



C++ Assignment

头文件

author.h

```
1  #ifndef HEADER_AUTHOR_H
2  #define HEADER_AUTHOR_H
3
4  #include <iostream>
5  #include <vector>
6
7  class Book;
8  class Author {
9  public:
10     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
11     void setListOfBooks(std::vector<Book *> all_books);
12
13     std::string getAuthorName();
14
15 private:
16     std::string name;
17     std::vector<Book *> list_f_Books;
18
19 };
20 #endif // HEADER_AUTHOR_H
```

pulication.h

```
1  #ifndef HEADER_PUBLICATION_H
2  #define HEADER_PUBLICATION_H
3  #include <iostream>
4  #include <vector>
5  class Book;
6  class Publisher{
7  public:
8     Publisher(std::string _name);
9     void setListOfBooks(std::vector<Book*> all_books);
```

```

10     std::string getPublisherName();
11 private:
12     std::string name;
13     std::vector<Book*> list_of_books;
14 };
15 #endif // HEADER_PUBLICATION_H

```

book.h

```

1  #ifndef HEADER_BOOK_H
2  #define HEADER_BOOK_H
3
4  #include <iostream>
5  #include <memory>
6  #include "author.h"
7  #include "publication.h"
8
9  class Book {
10 public:
11     Book(std::string _title,
12          std::string _genre,
13          std::string _price,
14          Author& _author,
15          std::string _publisher);
16     std::string getAuthorName();
17     std::string getPublisherName();
18
19 private:
20     std::string title;
21     std::string genre;
22     double price;
23
24     Author& author;
25     std::shared_ptr<Publisher> publisher;
26 };
27 #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 `publication.h` 里面声明 `class Book` 在 `book.h` 里面引用前两个头文件

0.1 assignment5.h

```

1  #include "Book.h"
2  #include <vector>

```

```

3  #include <string>
4  #include <fstream>
5  #include <regex>
6  #include <sstream>
7  #include <iomanip>
8  typedef std::vector<std::vector<std::string>> Dataframe;
9  Dataframe read_csv(const std::string& filename);
10 void sortBooksPrice(std::vector<Book> &books);
11 void showTable(Dataframe* table,int start, int stop);
12 std::vector<Book> defineBooks(Dataframe* Table);

```

源文件

author.cpp

```

1  #include "../include/author.h"
2
3  Author::Author(std::string _name)
4  {
5      name = _name;
6  }
7  void Author::setListOfBooks(std::vector<Book*> all_books)
8  {
9      list_f_Books = all_books;
10 }
11 std::string Author::getAuthorName() {
12     return name;
13 }

```

publication.cpp

```

1  #include "../include/publication.h"
2  Publisher::Publisher(std::string _name)
3  {
4      name = _name;
5  }
6  void Publisher::setListOfBooks(std::vector<Book*> all_books)
7  {
8      list_of_books = all_books;
9  }
10 std::string Publisher::getPublisherName() {
11     return name;
12 }

```

book.cpp

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

在这里提示我们时不能创建一个空的 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

```
1 #include "../include/book.h"
2 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
3 {
4     title = _title;
5     genre = _genre;
6     price = std::stod(&_price[1]);
7     publisher = std::make_shared<Publisher>(_publisher);
8 }
9 std::string Book::getAuthorName() {
10     return author.getAuthorName();
11 }
12 std::string Book::getPublisherName() {
13     return publisher->getPublisherName();
14 }
15 }
```

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

```
1 #include "../include/assignment5.h"
2
3 Dataframe read_csv(const std::string& filename){
4     std::ifstream file(filename);
5     if(!file.is_open()){
6         std::cout << "File not found" << std::endl;
7         exit(1);
8     }
9     std::string line;
10    std::getline(file, line);
11    std::vector<std::vector<std::string>> data;
12    while(std::getline(file, line)){
13        std::stringstream ss(line);
14        std::string cell;
15        std::vector<std::string> row;
16        while(std::getline(ss, cell, ',')){
17            row.push_back(cell);
```

