

计算机学院 高级程序语言设计 课程实验报告

实验题目：多文件项目，预编译	学号：202300130150	
日期：2024. 3. 28	班级： 4	姓名： 王成意

实验目的：

- 掌握多文件项目构建
- 熟悉编译预处理指令

实验步骤与内容：

-----方式一、新建一个项目，将所有文件加入-----

(1) 建立一个项目文件（名称自定）。在项目中添加例 5-10 中的三个文件。

- 新建一个项目文件：“新建” / “项目”，选“Empty project”，输入项目名，“确定”。建立一个 .dev 文件。
- 在左侧项目管理窗口，右击项目名称，选“添加”，添加如下三个文件：

Point.h
Point.cpp
5_10.cpp

项目中默认添加的未命名 n 文件可删除。

The screenshot shows the Microsoft Visual Studio interface. On the left, the Solution Explorer displays a project named 'Project1' containing files: Point.h, Point.cpp, and 5_10.cpp. The 5_10.cpp file is selected. On the right, the code editor shows the following C++ code:

```
1 //include "Point.h"
2 #include <iostream>
3 using namespace std;
4
5 int main() {
6     Point a(4, 5); //定义对象a, 其构造函数调用cout
7     cout << "Point A: " << a.getX() << ", " << a.getY()
8     Point::showCount(); //输出对象个数
9
10    Point b(a); //定义对象b, 其构造函数调用cout
11    cout << "Point B: " << b.getX() << ", " << b.getY()
12    Point::showCount(); //输出对象个数
13
14
15
16 }
```

```
#include "Point.h"
#include <iostream>
using namespace std;

int Point::count = 0; //使用类名初始化静态数据成员

Point::Point(const Point& p) : x(p.x), y(p.y) {
    count++;
}

void Point::showCount() {
    cout << " Object count = " << count << endl;
}
```

```
#pragma once

class Point { //类的定义
public: //外部接口
    Point(int x = 0, int y = 0) : x(x), y(y) { count++ }
    Point(const Point& p);
    ~Point() { count--; }
    int getX() const { return x; }
    int getY() const { return y; }
    static void showCount(); //静态函数成员
private: //私有数据成员
    int x, y;
    static int count; //静态数据成员
};
```

(c) 编译、运行该项目。

```
Point A: 4, 5 Object count = 1
Point B: 4, 5 Object count = 2

D:\C++ programs\project\Project1\x64\Debug\Projec
按任意键关闭此窗口. . .|
```

-----方式二、将类实现单独编译，与使用者分开。-----

Project Manager View Class Debug Point.cpp Point.cpp

```

1 #include "Point.h"
2 #include <iostream>
3 using namespace std;
4 int Point::count = 0; // 使用类名初始化
5 Point::Point(const Point &p) : x(p.x), y(
6     count++);
7 }
8 void Point::showCount() {
9     cout << " Object count = " << count
10 }
11

```

名称	修改日期	类型	大小
5-10.cpp	2024/3/28 10:23	C++ Source File	1 KB
Makefile.win	2024/3/28 10:23	WIN 文件	2 KB
Point.cpp	2024/3/28 10:23	C++ Source File	1 KB
Point.h	2024/3/28 10:22	C Header 源文件	1 KB
Point.o	2024/3/28 10:22	O 文件	3 KB
项目2.a	2024/3/28 10:22	A 文件	3 KB
项目2.dev	2024/3/28 10:18	Dev-C++ Project File	1 KB
项目2.layout	2024/3/28 10:22	LAYOUT 文件	1 KB
项目4.dev	2024/3/28 10:24	Dev-C++ Project File	2 KB
项目4.layout	2024/3/28 10:24	LAYOUT 文件	1 KB
5-10.o	2024/3/28 10:23	O 文件	5 KB
项目4.exe	2024/3/28 10:23	应用程序	1,879 KB

编译出 point.o 文件如上图。

实现第二个项目：

Project Manager View Class Debug Point.cpp Point.h 5-10.cpp

```

1 class Point { // 类的定义
2 public: // 外部接口
3     Point(int x = 0, int y = 0) : x(x), y(
4         y);
5     Point(const Point &p);
6     ~Point() { count--; }
7     int getX() const { return x; }
8     int getY() const { return y; }
9     static void showCount(); // 静态函数
10 private: // 私有数据成员
11     int x, y;
12     static int count; // 静态数据成员
13 }

