

cha6 数据抽象、对象与类 练习

李宇豪 21305412

语法回顾与掌握

1. 尝试写一个类，包含最全的基本结构，包括：构造函数（成员初始列表）；析构函数；公开类型数据成员和函数成员，函数成员有一般函数，带 const 结尾的函数，静态成员函数，友元函数；私有类型的数据成员，私有的常量；保护类型的成员数据和函数。

```
class Student
{
public:
    static int studentCount;
    Student(char* sname, int ssex, int smark):sex(ssex),math_mark(smark)
    {
        int n = strlen(sname);
        name = new char [n];
        name = sname;
    }
    ~Student()
    {
        delete [] name;
    }

    char* name;
    int sex; // 0 表示男, 1 表示女

    void input_mark(int mark);
    friend void print_information(Student stu);
    int get_studentID() const;
    static void getCount();

protected:
    int studentID;
    int get_mark();

private:
    const char school[16] = "物理学院";
    int math_mark;
    void a_fun()
    {
        cout << "还没想好这个函数用来干什么" << endl;
    }
};
```

```

int Student::studentCount = 0;

void Student::input_mark(int mark)
{
    math_mark = mark;
}

int Student::get_studentID() const
{
    return studentID;
}

void Student::getCount()
{
    cout << "已创建" << studentCount << "个学生对象。";
}

void print_information(Student stu)
{
    cout << "学生姓名: " << stu.name << endl;
    cout << "学号: " << stu.studentID << endl;
}

int Student::get_mark()
{
    return math_mark;
}

```

2. 数据成员初始化的方法有哪些？常量怎样初始化？

成员初始化方式主要有：就地初始化、列表初始化、赋值初始化

就地初始化是在声明数据成员时直接用 = 赋值；

列表初始化是在构造函数中采用 A(a_) : a(a_) 的方式初始化；

赋值初始化是在成员函数（尤其是构造函数）中对成员数据进行赋值。

还可以使用拷贝构造函数，用已经存在的对象初始化类成员。

常量可以直接采用就地初始化，即：const int A = 1;

3. 类与对象是什么关系？对象有哪些创建方法？对象能使用类的哪些成员？

类是对同一类型的对象的抽象，对象则是类的一个实例，二者是抽象与具象的关系。

创建对象可以直接用 ClassName objectName(initialization) 的方式创建，

也可以用指针的方式创建

```
ClassName* objectName = new ClassName(initialization);
```

```
delete objectName;
```

4. 什么是拷贝构造函数，什么时候会被使用？什么是隐式的拷贝构造函数？

拷贝构造函数是在创建对象时，用已经存在的同类的对象来初始化新创建的对象。

拷贝构造函数通常用于定义对象、将对象作为参数传递给函数，将对象作为函数的返回值。

隐式的拷贝构造函数是指如果在定义类时没有显式的定义拷贝构造函数，编译器会隐式的添加一个默认的拷贝构造函数。

5. const 成员函数意味着什么？

```
class A
{
    Type function() const
    {
    }
}
```

用 const 修饰成员函数，意味着该成员函数只能访问数据成员，但是不能修改数据成员的值。

6. 什么是静态成员？静态成员怎样访问？静态成员函数中能使用非静态数据吗？为什么？

```
class A
{
    static Type data;
    static Type function(){}
}
```

静态成员是在类的所有对象中共享的类成员。对普通的类成员来说，每个对象使用的类成员都是不同的，比如 objectA.x 和 objectB.x 中的两个 x 是两个不同的 x。但是对于静态成员 static int y 来说，objectA.y 和 objectB.y 中两个 y 是同一个 y。

静态成员有三种访问方式：按对象访问、按对象指针访问、按类名访问

例如，对于 ClassName object1, * object2 一个静态成员 static int data

按对象访问，即直接用 object1.data 访问

按对象指针访问，即用 object2->data 访问

按类名访问，即用 ClassName::data 访问

静态成员函数不能调用非静态数据，因为静态函数成员没有 this 指针，不会将对象中的数据成员传递到函数中。

7. 友元能否访问对象里的私有成员？为什么能？

可以，因为友元被赋予了与类成员函数相同的访问权限

上机练习

1. 写一个点的类和圆的类

先写一个点的类，

- 1) 点包括 x,y 坐标的成员数据；
- 2) 一个构造函数 Point(double x0, double y0)
- 3) 一点与另外一点的距离 double distance(Point& p)

