西安工业大学

XI`AN TECHNOLOGICAL UNIVERSITY

实验报告

实验课程名称 面向对象技术与C++

专　　业 计算机科学与技术

班　　级 21060102

姓　　名 张林鹏

学　　号 2021032449

实验学时

指导教师 耿军雪

成　　绩

2022 年 5 月 5 日

实验一 类与对象（一）——类与对象的定义

张林鹏_2021032449

一、实验目的

1. 熟悉类的构成，掌握类的定义方法；
2. 掌握对象的定义及对象成员的访问方法；
3. 初步熟悉类与对象简单应用及编程。

实验内容

程序1: exp_101.cpp

1. 程序的输出结果为:

```
real of complex A=3
imag of complex A=4
abs of complex A=5
```

2. 成员函数`set_complex(double r,double i)`的作用是设置实部和虚部的值

   `get_real( )`的作用是计算实部的值.

   `get_abs()`的作用是计算该复数的模.

3. 将main()函数中的语句行`cout <<A.get_real()<<endl`改为`cout <<A. real<<endl`,重新编译程序,将出现 编译错误 , 其原因是 real 是私有成员变量, 不能直接在类外部访问.

程序2: exp_102.cpp

1. 你分析的输出结果是:

```
x=20  y=30
x=20  y=30
x=25  y=35
```

程序的实际结果是:

```
x=20  y=30
x=20  y=30
x=25  y=35
```

程序3

1. 完善`hdata.h`中类的定义:

```
class Date
{
private:
    int year,month,day;
public:
    void set_date(int y=2000,int m=1,int d=1)  //对数据成员赋值
    {
        year = y;
        month = m;
        day = d;
    }
    int get_year()                          //返回year
    {
        return year;
    }
    int get_month()                        //返回month
    {
        return month;
    }
    int get_day()                          //返回day
    {
        return day;
    }
    int isleapyear(void);                   //是闰年返回1,不是闰年返回0
    void print_date(void)
    {
        cout<<year<<' -' <<month<<' -' <<day<<endl;
    }
};
int Date::isleapyear(void)                  //是闰年返回1,不是闰年返回0
{
    if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0)
        return 1;
    else
        return 0;
}
```

2. 按注释要求完善下列程序(exp_103.cpp)

```
#include <iostream.h>
#include"hdate.h"
void main(void)
{
    Date da1, da2;
    int y, m, d;
    da1.set_date(2004, 5, 1);
    da1.print_date();
    cout <<"year =";
    cin >> y;
    cout <<"month =";
    cin >> m;
    cout <<"day =";
    cin >> d;
    da2.set_date(y, m, d);
// 调用方法set_date(),用消息y,m,d对da2的数据成员赋值
    cout << da2.get_year() << "年" << da2.get_month() << "月" << da2.get_day() <<
"日" << endl; // 调用方法输出将da2用“    年    月    日”格式输出年月日
    cout << "da2的年是否为闰年 :" << da2.isleapyear() << endl;
// 调用方法输出da2的年是否为闰年
}
```

程序设计实验

1. time.h:

```
#ifndef HTIME_H
#define HTIME_H

#include <bits/stdc++.h>
using namespace std;

class time
{
private:
    int hour;
    int minute;
    int second;
    time(int h, int m, int s): hour(h), minute(m), second(s) {}

public:
    time(): hour(0), minute(0), second(0) {}
    string output_time();
    void input_time();
    int output_hour();
    int output_minute();
    int output_second();
```

```
};

#endif
```

2. time.cpp

```
#include "htime.h"

string time::output_time()
{
    string s;
    s = to_string(hour) + ":" + to_string(minute) + ":" + to_string(second);
    return s;
}

void time::input_time()
{
    cout << "Enter hour: ";
    cin >> hour;
    cout << "Enter minute: ";
    cin >> minute;
    cout << "Enter second: ";
    cin >> second;

    if (second >= 60)
    {
        minute += second / 60;
        second %= 60;
    }
    if (minute >= 60)
    {
        hour += minute / 60;
        minute %= 60;
    }
    if (hour >= 24)
    {
        hour %= 24;
    }
}

int time::output_hour()
{
    return hour;
}

int time::output_minute()
{
    return minute;
}

int time::output_second()
```

```
{
    return second;
}
```

3. exp_104.cpp

```
#include "htime.h"

int main()
{
    class time t1;
    class time *t1_ptr = &t1;
    class time &t1_ref = t1;

    t1.input_time();
    cout << t1.output_time() << endl;
    cout << t1.output_hour() << "时" << t1.output_minute() << "分" <<
t1.output_second() << "秒" << endl;

    t1_ptr->input_time();
    cout << t1_ptr->output_time() << endl;
    cout << t1_ptr->output_hour() << "时" << t1_ptr->output_minute() << "分" <<
t1_ptr->output_second() << "秒" << endl;

    t1_ref.input_time();
    cout << t1_ref.output_time() << endl;
    cout << t1_ref.output_hour() << "时" << t1_ref.output_minute() << "分" <<
t1_ref.output_second() << "秒" << endl;
}
```

实验二 类与对象（二）——构造函数、析构函数及对象的应用

张林鹏_2021032449

一、实验目的

1. 理解构造函数、析构函数的意义及作用, 掌握构造函数、析构函数的定义及调用时间, 熟悉构造函数的种类.
2. 理解this指针及使用方法, 熟悉对象数组、对象指针、对象引用的定义及使用方法, 熟悉对象作为函数参数的使用方法.
3. 熟悉类与对象的应用及编程.

二、实验内容

程序1: exp_201.cpp

1. 程序的运行输出结果为:

```
constructing!
destructing!
```

2. 该输出结果说明构造函数`Myclass( )`是在 构造函数时 执行的,

   而析构函数`~Mycalss( )`是在 对象销毁时 执行的.