```



加入 point.o 文件
编译运行成功。

```
main() {
    Point a(4, 5); // 定义对象a，其构造函数调用cout
    cout << "Point A: " << a.getX() << ", " << a.get
    Point b(3, 7);
    cout << "Point B: " << b.getX() << ", " << b.get
    Point c(5, 9);
    cout << "Point C: " << c.getX() << ", " << c.get
    return 0;
}
```

1. 多文件项目环境下的运行结果截图。显示了三个点的输出：Point A: 4, 5 Object count = 1, Point B: 4, 5 Object count = 2, Point C: 5, 9 Object count = 3。程序正常退出，耗时0.09634秒。

2. 在上述多文件项目环境中完成下列实验（多重定义）
 - (1) 在 Point.h 文件中加入以下函数的定义语句，尝试编译运行，对编译结果进行截图。

```
已启动生成...
1>----- 已启动生成: 项目: Project1, 配置: Debug x64 -----
1>5_10.cpp
1>Point.cpp
1>正在生成代码...
1>LINK : 没有找到 D:\C++ programs\project\Project1\x64\Debug\Project1.exe 或上一个增量链接没有生成它; 正在执行完全链接
1>Point.obj : error LNK2005: "int __cdecl test(void)" (?test@@YAHXZ) 已经在 5_10.obj 中定义
1>D:\C++ programs\project\Project1\x64\Debug\Project1.exe : fatal error LNK1169: 找到一个或多个多重定义的符号
1>已完成生成项目 "Project1.vcxproj" 的操作 - 失败。
===== 生成: 0 成功, 1 失败, 0 最新, 0 已跳过 ======
===== 生成 开始于 10:28, 并花费了 02.442 秒 ======
|
```

2) 将上述函数替换为以下函数, 再次尝试编译运行, 对结果进行截图。

结果: 编译成功。

```
输出
显示输出来源(S): 生成
已启动生成...
1>----- 已启动生成: 项目: Project1, 配置: Debug x64 -----
1>5_10.cpp
1>Point.cpp
1>正在生成代码...
1>Project1.vcxproj -> D:\C++ programs\project\Project1\x64\Debug\Project1.exe
===== 生成: 1 成功, 0 失败, 0 最新, 0 已跳过 ======
===== 生成 开始于 10:29, 并花费了 02.833 秒 ======
```

(3) 结合课上讲的 #include 的作用是将相应的文件内容嵌入到该文件中这一现实, 尝试解释上述两者的不同。可以尝试用 Point.h 里面的内容替换掉 #include "Point.h" 语句, 然后重新测试上述两个题目, 应该更能感受 inline 函数和普通函数的不同, 以及同一个文件被不同的文件 include 所产生的影响。

替换 point.cpp 后完成 (1), 编译错误:

解决方案资源管理器

搜索解决方案资源管理器

解决方案 'Project1' (1 个)

- Project1
 - 引用
 - 外部依赖项
 - 头文件
 - Point.h
 - 源文件
 - 5_10.cpp
 - Point.cpp
 - 资源文件

Project1 (全局范围) test()

5_10.cpp

```
1 #include <iostream>
2 using namespace std;
3 int test() {
4     return 315;
5 }
6 class Point { //类的定义
7 public: //外部接口
8     Point(int x = 0, int y = 0) : x(x), y(y)
9     Point(const Point& p);
10    ~Point() { count--; }
11    int getX() const { return x; }
12    int getY() const { return y; }
13    static void showCount(); //静态函数成员
14 private: //私有数据成员
15     int x, y;
16     static int count; //静态数据成员
17 };
18 int main() {
19     Point a(4, 5); //定义对象a, 其构造函数回使count++
20     cout << "Point A: " << a.getX() << ", ";
21     Point::showCount(); //输出对象个数
22
23     Point b(a); //定义对象b, 其构造函数回使count++
24     cout << "Point B: " << b.getX() << ". "
}
```

输出

显示输出来源(S): 生成

已启动生成...

