上机实验报告 3

姓名：王越洋　　学号：22009200894

1. 实验一

1.1题目

建立一个对象数组，内放 5 个学生的数据（学号、成绩）。
（1）用指针指向数组首元素，输出第 1，3，5 个学生的数据；
（2）设立一个函数 max，用指向对象的指针作函数参数，在 max 函数中找出 5 个学生中成绩最高者，并输出其学号。

1.2 代码

```
#include <iostream>
using namespace std;

struct Student {
    int id;       // 学号
    double score; // 成绩
};

void max(Student* students) {
    int maxIndex = 0;
    for (int i = 1; i < 5; i++) {
        if (students[i].score > students[maxIndex].score) {
            maxIndex = i;
        }
    }
    cout << "成绩最高者的学号是: " << students[maxIndex].id << endl;
}

int main() {
    Student students[5] = {
        {1001, 85.5},
        {1002, 92.0},
        {1003, 78.3},
        {1004, 88.4},
        {1005, 91.6}
    };

    Student* ptr = students;

    // 输出第1、3、5 个学生的数据
```

```
    cout << "第 1 个学生: 学号 = " << ptr->id << ", 成绩 = " << ptr->score << endl;
    cout << "第 3 个学生: 学号 = " << (ptr + 2)->id << ", 成绩 = " << (ptr + 2)->score << endl;
    cout << "第 5 个学生: 学号 = " << (ptr + 4)->id << ", 成绩 = " << (ptr + 4)->score << endl;

    // 调用 max 函数，输出成绩最高的学号
    max(ptr);

    return 0;
}
```

1.3 分析

（1）学生数据结构（Student）：Student 结构体包含两个成员变量：id（学号）和 score（成绩）。结构体用于存储与学生相关的数据。学号采用 int 类型，成绩采用 double 类型。

（2）数组和指针： 定义了一个包含 5 个 Student 数据的数组 students[5]，用于存储 5 个学生的学号和成绩。数组名 students 本身是一个指向数组首元素的指针，因此可以通过指针操作来访问数组中的元素。定义一个指针 Student* ptr = students，指向数组的首元素，利用指针的偏移操作来访问数组中的其他学生数据。

（3）输出指定学生数据： 通过指针操作数组元素时，使用 ptr 访问第 1 个学生的数据，使用 (ptr + 2) 访问第 3 个学生的数据，使用 (ptr + 4) 访问第 5 个学生的数据。这种方式通过指针偏移来访问数组中的任意元素。

（4）成绩最高者的查找（max 函数）： max 函数接受一个 Student 类型的指针数组作为参数，通过遍历数组来查找成绩最高的学生。遍历过程中比较每个学生的成绩，记录成绩最大者的索引，最终输出该学生的学号。

2. 实验二

2.1题目

定义一个复数类 Complex，重载运算符“+”，“-”，“*”，“/”，使之能用于复数的加、减、乘、除。运算符重载函数作为 Complex 类的成员函数。

编程序，分别求两个复数之和、差、积和商。

2.2代码

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


class Complex {
private:
    double real; // 实部
    double imag; // 虚部
```

```
12.
13.public:
14.    Complex(double r = 0, double i = 0) : real(r), imag(i) {}
15.
16.    // 重载加法运算符
17.    Complex operator+(const Complex& other) {
18.        return Complex(real + other.real, imag + other.imag);
19.    }
20.
21.    // 重载减法运算符
22.    Complex operator-(const Complex& other) {
23.        return Complex(real - other.real, imag - other.imag);
24.    }
25.
26.    // 重载乘法运算符
27.    Complex operator*(const Complex& other) {
28.        return Complex(real * other.real - imag * other.imag,
29.                       real * other.imag + imag * other.real);
30.    }
31.
32.    // 重载除法运算符
33.    Complex operator/(const Complex& other) {
34.        double denominator = other.real * other.real + other.imag * ot
   her.imag;
35.        return Complex((real * other.real + imag * other.imag) / denom
   inator,
36.                       (imag * other.real - real * other.imag) / denom
   inator);
37.    }
38.
39.    // 输出复数
40.    void display() {
41.        cout << real << (imag >= 0 ? " + " : " - ") << abs(imag) << "i
   " << endl;
42.    }
43.};
44.
45.int main() {
46.    Complex c1(4, 5), c2(1, -2);
47.
48.    Complex sum = c1 + c2;
49.    Complex diff = c1 - c2;
50.    Complex prod = c1 * c2;
51.    Complex quot = c1 / c2;
```