3. 将`main( )`函数中的`Myclass ob` 改为: `Mycls ob[2]` 后, 运行程序的输出结果为:

```
constructing!
constructing!
destructing!
destructing!
```

4. 将`main( )`中的`Myclass ob[2]`改为`Myclass *ob; ob = new Myclass[2]`后, 运行程序的输出结果为:

```
constructing!
```

```
constructing!
```

5. 在4. 的基础上, 在程序的末尾加入 `delete []ob` 后运行, 程序的输出结果为:

```
destructing!
destructing!
```

6. 比较3 - 5的输出结果, 说明:

   使用new创建的对象需要手动删除,否则会导致内存泄漏

程序2: exp_202.cpp

7. 运行该程序输出的结果为:

8. 程序中的成员函数 `A(A &ob)` 成为 拷贝构造函数 .

   该函数的执行时间时在执行 创建对象时 被调用的.

9. 将 `main( )` 中的 `A ob2(ob1);` 改为 `A ob2=ob1;` , 重新运行程序, 观察输出结果, 说明拷贝构造函数也可以在 赋值操作 时调用.

10. 将 `main( )` 函数中加注释的语句去掉前面的 `//` , 重新运行程序, 观察输出结果, 说明执行 `ob = ob1;` 时 会 调用拷贝构造函数, 原因是 `ob = ob1;` 只是对对象的 赋值操作 .

程序3: exp_203.cpp

1. 你分析的输出结果是:

```
程序的实际输出结果是:
调用func1:
拷贝构造函数被调用!
func1: a=10  b=10
析构函数被调用!
调用func2:
func2: a=10  b=10
调用func3:
func3: a=10  b=10
main: a=10  b=10
析构函数被调用!
```

2. 程序实际输出结果是:

```
程序的实际输出结果是:
```

```
调用func1：
拷贝构造函数被调用!
func1: a=10  b=10
析构函数被调用!
调用func2：
func2: a=10  b=10
调用func3：
func3: a=10  b=10
main: a=10  b=10
析构函数被调用!
```

程序4: exp_204.cpp

11. 程序中:
    i. 处应为: `person;`
    ii. 处应为: `name = new char[strlen(pn) + 1];`
    iii. 处应为: `if(name!=NULL) strcpy(name,pn);`
    iv. 处应为: `new char[strlen(ob.name)+1];`
    v. 处应为: `if(name!=NULL) strcpy(name,ob.name);`

程序5: exp_205.cpp

12. 完善类的定义, 程序中:
    i. 处的定义应为: `for(i=0;i<m;i++) {sc[i]=x[i];sum+=sc[i];}`
    ii. 处的定义应为: `for(i=0;i<m;i++) {sc[i]=x[i];sum+=sc[i];}`
13. 完善 `main( )` 函数, 程序中:
    i. 处的定义应为: `new score[n];`
    ii. 处的定义应为: `p,n,m;`
    iii. 处的定义应为: `delete []p;`
    iv. 处的定义应为: `p[i].set_score(x,m);`
    v. 处的定义应为: `p[i].print_score();`
    vi. 处的定义应为: `aver.set_score(s,m);`
    vii. 处的定义应为: `{k=j;a=p[j].get_aver();}`
    viii. 处的定义应为: `{t=p[i];p[i]=p[k];p[k]=t;}`

程序设计实验

1. result.h:

```cpp
#ifndef RESULT_H
#define RESULT_H

#include <bits/stdc++.h>
#define UMPIRE 5                // 裁判数量
using namespace std;

class result
{
private:
    struct player
    {
        int number;             //运动员编号
        string name;            //运动员姓名
        vector<double> score;   //运动员成绩
        double finalResult;     //运动员最终成绩

        player() : number(0), name(""), finalResult(0) {}
    };

    int playerNumber;           //运动员数量
    int judgeNumber = UMPIRE;   //裁判数量
    vector<player> players;     //运动员信息

public:
    result(): playerNumber(0) {}
    result(int p, int j) : playerNumber(p), judgeNumber(j) {}
    void inputPlayer();
    void inputScore();
    void calculateResult();
    void outputResult();
    void outputRank();
};

void result::inputPlayer()
{
    player temp;
    cout << "请输入运动员数量：" << endl;
    cin >> playerNumber;
    for (int i = 0; i < playerNumber; i++)
    {
        cout << "请输入第" << i + 1 << "个运动员的编号、姓名：";
        cin >> temp.number >> temp.name;
        players.push_back(temp);
    }
    cout << endl;
}

void result::inputScore()
{
    for (int i = 0; i < playerNumber; i++)
    {
```

```
        cout << "请输入第" << i + 1 << "个运动员的成绩：" << endl;
        for (int j = 0; j < judgeNumber; j++)
        {
            double temp;
            cout << "第" << j + 1 << "个裁判的成绩：";
            cin >> temp;
            players[i].score.push_back(temp);
        }
    }
    cout << endl;
}

void result::calculateResult()
{
    for (int i = 0; i < playerNumber; i++)
    {
        double sum = 0;
        sort(players[i].score.begin(), players[i].score.end());
        for (int j = 1; j < judgeNumber - 1; j++)
        {
            sum += players[i].score[j];
        }
        players[i].finalResult = sum / (judgeNumber - 2);
    }
}

void result::outputResult()
{
    cout << "运动员编号\t运动员姓名\t最终成绩" << endl;
    for (int i = 0; i < playerNumber; i++)
    {
        cout << players[i].number << "\t\t" << players[i].name << "\t\t" <<
players[i].finalResult << endl;
    }
}

void result::outputRank()
{
    sort(players.begin(), players.end(), [](player a, player b) { return
a.finalResult > b.finalResult; });
    cout << "运动员编号\t运动员姓名\t最终成绩" << endl;
    for (int i = 0; i < playerNumber; i++)
    {
        cout << players[i].number << "\t\t" << players[i].name << "\t\t" <<
players[i].finalResult << endl;
    }
}

#endif
```