再写一个圆的类，

- 1) 包含 Point 类型的圆心，半径 rad，均为私有类型；
- 2) 常量 PI；
- 3) 面积 double area()，公开类型；
- 4) 判断一个点是否在圆内，bool InCircle(Point &p)，用上点类里的距离函数。

```

class Point
{
protected:
    double x;
    double y;
public:
    Point(double x0, double y0) : x(x0), y(y0)
    {
    }
    double distance(Point& p);
};

double Point::distance(Point& p)
{
    double s = sqrt(pow((x - p.x), 2) + pow((y - p.y), 2));
    return s;
}

class Circle : private Point
{
private:
    double rad;
    const double PI = 3.1415926;
public:
    Circle(double x0, double y0, double r0) : Point(x0, y0), rad(r0)
    {
    }
    double area();
    bool InCircle(Point& p);
};

double Circle::area()
{
    return PI * pow(rad, 2);
}

bool Circle::InCircle(Point& p)
{
    double s = distance(p);
    if (s < rad)
        return true;
    else
        return false;
}

```

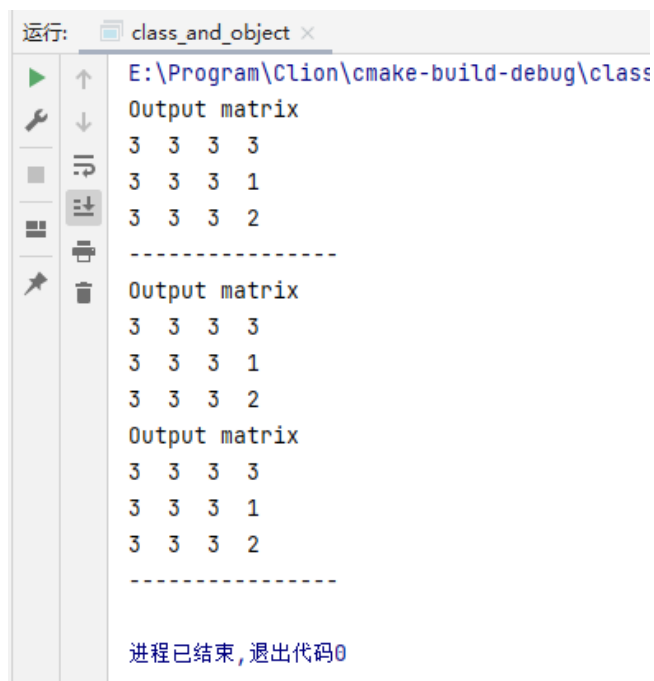
2. 利用附件提供的 Matrix, Vector 类，设计并编写下面的成员函数。

这里的设计与讲义有类似

1) 写 Matrix, Vector 拷贝构造函数，并测试其正确性；

```
Matrix(const Matrix& _ma)
{
    row = _ma.row;
    col = _ma.col;
    p_data = new double[row * col];
    for (int i = 0; i < row*col; i++)
        p_data[i] = _ma.p_data[i];
}

Vector(const Vector& _vec)
{
    size = _vec.size;
    p_data = new double [size];
    for (int i = 0; i < size; i++)
        p_data[i] = _vec.p_data[i];
}
```



```
运行: class_and_object x
E:\Program\Clion\cmake-build-debug\class
Output matrix
3 3 3 3
3 3 3 1
3 3 3 2
-----
Output matrix
3 3 3 3
3 3 3 1
3 3 3 2
Output matrix
3 3 3 3
3 3 3 1
3 3 3 2
-----
进程已结束,退出代码0
```

2) 写向量与向量点乘的运算。

```
double DotProduct(Vector& _vec)
{
    double sum = 0;
    for(int i = 0; i < size; i++)
    {
        sum += p_data[i] * _vec.p_data[i];
    }
    return sum;
}
```

3) 再设计一个 Vector 的成员函数，返回第 i 个元素值的引用。

```
double& data(int i)
{
    return p_data[i];
}
```

