

计算机学院 高级程序语言设计 课程实验报告

实验题目：类与对象应用（二）	学号：202300130150	
日期：2024. 3. 21	班级： 4	姓名： 王成意

实验目的：

- 掌握类组合应用
- 了解 struct 在 C++中的扩展
- 了解联合体的本质与应用
- 了解位域
- 掌握类的综合应用

实验步骤与内容：

- 设计一个用于人事管理的 People 类。要求包括：构造、复制构造和析构函数、内联成员函数、带默认形参值的成员函数、类的组合。

```
#include<cstdio>

#include<cstring>

#include<string>

#include<iostream>

using namespace std;

class birthday{

private:

    int day, month, year;

public:

    birthday(int a=1, int b=1, int c=2001)

    {

        month = a;

        day = b;

        year = c;

    }

};
```

```
//月日年  
birthday(birthday &a);  
  
inline void show()  
{  
    printf("Birthday: %d.%d.%d\n", month, day,  
year);  
}  
};  
  
birthday::birthday(birthday  
&p):month(p.month),year(p.year),day(p.day){}  
  
  
class people  
{  
public:  
    people(birthday &x, char a, int b, string c);  
    people(people &a);  
    ~people();  
    inline void show()  
    {  
        printf("People:\n");  
        cout << "Name:" << name << endl;  
        cout << "Sex:" << sex << endl;  
    }  
};
```

```
    birth.show();

    cout << "Years:" << years << endl;

}

private:

    birthday birth;

    char sex;

    int years;

    string name;

};

people::~people(){

    printf("People is done.\n");

}

people::people(birthday &x,char a, int b, string
c):birth(x)

{

    this->name = c;

    this->years = b;

    this->sex = a;

}
```

```
people::people(people  
&x):birth(x.birth),sex(x.sex),years(x.years),name(x.name){}  
  
int main()  
{  
  
    birthday a;  
  
    birthday b(4, 7,2023);  
  
    b.show();  
  
    a.show();  
  
    people x1(a, 'M', 18, "钟积润");  
  
    x1.show();  
  
    return 0;  
  
}
```

运行结果：

```
birthday: 4.7.2023  
birthday: 1.1.2001  
People:  
Name:钟积润  
Sex:M  
Birthday: 1.1.2001  
Years:18  
People is done.
```

2. 给第 4 章 PPT 中的例 4-4 程序(如下所示)加入 Line 和 Point 类的析构函数(含输出信息)，分析程序运行结果。

原结果：

```
C:\Users\Huang\source\repos\LineTest> LineTest
Calling the copy constructor of Point
Calling constructor of Line
Calling the copy constructor of Point
Calling the copy constructor of Point
Calling the copy constructor of Line
The length of the line is: 5
The length of the line2 is: 5
PS D:\C++ programs>
```

改动后代码：

```
#include <iostream>

#include <cmath>

using namespace std;

class Point { //Point 类定义

public:

    Point(int xx = 0, int yy = 0) {

        x = xx;

        y = yy;

    }

    Point(Point &p);

    ~Point();

    int getX() { return x; }

    int getY() { return y; }

private:

    int x, y;

};
```

```
Point::Point(Point &p) {      //复制构造函数的实现

    x = p.x;

    y = p.y;

    cout << "Calling the copy constructor of Point" <<

endl;

}

Point::~Point()

{

    printf("This point is done.\n");

}

//类的组合

class Line {      //Line 类的定义

public: //外部接口

    Line(Point xp1, Point xp2);

    Line(Line &l);

    ~Line();

    double getLen() { return len; }

private: //私有数据成员

    Point p1, p2; //Point 类的对象 p1,p2

    double len;

};

//组合类的构造函数实现
```

```
Line::Line(Point xp1, Point xp2) : p1(xp1), p2(xp2){  
    p1 = xp1;  
    p2 = xp2;  
    cout << "Calling constructor of Line" << endl;  
    double x = static_cast<double>(p1.getX() -  
        p2.getX());  
    double y = static_cast<double>(p1.getY() -  
        p2.getY());  
    len = sqrt(x * x + y * y);  
}  
  
Line::~Line()  
{  
    printf("This line is done.\n");  
}  
  
Line::Line (Line &l): p1(l.p1), p2(l.p2) { //组合类的复  
制构造函数  
    cout << "Calling the copy constructor of Line" <<  
    endl;  
    len = l.len;  
}  
  
//主函数
```

```
int main() {  
  
    Point myp1(1, 1), myp2(4, 5); //建立 Point 类的对象  
  
    Line line(myp1, myp2); //建立 Line 类的对象  
  
    Line line2(line); //利用复制构造函数建立一个新对象  
  
    cout << "The length of the line is: ";  
  
    cout << line.getLen() << endl;  
  
    cout << "The length of the line2 is: ";  
  
    cout << line2.getLen() << endl;  
  
    return 0;  
}
```