2. exp_206.cpp:

```cpp
#include <bits/stdc++.h>
#include "result.h"
using namespace std;

int main()
{
    result r1;
    r1.inputPlayer();
    r1.inputScore();
    r1.calculateResult();
    r1.outputResult();
    r1.outputRank();
}
```

实验三 类与对象（三）——静态成员、常量成员、友元、对象成员

张林鹏_2021032449

一、实验目的

1. 理解静态数据成员、静态成员函数的作用, 熟悉其应用;
2. 理解常量对象、常量数据成员、常量成员函数作用, 熟悉相互关系及应用;
3. 熟悉友元函数、友元类的定义及应用;
4. 熟悉对象成员 (容器类) 的应用;
5. 进一步熟悉类与对象的应用及编程.

二、实验内容

程序1: exp_301.cpp

1. 运行该程序的输出结果为:

2. 由输出结果可知, 静态成员属于 类 , 在类中只有 一份 拷贝.

3. 程序中的 `int counter::count=100;` 的作用是: 给静态数据成员count赋初值为100

   如果将其放在 `main( )` 函数中, 重新编译程序, 会出现 编译错误 , 其原因是 静态成员的初值必须在类外部定义时给出.

程序2: exp_302.cpp

4. 编译程序, 会出现 编译错误 , 其出错原因是 头文件 `hhpoint.h` 和 `point.h` 中的类定义重复.
5. 将程序中的 (1) 改为注释, 再将 (2) 行的注释去掉, 再编译运行程序, 程序的输出结果为:

```
Before moving: p: x=20 y=40
center: x=100 y=100
```

6. 将程序中 (3) ~ (6) 行首的注释去掉, 程序的输出结果为:

```
Before moving: p: x=20 y=40
```

```
center: x=100 y=100
After moving: p: x=30 y=60
```

7. 将程序中 (7) 行首的注释去掉, 编译程序时无错误, 而运行时会出现错误, 其原因是 center是一个const对象，它的成员函数move()没有被声明为const成员函数，因此不能被const对象调用 .

程序3: exp_303.cpp

8. 编译程序出错的原因是 girl 类中的 name 指针未初始化 .

9. 在 girl 类中加上 `friend void disp(girl &g);` 重新编译运行程序, 其输出结果为:

```
姓名:李小丫 年龄:12
姓名:王永兰 年龄:15
姓名:赵梦美 年龄:13
```

其中, `friend void disp(girl &g);` 表示函数 `disp( )` 为girl类的 友元函数 .

程序4: exp_304.cpp

1. 你的分析输出结果是:

```
(1): 15 (2): 30
```

2. 程序实际输出结果是:

```
(1): 15 (2): 30
```

程序5: exp_305.cpp

1. 你的分析输出结果是:

```
女孩姓名:李小丫 年龄:12
男孩姓名:张海兵 年龄:15
```

2. 程序实际输出结果是:

```
女孩姓名:李小丫 年龄:12
男孩姓名:张海兵 年龄:15
```

程序6:

1. 你的分析输出结果是:

```
姓名:张小三 性别:男 出生日期:1985年12月15日
姓名:李小丫 性别:女 出生日期:1986年3月9日
```

2. 程序实际输出结果是:

```
姓名:张小三 性别:男 出生日期:1985年12月15日
姓名:李小丫 性别:女 出生日期:1986年3月9日
```

程序7: exp_307.cpp

- 补全后的代码:

```cpp
#include <iostream>
#include <string.h>
using namespace std;
class boy; // 类boy的声明;
class girl
{
private:
    char *name;
    int age;

public:
    girl(char *na, int n)
    {
        name = new char[strlen(na) + 1];
        strcpy(name, na);
        age = n;
    }
    ~girl(void)
    {
        delete name;
    } // 释放new分配的内存
    void disp(boy &b);
};
class boy
{
private:
    char *name;
    int age;
    friend class girl; // 声明girl是boy的友元类
public:
```

```
    boy(char *na, int n)
    {
        name = new char[strlen(na) + 1];
        strcpy(name, na);
        age = n;
    }
    ~boy(void)
    {
        delete name;
    } // 释放new分配的内存
};
void girl::disp(boy &b)
{
    cout << "女孩姓名:" << name;
    cout << "  年龄:" << age << endl;
    cout << "男孩姓名:" << b.name; // b为boy类的一个成员
    cout << "  年龄:" << b.age << endl;
}
int main(void)
{
    girl g1("李小丫", 12);
    boy b1("张海兵", 15);
    g1.disp(b1);

    return 0;
}
```

程序设计实验:

- exp_308.cpp

```
#include <bits/stdc++.h>

class point
{
public:
    double x, y;
    point(double x, double y) : x(x), y(y) {}
    point() {}
    friend class distance;
};

class distance
{
public:
    double dis(point a, point b)
    {
        return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
    }
};

int main()
```

```
{
    point a(1, 2), b(3, 4);
    distance d;
    std::cout << d.dis(a, b) << std::endl;

    return 0;
}
```

实验四 继承（一）—— 派生类定义及访问权限

张林鹏_2021032449

一、实验目的

1. 理解继承的概念、特性及在C++语言中的实现方法;
2. 掌握C++语言派生类的定义, 熟悉不同的继承方式 (派生方式);
3. 掌握派生类构造函数的定义及在定义、释放派生类对象时构造函数、析构函数的执行顺序;
4. 掌握不同继承方式下, 基类的成员在派生类中的访问特性;
5. 初步熟悉派生类的应用.