```

18     }
19     data.push_back(row);
20 }
21 for(auto & i : data){
22     int k = 0;
23     for(int j = 0; j < i.size(); j++){
24         if(i[j][0] == ''){
25             i[j].erase(0,1);
26             k = j;
27         }
28         if(i[j][i[j].size()-1] == ''){
29             i[j].erase(i[j].size()-1,1);
30             if(k != j){
31                 i[k] += "," + i[j];
32                 i.erase(i.begin()+j);
33                 j--;
34             }
35         }
36     }
37 }
38 return data;
39 }
40
41 std::vector<Author*> storage;
42
43 std::vector<Book> defineBooks(Dataframe* Table){
44     std::vector<Book> books;
45     for( const auto & i : *Table){
46         storage.emplace_back(new Author(i[1]));
47         books.emplace_back(i[0],i[2],i[3],*storage.back(),i[4]);
48     }
49     return books;
50 }
51
52 void showTable(Dataframe* table,int start, int stop){
53     for(int i = start; i < stop; i++){
54         for(const auto & j : (*table)[i]){
55             std::cout << std::left << std::setw(50)<< j;
56         }
57         std::cout << std::endl;
58     }
59 }

```

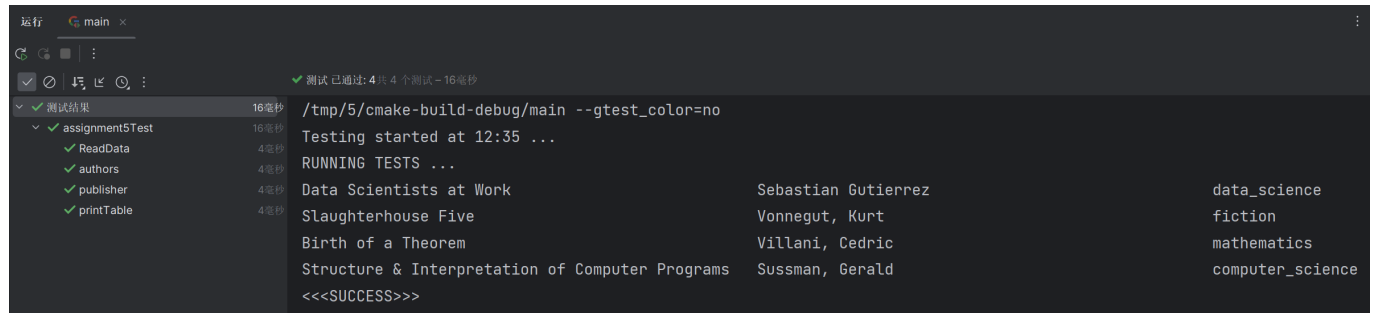
在这里需要根据具体情况修改 dataset 的路径

unittest.cpp

```
1  #include <climits>
2  #include "../include/assignment5.h"
3  #include "gtest/gtest.h" //google test
4  namespace
5  {
6
7
8      TEST(assignment5Test,ReadData)
9      {
10         auto table = read_csv("../dataset.csv");
11         EXPECT_EQ(211, table.size());
12         EXPECT_EQ("Data Smart",table.at(1).at(0));
13         EXPECT_EQ("Goswami, Jaideva", table[0][1]);
14     }
15
16     TEST(assignment5Test,authors)
17     {
18         auto table = read_csv("../dataset.csv");
19         auto books = defineBooks(&table);
20
21         EXPECT_EQ("Said, Edward",books[4].getAuthorName());
22         EXPECT_EQ("Vonnegut, Kurt",books[11].getAuthorName());
23     }
24
25     TEST(assignment5Test,publisher)
26     {
27         auto table = read_csv("../dataset.csv");
28         auto books = defineBooks(&table);
29
30         EXPECT_EQ("Dell",books[210].getPublisherName());
31         EXPECT_EQ("Penguin",books[177].getPublisherName());
32     }
33
34     TEST(assignment5Test,printTable)
35     {
36         auto table = read_csv("../dataset.csv");
37         showTable(&table,10,14);
38     }
39
40
41
42 }
```

main.cpp

```
1 #include <iostream>
2 #include "../include/assignment5.h"
3 #include "gtest/gtest.h"
4
5 int main(int argc, char **argv)
6 {
7     ::testing::InitGoogleTest(&argc, argv);
8     std::cout << "RUNNING TESTS ..." << std::endl;
9     int ret{RUN_ALL_TESTS()};
10    if (!ret)
11        std::cout << "<<<SUCCESS>>>" << std::endl;
12    else
13        std::cout << "FAILED" << std::endl;
14    return 0;
15 }
```



The screenshot shows a C++ IDE with a terminal window displaying the output of a test run. The test results are as follows:

| Test Case | Duration |
|-----------------|----------|
| assignment5Test | 16毫秒 |
| ReadData | 4毫秒 |
| authors | 4毫秒 |
| publisher | 4毫秒 |
| printTable | 4毫秒 |

The terminal output shows the following sequence of events:

```
/tmp/5/cmake-build-debug/main --gtest_color=no
Testing started at 12:35 ...
RUNNING TESTS ...
Data Scientists at Work          Sebastian Gutierrez          data_science
Slaughterhouse Five             Vonnegut, Kurt             fiction
Birth of a Theorem              Villani, Cedric             mathematics
Structure & Interpretation of Computer Programs Sussman, Gerald            computer_science
<<<SUCCESS>>>
```