改动后结果：

```
Calling the copy constructor of Point  
Calling constructor of Line  
This point is done.  
This point is done.  
Calling the copy constructor of Point  
Calling the copy constructor of Point  
Calling the copy constructor of Line  
The length of the line is: 5  
The length of the line2 is: 5  
This line is done.  
This point is done.  
This point is done.  
This line is done.  
This point is done.
```

观察发现，构造的对象在其作用域终点时调用其对应类的析构函数。
过程结果如图：

```
4 class Point { //Point类定义
5     public:
6         Point();
7         ~Point();
8         int getX() { return x; }
9         int getY() { return y; }
10    private:
11        int x, y;
12    };
13
14 Point::Point(Point &p) { //复制构造函数
15     x = p.x;
16     y = p.y;
17     cout << "Calling the copy constructor\n";
18 }
19
20 Point::~Point()
21 {
22     printf("This point is done.\n");
23 }
24
25 //类的组合
26
27 class Line { //Line类的定义
28     public: //外部接口
29     Line(Point p1, Point p2);
30 }
```

3. 将以下代码放入 test.c 文件内，是否能够正常编译？在 Student 中增加 private 成员，还能否正常编译？
第一问：可以。

```
1 #include <stdio.h>
2
3 struct Student { //学生信息结构体
4     //private:
5     int num;           //学号
6     const char *name; //姓名，字符串对象
7     char sex;          //性别
8     int age;           //年龄
9 };
10
11 int main() {
12     Student stu = { 97001, "Lin Lin", 'F', 19 };
13     printf("Num: %d\n", stu.num);
14     printf("Name: %s\n", stu.name);
15     printf("Sex: %c\n", stu.sex);
16     printf("Age: %d\n", stu.age);
17     return 0;
18 }
19
```

顶部 调试 控制台 终端 端口 + ~ 运行 cppdbg: 1.exe

```
D:\C++ programs> ^C
D:\C++ programs>
D:\C++ programs> & 'd:\VsCode-extentions\.vscode-insiders\extensi
scode.cppTools-1.19.8-win32-x64\debugAdapters\bin\WindowsDebugLaunc
'--stdin=Microsoft-MIEngine-In-qyab2111.qkt' '--stdout=Microsoft-M
Out-f0ugsoov.g2k' '--stderr=Microsoft-MIEngine-Error-x4kakcqn.f4n'
icrosoft-MIEngine-Pid-32iwb24f.wtw' '--dbgExe=D:\mingw64\bin\gdb.e
terpreter=mi'
m: 97001
m: Lin Lin
m: F
m: 19
```

第二问：

不能。

The screenshot shows a C++ development environment with the following details:

Code Editor (1.c tab):

```
1 #include <stdio.h>
2
3 struct Student { //学生信息结构体
4     private:
5         int num; //学号
6         const char *name; //姓名, 字符串对象, 将在
7         char sex; //性别
8         int age; //年龄
9     };
10
11 int main() {
12     Student stu = { 97001, "Lin Lin", 'F', 19 };
13     printf("Num: %d\n", stu.num);
14     printf("Name: %s\n", stu.name);
15     printf("Sex: %c\n", stu.sex);
16     printf("Age: %d\n", stu.age);
17     return 0;
18 }
19
```

Terminal Window:

```
D:\C++ programs\1.c:6:15: note: declared private here
    const char *name; //姓名, 字符串对象, 将在第6章详细介绍
                  ^~~~
D:\C++ programs\1.c:15:26: error: 'char Student::sex' is private within this context
    printf("Num: %c\n", stu.sex);
                  ^~~~
D:\C++ programs\1.c:7:8: note: declared private here
    char sex; //性别
              ^~~~
D:\C++ programs\1.c:16:26: error: 'int Student::age' is private within this context
```

Num, sex 等都是 student 的 private，无法外部访问。

对比 union 文件夹中的代码有什么不同？解释代码与运行结果的不同点。

4-8

```
D:\C++ programs\tmp\4_8.ex > +  
English: B  
Calculus: PASS  
C++ Programming: 85
```