二、实验内容

程序1: exp_401.cpp

1. 运行程序输出结果为:

```
x=10  y=20
x=10  y=20  z=30
x=10  y=20
```

2. 程序中`a.print( );`调用的是基类成员中的`print( )`成员函数, `b.print( );`调用的是Derived类成员中的`print( )`成员函数, `b.Base::print( );`调用的是类成员中的`print( )`成员函数.

3. 构造函数`Derived(float a=0,float b=0,float c=0):Base(a,b)`中的`Base(a,b)`的作用是:

   调用基类Base的构造函数,将a和b分别传递给基类构造函数中定义的x和y.

4. 将派生类定义中的`public`改为`private`, 重新编译程序, 程序中调用派生类函数语句会出现编译错误, 其原因是访问权限被限制, 无法访问私有成员.

程序2: exp_402.cpp

- 你认为的输出结果是:

```
基类构造函数被调用!
基类构造函数被调用!
派生类构造函数被调用!
x=10  y=20
x=10  y=20  z=30
派生类析构函数被调用!
基类析构函数被调用!
基类析构函数被调用!
```

- 程序运行输出结果是:

```
基类构造函数被调用!
基类构造函数被调用!
派生类构造函数被调用!
x=10  y=20
x=10  y=20  z=30
派生类析构函数被调用!
基类析构函数被调用!
基类析构函数被调用!
```

程序3: exp_403.cpp

- 你认为的输出结果是:

```
A 类的构造函数被调用!
Data 类的构造函数被调用!
B 类的构造函数被调用!
B 类的析构函数被调用!
Data 类的析构函数被调用!
A 类的析构函数被调用!
```

- 程序运行输出结果是:

```
A 类的构造函数被调用!
Data 类的构造函数被调用!
B 类的构造函数被调用!
B 类的析构函数被调用!
Data 类的析构函数被调用!
A 类的析构函数被调用!
```

程序4: exp_404.cpp

5. 程序中:
    i. 处应改为: `float a=0,float b=0,float c=0`

ii. 处应改为: `Base(a,b)`

iii. 处应改为: `float a=0,float b=0,float c=0`

iv. 处应改为: `setBase(a,b);`

程序设计实验:

- point.h:

```
#ifndef _POINT_H
#define _POINT_H

#include <bits/stdc++.h>
#define M_PI acos(-1)

class point
{
public:
    double x, y;
    point(double x, double y) : x(x), y(y) {}
    point() {}
    friend class distance;
};

class circle : public point
{
public:
    double r;
    circle(double x, double y, double r) : point(x, y), r(r) {}
    circle() {}

    void inputCircle();
    void outputCircle();
};

void circle::inputCircle()
{
    std::cout << "Input x(cm): ";
    std::cin >> x;
    std::cout << "Input y(cm): ";     std::cin >> y;
    std::cout << "Input r(cm): ";
    std::cin >> r;
}

void circle::outputCircle()
{
    std::cout << "Circumference: " << 2 * M_PI * r << "cm" << std::endl;
    std::cout << "Area: " << M_PI * r * r << "cm^2" << std::endl; }

#endif
```

- exp_405.cpp:

```cpp
#include "point.h"

int main()
{
    circle c;
    c.inputCircle();
    c.outputCircle();

    return 0;
}
```

实验五 继承（二）—— 基类对象与派生类对象

张林鹏_2021032449

一、实验目的

1. 理解公有继承、私有继承方式下, 类及对象的访问权限; 掌握公有继承方式下派生类对象访问基类私有成员、保护成员的方法; 了解私有继承方式下派生类对象访问基类私有成员、保护成员的方法.
2. .理解继承的传递性; 掌握公有继承方式下, 间接派生类对象访问间接基类私有成员、保护成员的方法; 了解私有继承方式下间接派生类对象访问间接基类私有成员、保护成员的方法.
3. 理解基类与派生类对象、指针、引用的兼容性规则, 熟悉派生类对象、指针、引用作为函数参数的各种方式.

二、实验内容

程序1: exp_501.cpp

1. 编译运行输出结果为:

```
x=10 y=20
x=10 y=20 z=30
x=10 y=20
```

2. 程序中的`b.get_x()`, `b.get_y()`表明在公有继承方式下, 派生类对象可以直接访问基类的公有成员变量和成员函数.

3. 将程序中的派生类定义由`public`改为`private`继承方式, 重新调试程序将出现编译错误, 原因是派生类对象不能直接访问基类的成员变量和成员函数.

4. 在派生类的公有成员定义 (即`public:`) 下面加上`Base::get_x;Base::get_y;`, 再重新编译运行程序, 将得到正确结果. `Base::get_x;Base::get_y;`成为使用基类作用域解析运算符来指定派生类的成员函数调用基类中的同名函数.

程序2: exp_502.cpp

5. 程序编译运行输出结果为:

```
x=10 y=20 z=30
```

6. 将`class B:public A`, `class C:public B`的继承方式改为`private`, 重新编译程序将会出现编译错误, 原因是派生类不能直接访问基类的成员变量和成员函数.

7. 在类`B`中增加公有成员函数`float get_x() {return A::get_x();}`,

   在类`C`中增加公有成员函数`float get_x() {return B::get_x();}`, `float get_y() {return B::get_y();}`,

   重新编译运行程序, 输出结果为:

```
x=10 y=20 z=30
```

程序3: exp_503.cpp

- 你分析的程序输出结果是:

```
x=25.5  y=35.5
x=25.5  y=35.5  z=50.5
x=25.5  y=35.5
x=25.5  y=35.5  z=50.5
x=25.5  y=35.5
```

- 程序实际运行输出结果是:

```
x=25.5  y=35.5
x=25.5  y=35.5  z=50.5
x=25.5  y=35.5
x=25.5  y=35.5  z=50.5
x=25.5  y=35.5
```

程序4: exp_504.cpp

- 你分析的程序输出结果是:

```
x=15  y=25
x=15  y=25
x=15  y=25
```

- 程序实际运行输出结果是:

```
x=15  y=25
x=15  y=25
x=15  y=25
```

程序5: exp_505.cpp

8. 修改对应编号下的代码
   i. 处应为: `pb->getx()`.
   ii. 处应为: `pb->gety()`.
   iii. 处应为: `dynamic_cast<Derived*>(pb)->getz()`.