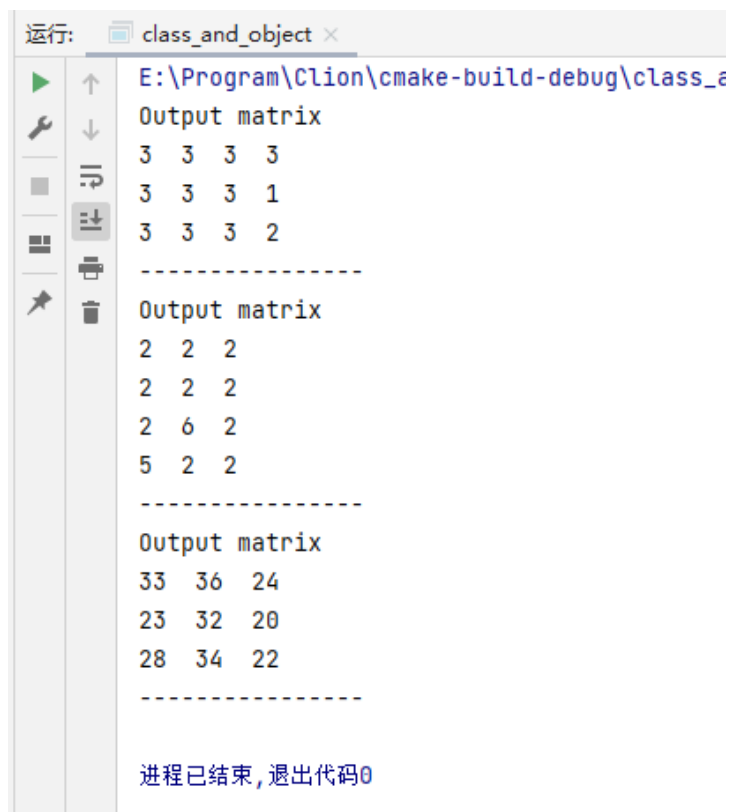
这个设计的目的是获得第 i 个元素值的引用后，可以在类外面设置其值。方便第 5) 问。

4) 以不同方式写矩阵乘以矩阵的函数，函数返回值为矩阵的引用，即在函数里分配好矩阵的空间。这也是为什么我们一定要做好第一问题的原因，如果 1) 的拷贝构造函数不提供，程序会怎样？

成员函数

```
Matrix multiply(Matrix& _ma)
{
    Matrix* mul_Matrix = new Matrix(row, _ma.col);
    for(int i = 0; i < row ; i++)
    {
        for(int j = 0; j < _ma.col ; j++)
        {
            double sum = 0;
            for(int k = 0; k < col; k++)
            {
                sum += p_data[i*col + k] * _ma.p_data[k*_ma.col + j];
            }
            mul_Matrix->data(i,j) = sum;
        }
    }
    return *mul_Matrix;
}
```

运行结果:



```
运行: class_and_object x
E:\Program\Clion\cmake-build-debug\class_a
Output matrix
3 3 3 3
3 3 3 1
3 3 3 2
-----
Output matrix
2 2 2
2 2 2
2 6 2
5 2 2
-----
Output matrix
33 36 24
23 32 20
28 34 22
-----
进程已结束,退出代码0
```

友元函数：

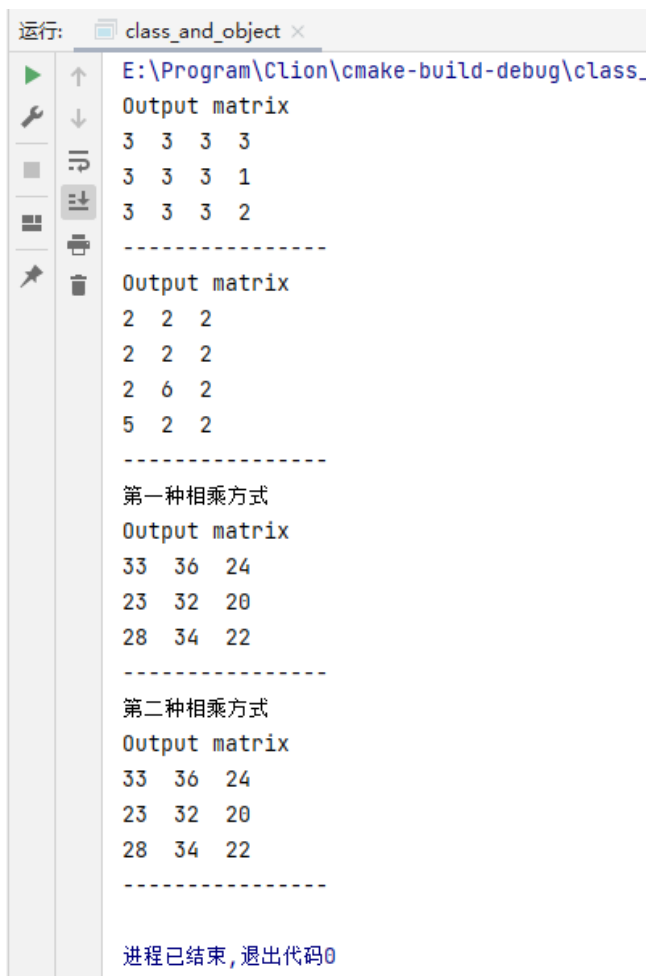
类内部声明：

```
friend Matrix multiply(Matrix &a, Matrix &b);
```