4-8_1

```
English: B  
Calculus: PASS  
C++ Programming: 85
```

4-8_2

```
English: B  
Calculus: □  
C++ Programming: U
```

观察发现，代码 4-8_2 中出现问题：

```
course1: {...}  
> name: "English"  
mode: ExamInfo::GRADE  
m  
grade: 66 'B'  
pass: 66  
percent: 66  
t: 66 'B'  
course2: {...}  
> name: "Calculus"  
mode: ExamInfo::GRADE  
m  
grade: 1 '\001'  
pass: true  
percent: 1  
t: 1 '\001'  
course3: {...}  
> name: "C++ Programming"  
mode: ExamInfo::GRADE  
m  
grade: 85 'U'  
pass: 85  
percent: 85  
cout << m.grade;  
break;  
case PASS:  
cout << (m.pass ? "PASS" : "FAIL");  
break;  
case PERCENTAGE:  
cout << m.percent;  
break;  
}  
cout << endl;  
}  
int main() {  
ExamInfo course1("English", 'B');  
ExamInfo course2("Calculus", true);  
ExamInfo course3("C++ Programming", 85);  
course1.show();  
course2.show();  
course3.show();  
return 0;  
}
```

Mode 都是 “Grade”

寻找代码：

```
ExamInfo(string name, union M w)  
: name(name), mode(GRADE), m(w){ }
```

发现问题，改正后：

```
#include <string>
```

```
#include <iostream>

using namespace std;

// M
class ExamInfo;

enum mod
{
    GRADE,
    PASS,
    PERCENTAGE
};

union M {
    char grade;      // cijer
    bool pass;       // p?
    int percent;     // broj
    M(){}
    M(char c) {
        grade = c;
        t = GRADE;
    }
    M(bool b){ pass = b;
        t = PASS;
    }
};
```



```
union M m;

};

void ExamInfo::show() {

    cout << name << ": ";

    switch (mode) {

        case GRADE:

            cout << m.grade;

            break;

        case PASS:

            cout << (m.pass ? "PASS" : "FAIL");

            break;

        case PERCENTAGE:

            cout << m.percent;

            break;

    }

    cout << endl;
}

int main() {

    ExamInfo course1("English", 'B');

    ExamInfo course2("Calculus", true);
```

```
ExamInfo course3("C++ Programming", 85);

course1.show();

course2.show();

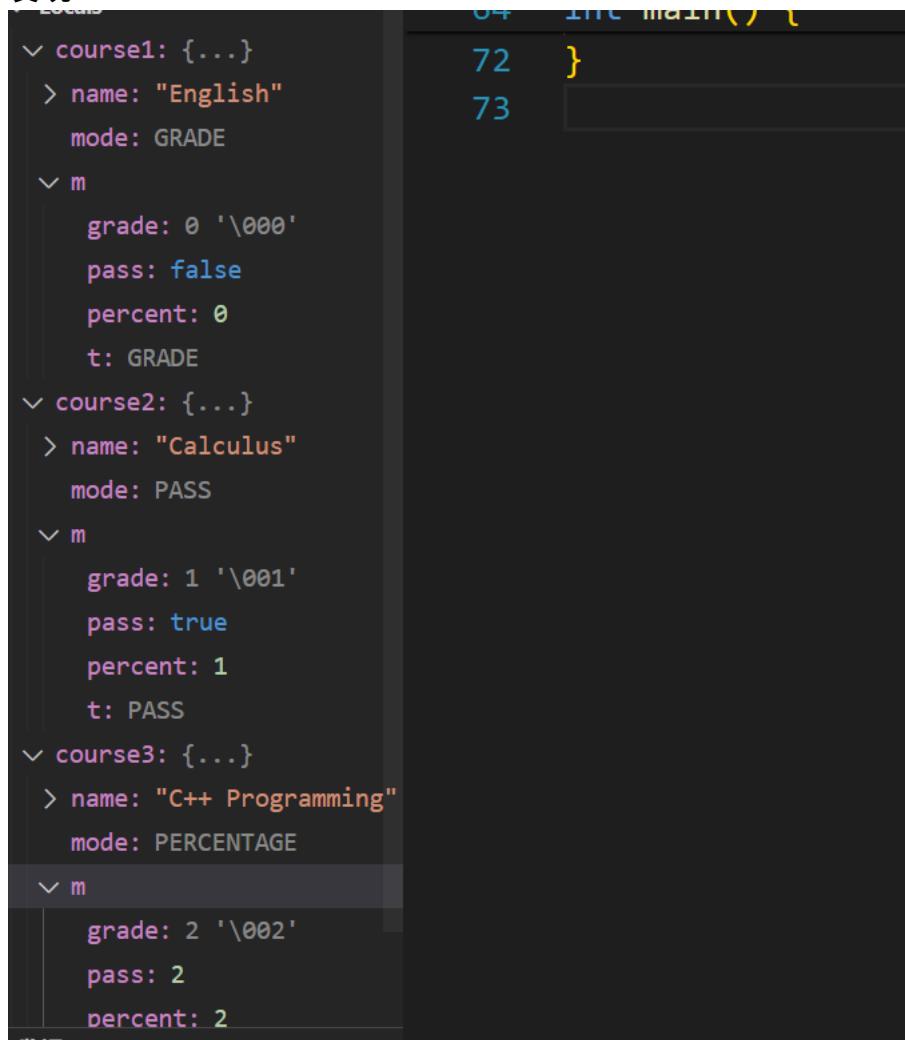
course3.show();

return 0;

}
```