程序设计:

- person.h:

```
#ifndef PERSON_H
#define PERSON_H

#include <bits/stdc++.h>
using namespace std;

class person
{
private:
    string name;
    string gender;
    int age;

public:
    person(string name = "", string gender = "", int age = 0);

    void setName(string name);
    void setGender(string name);
    void setAge(int age);

    string getName();
    string getGender();
    int getAge();

    void showInfo();
};

person::person(string name, string gender, int age)
{
    this->name = name;
    this->gender = gender;
```

```
    this->age = age;
}

void person::setName(string name)
{
    this->name = name;
}

void person::setGender(string gender)
{
    this->gender = gender;
}

void person::setAge(int age)
{
    this->age = age;
}

string person::getName()
{
    return name;
}

string person::getGender()
{
    return gender;
}

int person::getAge()
{
    return age;
}

void person::showInfo()
{
    cout << "Name: " << name << " Gender: " << gender << " Age: " << age << endl;
}

#endif
```

- teacher.h:

```
#ifndef TEACHER_H
#define TEACHER_H

#include "person.h"

class teacher : public person
{
private:
    string collage;
```

```
    string major;
    string education;
    string degree;
    string title;
    int teachingYears;

public:
    teacher(
            string name = "",
            string gender = "",
            int age = 0,
            string collage = "",
            string major = "",
            string education = "",
            string degree = "",              string title = "",
            int teachingYears = 0
            );
    void setCollage(string collage);
    void setMajor(string major);
    void setEducation(string education);
    void setDegree(string degree);
    void setTitle(string title);
    void setTeachingYears(int teachingYears);

    string getCollage();
    string getMajor();
    string getEducation();
    string getDegree();
    string getTitle();
    int getTeachingYears();

    void showInfo();
};

teacher::teacher(
    string name,
    string gender,
    int age,
    string collage,
    string major,
    string education,
    string degree,
    string title,
    int teachingYears)
{
    this->setName(name);
    this->setGender(gender);
    this->setAge(age);
    this->collage = collage;
    this->major = major;
    this->education = education;
    this->degree = degree;
    this->title = title;
```

```
    this->teachingYears = teachingYears;
}

void teacher::setCollage(string collage)
{
    this->collage = collage;
}

void teacher::setMajor(string major)
{
    this->major = major;
}

void teacher::setEducation(string education)
{
    this->education = education;
}

void teacher::setDegree(string degree)
{
    this->degree = degree;
}

void teacher::setTitle(string title)
{
    this->title = title;
}

void teacher::setTeachingYears(int teachingYears)
{
    this->teachingYears = teachingYears;
}

string teacher::getCollage()
{
    return collage;
}

string teacher::getMajor()
{
    return major;
}

string teacher::getEducation()
{
    return education;
}

string teacher::getDegree()
{
    return degree;
}
```

```
string teacher::getTitle()
{
    return title;
}

int teacher::getTeachingYears()
{
    return teachingYears;
}

void teacher::showInfo()
{
    cout << "Name: " << getName() << " Gender: " << getGender() << " Age: " <<
getAge() << endl;
    cout << "Collage: " << collage << " Major: " << major << " Education: " <<
education << " Degree: " << degree << " Title: " << title << " TeachingYears: " <<
teachingYears << endl;
}

#endif
```

- exp_506.h:

```
#include "teacher.h"
#include "person.h"
using namespace std;

int main()
{
    person p1("张三", "男", 20);
    teacher t1("李四", "女", 30, "计算机学院", "计算机科学与技术", "本科", "硕士", "教
授", 10);
    p1.showInfo();
    t1.showInfo();

    return 0;
}
```

实验六 继承（三）—— 多继承及继承的应用

> 张林鹏_2021032449

一、实验目的

1. 理解多继承的概念, 熟悉多继承的定义及应用;
2. 理解多继承方式下的二义性产生原因, 熟悉解决二义性的方法;
3. 进一步熟悉继承的综合应用.

二、实验内容

程序1: exp_601.cpp

1. 编译该程序, 会出现编译错误, 其原因是在定义Z类的`print()`成员函数中的`show()`函数调用时出现了二义性. 解决方法是:
2. 将`show(); //显示x的值`改为: `X::show()`.
3. 将`show(); //显示y的值`改为: `Y::show()`.
4. 调试成功后输出结果过为:

```
x=3  y=4  z=5
x=10  y=20  z=30
```

5. 成员函数`void set_xyz(int a,int b,int c)`中`set_x(a)`将派生类 **Z** 的对象 **x** 成员变量设置为 **a**,

   `set_y(b)`的作用是将派生类 **Z** 对象的 **y** 成员变量设置为 **b**.

程序2: exp_602.cpp

6. 编译运行程序的输出结果是:

```
Constructing base a=2
Constructing base1 b=3
```

```
Constructing base2 c=4
Constructing derived d=5
```

7. 从输出结果中可看出, 间接基类 `base` 被调用了两次, 产生原因是 `derived` 的两个直接基类 `base1`, `base2` 有用共同基类 `base`, 产生了二义性.

8. 将程序中:

   i. `class base1:public base` 改为: `class base1:virtual public base`;
   ii. `class base2:public base` 改为: `class base2:virtual public base`;
   iii. `derived(int x1,int x2,int x3,int x4):base1(x1,x2),base2(x1,x3)` 改为: `derived(int x1,int x2,int x3,int x4):base(x1),base1(x1,x2),base2(x1,x3)`,

   编译运行输出结果为:

```
Constructing base a=2
Constructing base1 b=3
Constructing base2 c=4
Constructing derived d=5
```

9. `virtual public base` 表示间接基类 `base` 为虚基类.