```
52.
53.     cout << "c1 + c2 = "; sum.display();
54.     cout << "c1 - c2 = "; diff.display();
55.     cout << "c1 * c2 = "; prod.display();
56.     cout << "c1 / c2 = "; quot.display();
57.
58.     return 0;
59.}
```

2.3 分析

（1）复数类（Complex）： Complex 类表示复数，包含两个成员变量：real（实部）和 imag（虚部）。这些成员变量用于存储复数的实部和虚部。

（2）运算符重载：

- 加法（+）：复数加法的规则是：实部相加，虚部相加。因此重载加法运算符时，通过分别相加两个复数的实部和虚部来得到结果。
- 减法（-）：复数减法的规则是：实部相减，虚部相减。重载减法运算符时，通过分别相减两个复数的实部和虚部来得到结果。
- 乘法（*）：复数乘法的公式为：(a + bi)(c + di) = (ac - bd) + (ad + bc)i。在乘法运算符的重载中，按照此公式计算新的实部和虚部。
- 除法（/）：复数除法的公式为：(a + bi) / (c + di) = [(a + bi)(c - di)] / (c^2 + d^2)。首先计算共轭复数，然后根据公式计算商的实部和虚部。

（3）成员函数 display： 复数的输出形式为“实部 + 虚部 i”或“实部 - 虚部 i”，根据虚部的符号调整输出格式。在 display 函数中，判断虚部的符号，确保输出的复数形式符合标准的数学表达。

3. 实验三

3.1 题目

对于 2 行 3 列矩阵，重载流插入运算符“<<”和流提取运算符“>>”，使之能用于矩阵的输入和输出。

3.2 代码

```
1. #include <iostream>
2. using namespace std;
3.
4. class Matrix {
5. private:
6.     int mat[2][3]; // 2 行 3 列的矩阵
7.
8. public:
9.     // 输入矩阵
10.    friend istream& operator>>(istream& in, Matrix& m) {
11.        cout << "请输入矩阵元素（2 行 3 列）:" << endl;
12.        for (int i = 0; i < 2; i++) {
13.            for (int j = 0; j < 3; j++) {
```

```
                in >> m.mat[i][j];
            }
        }
        return in;
    }

    // 输出矩阵
    friend ostream& operator<<(ostream& out, const Matrix& m) {
        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < 3; j++) {
                out << m.mat[i][j] << " ";
            }
            out << endl;
        }
        return out;
    }
};

int main() {
    Matrix m;

    // 输入矩阵
    cin >> m;

    // 输出矩阵
    cout << "矩阵为:" << endl;
    cout << m;

    return 0;
}
```

3.3 分析

（1）矩阵类（Matrix）： Matrix 类表示一个 2 行 3 列的矩阵，内部使用一个二维数组 mat[2][3] 存储矩阵的元素。每个矩阵对象包含一个静态的 2 行 3 列数组来存储元素。

（2）流操作符重载：

- 流提取运算符（>>）：重载流提取运算符以支持矩阵元素的输入。使用 cin 提取数据并将其存入矩阵的数组 mat[2][3] 中。输入时通过 for 循环逐一接收用户输入的每个矩阵元素。
- 流插入运算符（<<）：重载流插入运算符以支持矩阵元素的输出。通过 cout 按行输出矩阵中的每个元素，输出时也使用 for 循环逐行逐列地打印矩阵元素。

（3）友元函数： 由于输入输出操作符需要访问矩阵类的私有成员（即存储矩阵元素的 mat 数组），因此将输入输出操作符声明为友元函数，使得这些函数能够直接访问 Matrix 类的私有数据。

4. 实验四

4.1 题目

定义 Time 类和 Date 类，Time 类为 Date 类的友元类，通过 Time 类中的 display 函数引用 Date 类对象的私有数据，输出年、月、日和时、分、秒。

4.2 代码

```
#include <iostream>
using namespace std;

class Date;

class Time {
public:
    void display(const Date& d); // 在 Time 类中定义显示 Date 的函数
};

class Date {
private:
    int year, month, day;

public:
    Date(int y, int m, int d) : year(y), month(m), day(d) {}

    // 让 Time 类成为 Date 类的友元类
    friend class Time;
};

void Time::display(const Date& d) {
    cout << "日期: " << d.year << "年 " << d.month << "
月 " << d.day << "日" << endl;
}

int main() {
    Date date(2024, 11, 14);
    Time time;

    time.display(date);

    return 0;
}
```

4.3 分析

（1）Time 类与 Date 类： Date 类包含三个私有成员变量：年（year）、月（month）和日（day），用于表示日期。Time 类包含一个成员函数 display，用于显示 Date 类中的日期信息。由于 Time 类需要访问 Date 类的私有成员，因此 Time 类被声明为 Date 类的友元类。

（2）友元类机制： 通过将 Time 类声明为 Date 类的友元类，Time 类的成员函数可以直接访问 Date 类的私有数据成员。这个机制允许在类之间进行灵活的数据共享，而无需暴露类的内部实现细节。

（3）display 函数： display 函数接收一个 Date 对象作为参数，并输出该对象的日期信息。由于 Time 类是 Date 类的友元类，能够直接访问并显示 Date 对象的年、月、日。