(注释由于编码糊了)

发现：



这里 t 和 grade, pass, percent 共用内存，导致原来的值被覆盖，结果出错。

再观察 4-8_1 发现，其 union 开在 class 里，每次只用一个，不会出错。

```
private:  
    string name;      //姓名  
    enum {  
        GRADE,  
        PASS,  
        PERCENTAGE  
    } mode;           //模式  
  
    union {  
        char grade;    //成绩  
        bool pass;     //是否通过  
        int percent;   //百分比  
    } m;  
};
```

5 第4章习题4-19 拓展练习：位域。通过改变各个成员需要的bit数看看对于最后sizeof的值有什么影响，发现了什么规律。

源码：

```
#include<cstdio>  
  
#include<iostream>  
  
using namespace std;  
  
enum wordlen  
{  
    Bit32,  
    Bit64  
};  
  
enum core  
{  
    single,
```

```
dual,  
  
quad  
};  
  
enum HyperThreading  
{  
  
nosupport,  
  
support  
};  
  
class CPU{  
  
private:  
  
float freq;  
  
HyperThreading mode;  
  
core cores;  
  
wordlen bits;  
  
public:  
  
CPU(float x, HyperThreading y, core z, wordlen  
w);  
  
CPU(CPU &p);  
  
void show();  
};  
  
CPU::CPU(float x, HyperThreading y, core z, wordlen w)  
{
```

```
    this->freq = x;
    this->mode = y;
    this->cores = z;
    this->bits = w;
}

CPU::CPU(CPU
&p):freq(p.freq),mode(p.mode),cores(p.cores),bits(p.bi
ts){}

void CPU::show()
{
    printf("Frequency:%f\n", freq);
    printf("Mode:");
    switch(mode)
    {
        case(support):
            printf("Support Hyper-Threading\n");
        case(nosupport):
            printf("Not support Hyper-Threading\n");
        default:
            break;
    }
    printf("Core Number:");
}
```

```
switch(cores)

{
    case(single):
        printf("1\n");
    case(dual):
        printf("2\n");
    case(quad):
        printf("4\n");
    default:
        break;
}

printf("Bit:");

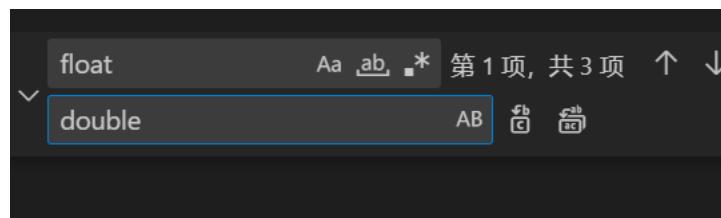
switch(bits)

{
    case(Bit32):
        printf("32-Bits\n");
    case(Bit64):
        printf("64-Bits\n");
    default:
        break;
}
}
```

```
int main()
{
    CPU N3160(2662.5, support, quad, Bit64);
    printf("Sizeof CPU: %llu\n", sizeof(CPU));
    CPU n2(N3160);
    N3160.show();
    return 0;
}
```

```
xe=D:\mingw64\bin\gdb
Sizeof CPU: 16
结果: Frequency:2662.500000
```

改后源码区别：



改后结果：

```
xe=D:\mingw64\bin\gdb.e
Sizeof CPU: 24
Frequency:2662.500000
```

再改成 long double:

```
xe=D:\mingw64\bin\gdb.e
Sizeof CPU: 32
```

发现其 size 随成员 size 变大而变大。

6

运行并分析以下代码的运行结果。

```
xe=D:\mingw64\bin\gdb.e
0
1
```

分析：

Union 共享内存：

```
1 #include <iostream>
2 using namespace std;
3
4 class A {
5 public:
6     int a;
7 };
8
9 union U {
10    A a;
11    double d;
12 };
13
14 int main() {
15    U u;
16    A a;
17    a.a = 1;
18    u.a = a;
19    u.d = 1.0;
```

```
1 #include <iostream>
2 using namespace std;
3
4 class A {
5 public:
6     int a;
7 };
8
9 union U {
10    A a;
11    double d;
12 };
13
14 int main() {
15    U u;
16    A a;
17    a.a = 1;
18    u.a = a;
19    u.d = 1.0;
20    cout << u.a.a << endl;
```

在改变 d 时也改变了 a。

结论分析与体会：

非常好的实验，让我对类的组合，union 理解更深刻。