程序: exp_603.cpp & person.h & teacher.h & student.h & score.h

- exp_603.cpp

```cpp
#include <bits/stdc++.h>
#include "score.h"
using namespace std;

void input_base(score *p, int n);          // 学生基本数据输入
void input_score(score *p, int n, int m); // 学生成绩输入
void print_base(score *p, int n);          // 学生基本数据输出
void print_score(score *p, int n, int m); // 学生成绩输出
score &average(score *p, int n, int m);    // 普通函数：平均成绩计算
void sort(score *p, int n, int m);         // 普通函数：按平均成绩排序

int main()
{
    int n, m;
    cout << "学生人数：";
    cin >> n;
    cout << "考试科数：";
    cin >> m;
    score *p, aver;
    p = new score[n]; // 动态分配内存单元—动态数组
    if (p == NULL)
```

```
    {
        cout << "内存分配失败" << endl;
        return 0;
    }
    int ch;
    do
    {
        cout << "\n\n      请选择:\n";
        cout << "  1. 输入学生基本数据\n";
        cout << "  2. 输入学生成绩\n";
        cout << "  3. 计算课程平均成绩\n";
        cout << "  4. 输出学生基本数据\n";
        cout << "  5. 输出学生成绩\n";
        cout << "  6. 按平均成绩排序\n";
        cout << "  0. 退出\n";
        cout << "\n    输入你的选择:";
        cin >> ch;
        switch (ch)
        {
        case 1:
            input_base(p, n);
            break;
        case 2:
            input_score(p, n, m);
            break;
        case 3:
            aver = average(p, n, m);
            break;
        case 4:
            print_base(p, n);
            break;
        case 5:
            print_score(p, n, m);
            aver.print_score();
            break;
        case 6:
            sort(p, n, m);
            break;
        case 0:
            break;
        }
    } while (ch);
    delete[] p; // 释放内存
}

void input_base(score *p, int n) // 学生基本数据输入
{
    int i, id, y1, m1, d1, y2, m2, d2;
    char name[11], sex[3], dpt[20];
    cout << "\n  请输入学生基本数据:";
    for (i = 0; i < n; i++)
    {
        cout << "第" << i + 1 << "个学生:\n";
```

```
        cout << "学号:";
        cin >> id;
        cout << "姓名:";
        cin >> name;
        cout << "性别:";
        cin >> sex;
        cout << "出生年:";
        cin >> y1;
        cout << "出生月:";
        cin >> m1;
        cout << "出生日:";
        cin >> d1;
        cout << "所在系:";
        cin >> dpt;
        cout << "入学年:";
        cin >> y2;
        cout << "入学月:";
        cin >> m2;
        cout << "入学日:";
        cin >> d2;
        p[i].set_person(name, sex, y1, m1, d1);  // 完成函数的调用
        p[i].set_student(id, sex, y2, m2, d2); // 完成函数的调用
    }
}

void input_score(score *p, int n, int m)
{
    int i, j;
    float x[M];
    for (i = 0; i < n; i++)
    {
        cout << p[i].get_id() << p[i].get_name() << "的成绩：" << endl;
        for (j = 0; j < m; j++)
        {
            cout << "第" << j + 1 << "科成绩：";
            cin >> x[j];
        }
        p[i].set_score(x, i); // 完成函数的调用
    }
}

void print_base(score *p, int n)
{
    int i;
    for (i = 0; i < n; i++)
        p[i].print_base();
    cout << endl;
}

void print_score(score *p, int n, int m)
{
    int i;
    for (i = 0; i < n; i++)
```

```
        p[i].print_score();
}

score &average(score *p, int n, int m) // 用返回引用的方法
{
    int i, j;
    float s[M] = {0};
    static score aver; // 返回的对象必须是静态的
    for (j = 0; j < m; j++)
    {
        for (i = 0; i < n; i++)
            s[j] = s[j] + p[i].get_score(j);
        s[j] = s[j] / n;
    }
    aver.set_person("平均成绩", " ", 0, 0, 0);
    aver.set_score(s, j); // 完成函数的调用
    return aver;
}

void sort(score *p, int n, int m) // 选择法排序：完成空白处的内容
{
    score t;
    float a;
    int i, j, k;
    for (i = 0; i < n - 1; i++)
    {
        a = p[i].get_aver();
        k = i;
        for (j = i + 1; j < n; j++)
            if (p[j].get_aver() < a)
            {
                a = p[j].get_aver();
                k = j;
            }
        if (i != k)
        {
            t = p[i];
            p[i] = p[k];
            p[k] = t;
        }
    }
}
```

- person.h

```
#ifndef PERSON_H
#define PERSON_H

#include "date.h"
#include <string>
#include <string.h>
```

```
using namespace std;

class person
{
protected:
    char name[11];
    char sex[2];
    date birthday;

public:
    person(void); // 无参构造函数
    void set_person(char *na, char *s, int y, int m, int d);
    char *get_name(void)
    {
        return name;
    } // 完成成员函数的定义
    char *get_sex(void)
    {
        return sex;
    } // 完成成员函数的定义
    int get_year(void)
    {
        return birthday.get_year();
    } // 完成成员函数的定义
    int get_month(void)
    {
        return birthday.get_month();
    }
    int get_day(void)
    {
        return birthday.get_day();
    } // 完成成员函数的定义
    void print(void);
};

person::person(void) // 无参构造函数
{
    strcpy(name, "无名氏");
    strcpy(sex, "男");
    birthday.set_date(1980, 1, 1);
}

void person::set_person(char *na, char *s, int y, int m, int d)
{
    strcpy(this->name, na);
    strcpy(this->sex, s);
    birthday.set_date(y, m, d);
} // 完成成员函数的定义

void person::print(void)
{
    cout << "姓名:" << name << endl;
    cout << "性别:" << sex << endl;
```

```
    cout << "出生日期:" << birthday.get_year() << "年";
    cout << birthday.get_month() << "月";
    cout << birthday.get_day() << "日" << endl;
}

#endif
```

