编号： 第 1 章第 1 次

山东师范大学
SHANDONG NORMAL UNIVERSITY

信息科学与工程学院实验报告

《面向对象程序设计》

Object-Oriented Programming

姓名：	朱会琛
学号：	202311000202
班级：	计工本 2302
教师：	张庆科
时间：	2024 年 09 月 20 日

《面向对象程序设计》实验报告

报告要求：实验报告包含实验目的、实验内容、实验过程（详细操作流程）、实验结果（程序运行结果高清截图）、实验分析总结五个部分。报告中若涉及代码程序，请在附录部分提供完整程序源码及源码托管地址(基于 Highlight 软件导入源码)。报告撰写完毕后请将 PDF 格式版本上传到坚果云作业提交系统。

一、实验目的

- 理解面向过程和面向对象编程的异同点
- 理解基于 C/C++的多文件编程基本原理
- 能够根据任务需求独立构建多文件程序项目
- 掌握 Visual Studio C++的项目构建及运行过程

二、实验内容

任务 1：简答论述

1. 论述面向过程编程和面向对象编程的差异性。

答：1.面向对象和面向过程不是对立的。面向对象是面向过程发展到一定阶段的产物，是程序设计的高级阶段。
面向对象以面向过程为基础，通过引入对象的概念，使得程序设计更加符合人类的思维方式，提高了软件的可维护性和可扩展性。

2.面向过程适合于一个人的小量工作，而面向对象更侧重于团队合作，需要很多人完成的大量工作。

3.面向过程 ： 按步骤进行执行。考虑怎么来完成某一需求，分析出具体的步骤，然后按照步骤来一步步实现。

面向对象 ： 按对象的功能进行调用。适合复杂的工作需要团队合作

2. 简述什么是多文件编程，如何实现多文件编程。

答：• 1.1 什么是多文件编程 多文件编程是指将一个大型程序拆分成多个独立的文件来编写和管理代码。 每个文件通常包含特定功能或模块的实现，通过模块间的调用和依赖关系来构建完整的程序。 这种方式可以有效地减少单个文件的复杂度，提高代码的可读性和维护性。

• 1.2 多文件编程的优势和应用场景 多文件编程的优势包括：模块化管理：将代码分解成多个模块，便于分工合作和统一管理。 代码重用：可以将通用的功能封装成独立的模块，在不同的项目中重复利

用。 易于扩展：当需求发生变化时，可以更方便地新增、修改或删除特定模块。

- 1.3 多文件编程与单文件编程的对比 相比单文件编程，多文件编程具有以下对比优势： 结构清晰：代码分布在多个文件中，结构更加清晰易懂。 可维护性更高：模块化设计使得代码修改和维护更为简便。

3. **简述条件编译方法在多文件编程中的作用,举例分析。**

答：条件指示符最主要的目的是防止头文件的重复包含和编译。

当第一次包含 test.h 时，由于没有定义_TEST_H，条件为真，这样就会包含（执行）#ifndef _TEST_H 和#endif 之间的代码，当第二次包含 test.h 时前面一次已经定义了_TEST_H，条件为假，#ifndef _TEST_H 和#endif 之间的代码也就不会再次被包含，这样就避免了重定义了。

任务 2： 程序设计

基于 Visual Studio 集成开发环境，使用 C++编写一个"大象装进冰箱"的程序，要求如下：

1. 采用多文件编程方式编写，代码功能自行拟定，代码框架可参考课件 PPT 简易程序。
2. 结合该程序论述多文件编程设计过程，以及多文件编程采用的主要技术实现方法。

三、实验过程

1. 搭建环境
2. 创建各对象
3. 声明和定义各对象
4. Main 函数中调用
5. 运行程序

四、实验结果

Main.cpp

```cpp
#include <iostream>
#include "OPEN.h"
#include "PUSH.h"
#include "CLOSE.h"
#include "Swap.h"

using namespace std;

int elephant = 01, fridge = 02;
int main() {

    /*1.open the fridge */
    Touch(fridge);
    Push(fridge);

    /*2.push an elephant into the fridge*/
    Push(elephant);
    Faseten(elephant);

    /*3.close the fridge*/
    Touch(fridge);
    Close(fridge);

    /*int value1 = 10, value2 = 20;
    cout << "value1 :" << value1 << endl;
    cout << "value2 :" << value2 << endl;

    Swap(value1, value2);
    cout << "value1 :" << value1 << endl;
    cout << "value2 :" << value2 << endl;*/
```

Close.cpp

```cpp
#include "CLOSE.h"
#include <iostream>

using namespace std;

void Close(int f) {
    cout << "fasten e" << endl;
}
```

Close.h

```cpp
#ifndef _CLOSE_H_
#define _CLOSE_H_

void Close(int f);

#endif // !_CLOSE_H_
#pragma once
```

Push.h

```
#ifndef _PUSH_H_
#define _PUSH_H_

void Push(int e);
void Faseten(int e);

#endif // !_PUSH_H_
#pragma once
```

Push. cpp

```
#include <iostream>
#include "PUSH.h"

using namespace std;

void Push(int e) {
    cout << "push e" << endl;
}

void Faseten(int e) {
    cout << "fasten e" << endl;
};
```

Open. cpp

```
#include "OPEN.h"
#include <iostream>

 using namespace std;

void Touch(int f) {
    cout << "touch f" <<endl;
}

void Pull(int f) {
    cout << "pull f" << endl;
};
```

Open. h

```
#ifndef _OPEN_H_
#define _OPEN_H_

void Touch(int f);
void Pull(int f);

#endif // !_OPEN_H_
#pragma once
```

运行结果：

五、实验总结

对于此次实验，我的收获颇丰。

1. 相较于之前学过的 pop 面向过程的语言，oop 面向对象语言写起来更简便，清楚的分为各个对象，将一个总的任务分派到各对象，再由各对象执行自己的任务，整个流程简单易懂，并无太多逻辑性的过程。
2. 相较于之前的单文件编程，多文件编程很好的将函数封装起来，需要时很容易就能调用，当程序出现 bug 时，很容易就能找到错误的源文件，进行后续的修改，极大的减少了寻找错误的时间。

附录：实验源代码（基于 Highlight 软件粘贴带有行号的源码）