类外部定义：

```
Matrix multiply(Matrix &a, Matrix &b)
{
    Matrix* mul_Matrix = new Matrix(a.row, b.col);
    for(int i = 0; i < a.row ; i++)
    {
        for(int j = 0; j < b.col ; j++)
        {
            double sum = 0;
            for(int k = 0; k < a.col; k++)
            {
                sum += a.p_data[i*a.col + k] * b.p_data[k*b.col + j];
            }
            mul_Matrix->data(i,j) = sum;
        }
    }
    return *mul_Matrix;
}
```

运行结果：



```
运行: class_and_object x
E:\Program\Clion\cmake-build-debug\class_
Output matrix
3 3 3 3
3 3 3 1
3 3 3 2
-----
Output matrix
2 2 2
2 2 2
2 6 2
5 2 2
-----
第一种相乘方式
Output matrix
33 36 24
23 32 20
28 34 22
-----
第二种相乘方式
Output matrix
33 36 24
23 32 20
28 34 22
-----
进程已结束,退出代码0
```

如果返回值为 Matrix&, 函数应该如何设计?

```
Matrix& multiply2(Matrix& _ma)
{
    Matrix* mul_Matrix = new Matrix(row, _ma.col);
    for(int i = 0; i < row ; i++)
    {
        for(int j = 0; j < _ma.col ; j++)
        {
            double sum = 0;
            for(int k = 0; k < col; k++)
            {
                sum += p_data[i*col + k] * _ma.p_data[k*_ma.col + j];
            }
            mul_Matrix->data(i,j) = sum;
        }
    }
    return *mul_Matrix;
}
```

返回值为 Matrix 与 Matrix&, 从效率上看, 哪个设计更好? 为什么?

如果返回的是值 Matrix, 编译器首先会创建一个临时变量, 将函数的返回值存储到临时变量中, 然后再将临时变量赋值给指定的变量。如果返回的是引用 Matrix&, 实际上是返回了一个指向 Matrix 的隐式指针。从效率上看, 返回引用的效率会更高一点, 尤其是遇到返回值是一个复杂的类对象, 并且程序中会大量调用该函数的情况, 返回引用要比返回值效率更高。

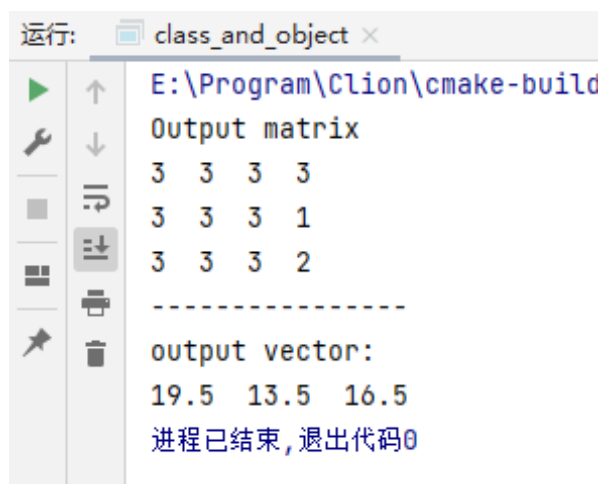
但是我在这里有一个疑问:

如果返回引用的话, 不能对局部变量进行引用, 所以必须要用 new 动态分配内存。如果遇到类似上面的情况, 要在函数内部分配内存, 那么返回引用后, 函数内部的指针会被释放, 就无法在程序内再释放引用的这块内存, 可能会导致内存泄露的问题。请问老师这种情况该怎么处理呢?

5) 重载一个矩阵和矢量相乘的函数, 同样地, 注意返回的值为向量时, 这个向量要在函数里使用动态分配的方法来构造, 避免函数结束时释放空间。

```
Vector& multiply(Vector& _vec)
{
    Vector* mul_Vector = new Vector(row);
    for(int i = 0; i < row ; i++)
    {
        double sum = 0;
        for(int k = 0; k < col; k++)
        {
            sum += p_data[i*col + k] * _vec.getdata(k);
        }
        mul_Vector->data(i) = sum;
    }
    return *mul_Vector;
}
```


运行结果：



```
运行: class_and_object x
E:\Program\Clion\cmake-build
Output matrix
3 3 3 3
3 3 3 1
3 3 3 2
-----
output vector:
19.5 13.5 16.5
进程已结束,退出代码0
```