1>----- 已启动生成: 项目: Project1, 配置: Debug x64 -----
1>Point.obj : error LNK2005: "int __cdecl test(void)" (?test@@YAHXZ) 已经在 5_10.obj 中定义
1>D:\C++ programs\project\Project1\x64\Debug\Project1.exe : fatal error LNK1169: 找到一个或多个多重定义的符号
1>已完成生成项目 "Project1.vcxproj" 的操作 - 失败。
===== 生成: 0 成功, 1 失败, 0 最新, 0 已跳过 ======

===== 生成: 开始于 10:34, 并花费了 00.738 秒 ======

```
#include <iostream>
using namespace std;
int test() {
    return 315;
}
class Point { //类的定义
public: //外部接口
    Point(int x = 0, int y = 0) : x(x), y(y) {
        count++;
    }
    Point(const Point& p);
    ~Point() { count--; }
    int getX() const { return x; }
    int getY() const { return y; }
    static void showCount(); //静态函数
private: //私有数据成员
    int x, y;
    static int count; //静态数据成员
};
int Point::count = 0; //使用类名初始化静态成员变量

Point::Point(const Point& p) : x(p.x), y(p.y) {
    count++;
}

void Point::showCount() {
    cout << " Object count = " << count << endl;
}
```

执行 (2), 成功编译:

```
#include <iostream>
using namespace std;
inline int test() {
    return 315;
}
class Point { //类的定义
public: //外部接口
    Point(int x = 0, int y) : ~Point() { count--; }
    int getX() const { return x; }
    int getY() const { return y; }
    static void showCount();
private:
    int x, y;
    static int count; //私有数据成员
};
int main() {
    Point a(4, 5); //定义对象a
    cout << "Point A: " << a.getX() << endl;
    Point::showCount(); //调用类的静态成员函数
    cout << "Point B: " << a.getY() << endl;
    Point b(a); //定义对象b
    cout << "Point B: " << b.getY() << endl;
    return 0;
}
```

```
#include <iostream>
using namespace std;
inline int test() {
    return 315;
}
class Point { //类的定义
public: //外部接口
    Point(int x = 0, int y = 0) : ~Point() { count--; }
    int getX() const { return x; }
    int getY() const { return y; }
    static void showCount();
private:
    int x, y;
    static int count; //私有数据成员
};
int main() {
    Point a(4, 5);
    cout << "Point A: " << a.getX() << endl;
    cout << "Point B: " << a.getY() << endl;
    Point b(a);
    cout << "Point B: " << b.getY() << endl;
    return 0;
}
```

(4) 如果在 Point.cpp 中加入以下函数的定义语句, 如何在 5_10.cpp 中调用该函数?
加 `extern` 关键字。

```

Point.cpp
1 #include <iostream>
2 using namespace std;
3 extern int test() {
4     return 315;
5 }
6 class Point { //类的定义
7     public: //外部接口
8         int x, y;
9         Point(int x, int y) : x(x), y(y) {}
10    void show() const { cout << "Point(" << x << ", " << y << ")" << endl; }
11    int getX() const { return x; }
12    int getY() const { return y; }
13    static int count; //静态数据成员
14 };
15 extern int test();
16 int main() {
17     Point a(4, 5); //定义对象a, 其构造函数
18     cout << "Point A: " << a.getX() << endl;
19     Point::showCount(); //输出对象个数
20
21     Point b(a); //定义对象b, 其构造函数
22     cout << "Point B: " << b.getX() << endl;
23     Point::showCount(); //输出对象个数
24     cout << test();
25
26 }

```

(1) 分别在 Point.cpp 和 5_10.cpp 中定义一个全局变量 int val = 315; 并在 main 函数中打印该值，然后进行编译，观察编译运行结果。

```

1>----- 已启动生成: 项目: Project1, 配置: Debug x64 -----
1>5_10.cpp
1>Point.cpp
1>正在生成代码...
1>LINK : 没有找到 D:\C++ programs\project\Project1\x64\Debug\Project1.exe 或上一个增量链接没有生成它; 正在执行完全链接
1>Point.obj : error LNK2005: "int val" (?val@@3HA) 已经在 5_10.obj 中定义
1>D:\C++ programs\project\Project1\x64\Debug\Project1.exe : fatal error LNK1169: 找到一个或多个多重定义的符号
1>已完成生成项目 “Project1.vcxproj”的操作 - 失败。
===== 生成: 0 成功, 1 失败, 0 最新, 0 已跳过 ======
===== 生成 开始于 10:43, 并花费了 03.037 秒 ======
|

