            if(i[j][i[j].size()-1] == '"'){
                i[j].erase(i[j].size()-1,1);
                if(k != j){
                    i[k] += "," + i[j];
                    i.erase(i.begin()+j);
                    j--;
                }
            }
        }
    }
    return data;
}

std::vector<Author*> storage;

std::vector<Book> defineBooks(Dataframe* Table){
    std::vector<Book> books;
    for( const auto & i : *Table){
        storage.emplace_back(new Author(i[1]));
        books.emplace_back(i[0],i[2],i[3],*storage.back(),i[4]);
    }
    return books;
}

void showTable(Dataframe* table,int start, int stop){
    for(int i = start; i < stop; i++){
        for(const auto & j : (*table)[i]){
            std::cout << std::left << std::setw(50)<< j;
        }
        std::cout << std::endl;
    }
}
```

在这里需要根据具体情况修改 dataset 的路径

unittest.cpp

```cpp
#include <climits>
#include "../include/assignment5.h"
#include "gtest/gtest.h" //google test
namespace
{


    TEST(assignment5Test,ReadData)
    {
        auto table = read_csv("../dataset.csv");
        EXPECT_EQ(211, table.size());
        EXPECT_EQ("Data Smart",table.at(1).at(0));
        EXPECT_EQ("Goswami, Jaideva", table[0][1]);
    }

    TEST(assignment5Test,authors)
    {
        auto table = read_csv("../dataset.csv");
        auto books = defineBooks(&table);

        EXPECT_EQ("Said, Edward",books[4].getAuthorName());
        EXPECT_EQ("Vonnegut, Kurt",books[11].getAuthorName());
    }

    TEST(assignment5Test,publisher)
    {
        auto table = read_csv("../dataset.csv");
        auto books = defineBooks(&table);

        EXPECT_EQ("Dell",books[210].getPublisherName());
        EXPECT_EQ("Penguin",books[177].getPublisherName());
    }

    TEST(assignment5Test,printTable)
    {
        auto table = read_csv("../dataset.csv");
        showTable(&table,10,14);
    }



}
```

main.cpp

```cpp
#include <iostream>
#include "../include/assignment5.h"
#include "gtest/gtest.h"

int main(int argc, char **argv)
{
    ::testing::InitGoogleTest(&argc, argv);
    std::cout << "RUNNING TESTS ..." << std::endl;
    int ret{RUN_ALL_TESTS()};
    if (!ret)
        std::cout << "<<<SUCCESS>>>" << std::endl;
    else
        std::cout << "FAILED" << std::endl;
    return 0;
}
```