- date.h

```
#ifndef DATE_H
#define DATE_H
#include <bits/stdc++.h>

class date
{
private:
    int year, month, day; // 年、月、日三个私有成员
public:
    date(void)
    {
        year = 1980;
        month = 1;
        day = 1;
    }
    void set_date(int y, int m, int d)
    {
        year = y;
        month = m;
        day = d;
    } // 完成成员函数的定义
    int get_year(void)
    {
        return year;
    } // 完成成员函数的定义
    int get_month(void)
    {
        return month;
    } // 完成成员函数的定义
    int get_day(void)
    {
        return day;
    } // 完成成员函数的定义
};

#endif
```

- student.h

```
#ifndef STUDENT_H
#define STUDENT_H
```

```cpp
#include "person.h"
#include "date.h"

class student : public person
{
public:
    int id;
    char department[20];
    date enterdate;

public:
    student(void);
    void set_student(int n, char *s, int y, int m, int d);
    int get_id(void)
    {
        return id;
    } // 完成成员函数的定义
    char *get_department(void)
    {
        return department;
    } // 完成成员函数的定义
    int get_enteryear(void)
    {
        return enterdate.get_year();
    } // 完成成员函数的定义
    int get_entermonth(void)
    {
        return enterdate.get_month();
    } // 完成成员函数的定义
    int get_enterday(void)
    {
        return enterdate.get_day();
    } // 完成成员函数的定义
    void print(void);
    void print_base();
};

student::student(void)
{
    strcpy(name, "无名氏");
    strcpy(sex, "男");
    birthday.set_date(1980, 1, 1);
    id = 0;
    strcpy(department, "计算机");
    enterdate.set_date(2000, 9, 1);
}
void student::set_student(int n, char *s, int y, int m, int d)
// n、s、y、m、d分别为id、department、enterdate提供值
{
    id = n;
    strcpy(department, s);
    enterdate.set_date(y, m, d);
} // 完成成员函数的定义
```

```
void student::print(void)
{
    cout << "学号:" << id << endl;
    person::print();
    cout << "系(专业):" << department << endl;
    cout << "进校日期:" << enterdate.get_year() << "年";
    cout << enterdate.get_month() << "月";
    cout << enterdate.get_day() << "日" << endl;
}
void student::print_base()
{
    cout << setw(8) << get_id();
    cout << setw(12) << get_name();
    cout << setw(4) << get_sex();
    cout << setw(6) << get_year() << "-" << get_month() << "-" << get_day();
    cout << setw(20) << get_department();
    cout << setw(6) << get_enteryear() << "-" << get_entermonth();
    cout << "-" << get_enterday() << endl;
}

#endif
```

- score.h

```
#ifndef SCORE_H
#define SCORE_H

#include "student.h"
const int M = 10;
class score : public student
{
private:
    float sc[M], aver;
    int m;

public:
    score(void);                        // 无参构造函数
    void set_score(float x[], int n); // 提供成绩
    float get_score(int i)              // 得到第i科成绩
    {
        return sc[i];
    }                       // 完成成员函数的定义
    float get_aver(void) // 得到平均成绩
    {
        return aver;
    } // 完成成员函数的定义
    void print(void);
    void print_score(void);
};

score::score(void) // 无参构造函数
```

```
{
    strcpy(name, "无名氏");
    strcpy(sex, "男");
    birthday.set_date(1980, 1, 1);
    id = 0;
    strcpy(department, "计算机");
    enterdate.set_date(2000, 9, 1);
    int i;
    m = M;
    for (i = 0; i < m; i++)
        sc[i] = 0;
    aver = 0;
}
void score::set_score(float x[], int n) // 提供成绩:完成成员函数的定义
{
    int i;
    float sum = 0;
    m = n;
    for (i = 0; i < m; i++)
    {
        sc[i] = x[i];
        sum += x[i];
    }
    aver = sum / m;
}

void score::print(void) // 重载输出print()
{
    student::print();
    int i;
    for (i = 0; i < m; i++)
        cout << "  " << sc[i];
    cout << "  " << aver << endl;
}
void score::print_score(void)
{
    int j;
    cout << setw(8) << get_id();
    cout << setw(12) << get_name();
    for (j = 0; j < m; j++)
        cout << setw(6) << get_score(j);
    cout << " " << setw(6) << get_aver() << endl;
}

#endif
```

实验七 ——虚函数及应用

张林鹏_2021032449

一、实验目的

1. 理解虚函数与运行时 (动态) 多态性之间的关系, 掌握虚函数的定义及应用;
2. 理解纯虚函数与抽象类的概念, 掌握抽象类的定义及应用;
3. 理解虚析构函数的概念及作用.

二、实验内容

程序1: exp_701.cpp

1. 编译运行程序的输出结果是:

```
a=50  b=50
a=10  b=20
a=10  b=20  c=30
```

2. 根据程序的输出结果可知:

   i. 执行 `mp=&mb; mp->show();` 时, 调用的是 Base 类的 `show()`;
   ii. 执行 `mp=&md; mp->show();` 时, 调用的是 Base 类的 `show()`;
   iii. 执行 `((Derived*)mp)->show();` 时, 调用的是 Derived 类的 `shou()`;
   iv. 其中 `((Derived*)mp)` 是将 **Base** 类 `mp` 指针强制转换为 Derived 类的指针.