```

(2) 在 Point.cpp 中定义全局变量 int val = 315;，之后在 5_10.cpp 中通过 extern int val; 定义该变量为外部变量。在 main 函数中打印该值，然后进行编译，观察编译并解释运行结果。

```

5_10.cpp  Point.h  Point.cpp
Project1  (全局范围)
1 #include <iostream>
2 #include "Point.h"
3 using namespace std;
4 extern int val;
5 int main() {
6     Point a(4, 5); //定义对
7     cout << "Point A: " <<
8     Point::showCount(); //输出
9 }

```

```

Point.h  Point.cpp
t1  (全局范围)
1 #include <iostream>
2 #include "Point.h"
3 using namespace std;
4 int Point::count = 0; //使用类名初始化
5
6 Point::Point(const Point& p) : x(p.x),
7                                 count++;
8
9     int val = 315;
10    void Point::showCount() {

```

结果：val 由 Point.cpp 的赋值 extern 挂载到了 5_10.cpp 中，输出 315.

(4) 将以下实验三中的抛骰子的代码通过多文件项目进行实现。该项目至少包含三个文件：dice.h 对 Dice 类及 GameStatus 进行了定义。Dice.cpp 对 Dice 的各个函数进行了实现，main.cpp 函数定义了 main 函数实现了对类的实例化和相应函数的调用。请将每个文件截图。运行后的结果也需要截图。

结果&& main.cpp：

```

解决方案资源管理器  main.cpp  dice.cpp  dice.h
解决方案 'Dice' (1 个项目)
 Dice
   |> 口 引用
   |> 外部依赖项
   |> 头文件
   |> dice.h
   |> 源文件
   |> dice.cpp
   |> main.cpp
   |> 资源文件
main.cpp  dice.cpp  dice.h
Dice  (全局范围)  main()
1 #include "dice.h"
2 #include <iostream>
3 using namespace std;
4 int main()
5 {
6     unsigned seed;
7     cout << "Please enter an unsigned integer: ";
8     cin >> seed; //输入随机数种子
9     Dice dice(seed);
10    dice.play();
11    return 0;
12 }

```

Dice.cpp：

解决方案资源管理器

解决方案 'Dice' (1 个项目)

- Dice
 - 引用
 - 外部依赖项
 - 头文件 dice.h
 - 源文件 dice.cpp
- main.cpp
- 资源文件

```
1 #include "dice.h"
2
3 #include<iostream>
4 using namespace std;
5 enum GameStatus { WIN, LOSE, PLAYING };
6
7 Dice::Dice(int seed) :seed(seed)
8 {
9     srand(seed);
10 }
11 int Dice::rollDice()
12 {
13     int die1 = 1 + rand() % 6;
14     int die2 = 1 + rand() % 6;
15     int sum = die1 + die2;
16     cout << "player rolled " << die1 << endl;
17     return sum;
18 }
19 void Dice::play() {
20     GameStatus status = PLAYING;
21     while (status == PLAYING) { //只要状态是PLAYING，就一直循环
22         int sum = rollDice(); //第一轮投骰子
23         switch (sum) {
24             case 7: //如果和数为7或11则为胜
25                 status = WIN;
26                 break;
27             case 3: //和数为2、3或12则为负
28                 status = LOSE;
29                 break;
30             case 11:
31                 status = WIN;
32                 break;
33             default: /*其它情况，游戏尚无结果，将状态设为PLAYING，记下点数，为下一轮做准备*/
34                 status = PLAYING;
35                 break;
36         }
37     }
38 }
```

Dice.h:

```
全局范围
1 #pragma once
2 class Dice {
3     int seed;
4 public:
5     Dice(int seed);
6     int rollDice();
7     void play();
8 };
9 
```

结论分析与体会：

非常好的实验，使我重新认识了多文件项目构建和 Vs 2022 的伟大。
同时我还学到了掌握多文件项目构建，熟悉编译预处理指令。