3. 在基类 `Base` 中的成员函数 `void show()` 改为 `virtual void show()`, 在重新编译运行程序, 输出结果为:

```
a=50  b=50
a=10  b=20
a=10  b=20  c=30
```

4. 当执行 `mp=&md; mp->show();` 时, 调用的是 Derived 类的 `show()`;

   函数 `show()` 成为 虚函数, 实现了 动态 多态性.

程序2: exp_702.cpp

5. 编译运行程序的输出结果为:

```
Base该类无计算
a=10
b=20
```

6. 将`Base`类中的`virtual void show()`改为`virtual void show()=0`, 重新编译程序会出现 编译错误, 其中的`show()`成为 纯 虚函数, 该类成为 抽象 类, 出错的原因是 该类不能被实例化,因为该类中存在纯虚函数.

7. 去掉`main()`函数中的`Base mb;`及`mp=&mb; mp->show();`, 重新编译运行程序的输出结果为:

```
a=10
b=20
```

程序3: exp_703.cpp

- 你分析的程序输出结果是:

```
The parent. version A
The derived 1 info:3 version 1
The derived 2 info:15 version A
```

- 程序运行实际输出结果为:

```
The parent. version A
The derived 1 info:3 version 1
The derived 2 info:15 version A
```

程序4: exp_704.cpp

- 你分析的程序输出结果是:

```
----base1----
----base2----
----derived----
----derived----
```

- 程序运行实际输出结果时:

```
----base1----
----base2----
----derived----
----derived----
```

程序5: exp_705.cpp

- 你分析的程序输出结果是:

```
Triangle with height 10 and base 6 has an area of30
Square with dimension 10 * 6 has an area of 60
Circle with radius 10 has an area of 314.159
```

- 程序实际运行时出结果是:

```
Triangle with height 10 and base 6 has an area of30
Square with dimension 10 * 6 has an area of 60
Circle with radius 10 has an area of 314.159
```

程序6: exp_706.cpp

8. 程序中:
   i. 处应改为: `virtual double area()=0;`;
   ii. 处应改为: `p=&ob1;`;
   iii. 处应改为: `p=&ob2;`;
   iv. 处应改为: `p=&ob3;`;

程序设计实验 (exp_707.cpp)

- exp_707.cpp:

```
#include <bits/stdc++.h>
using namespace std;

class process
{
public:
    virtual void print() = 0;
};
```

```
class person : public process
{
protected:
    string name;
    int age;
    string gender;

public:
    person(string n, int a, string g) : name(n), age(a), gender(g) {}
    void print()
    {
        cout << "name: " << name << endl;
        cout << "age: " << age << endl;
        cout << "gender: " << gender << endl;
    }
};

class student : public person
{
protected:
    string id;
    string major;

public:
    student(string n, int a, string g, string i, string m) : person(n, a,
g), id(i), major(m) {}
    void print()
    {
        person::print();
        cout << "id: " << id << endl;
        cout << "major: " << major << endl;
    }
};

int main()
{
    // 定义 process 指针变量调用 person 对象及 student 对象
    process *p;
    person p1("Tom", 20, "male");
    student s1("Jerry", 18, "male", "20210001", "Computer Science");
    p = &p1;
    p->print();
    cout << endl;
    p = &s1;
    p->print();
    return 0;
}
```

实验八 ——运算符重载及应用

张林鹏_2021032449

一、实验目的

1. 理解多态性概念及分类, 熟悉C++中静态多态性的实现方法;
2. 理解运算符重载的意义, 掌握在C++中用成员函数及友元函数实现运算符重载的方法;
3. 熟悉运算符重载的应用.

二、实验内容

程序1: exp_801.cpp

1. 程序编译运行的结果是:

```
x=10  y=20
x=11  y=21
```

2. 程序中前置运算符 `++` 采用的重载方法是 成员函数重载 .

3. 将重载函数 `coord coord::operator ++()` 中的 `++x; ++y;` 改为: `++this->x; ++this->y ;` , 重新运行程序:

```
x=10  y=20
x=11  y=21
```

4. 将 `main()` 函数中的 `++ob;` 改为 `“ob.operator++( );` , 重新运算程序:

```
x=10  y=20
x=11  y=21
```

程序2: exp_802.cpp

5. 编译程序时, 将会出现编译错误, 其原因是 重载运算符**+**应该是一个类的成员函数或者一个友元函数, 而在该程序中, 是一个普通函数 .

6. 将类中的 `complex operator+(complex c1,complex c2);` 的前面加上 `friend` , 重新编译调试程序, 输出结果为:

```
2.3+4.6i
3.6+2.8i
6+7.4i
```

7. 将 `main()` 中的 `A3=A1+A2;` 改为 `A3=operator(A1,A2);` , 重新运行程序, 输出为:

```
2.3+4.6i
3.6+2.8i
6+7.4i
```

程序3: exp_803.cpp

8.
```
#include <iostream>
using namespace std;
class complex
{
private:
    double real;
    double imag;

public:
    complex(double r = 0, double i = 0)
    {
        real = r;
        imag = i;
    }
    void print();
    complex operator+(const complex &c);
    complex operator-(complex c);
};
complex complex ::operator+(const complex &c) // 重载“+”
{
    complex temp;
    temp.real = this->real + c.real;
    temp.imag = this->imag + c.imag;
    return temp;
}
complex complex ::operator-(complex c) // 重载“-”
{
    complex temp;
```

```
    temp.real = this->real - c.real;
    temp.imag = this->imag - c.imag;
    return temp;
}
void complex::print()
{
    cout << real;
    if (imag > 0)
        cout << "+";
    if (imag != 0)
        cout << imag << "i" << endl;
}
int main()
{
    complex A1(2.3, 4.6), A2(3.6, 2.8), A3, A4;
    A3 = A1 + A2;
    A4 = A1 - A2;
    A1.print();
    A2.print();
    A3.print();
    A4.print();
    return 0;
}
```

程序4: exp_804.cpp

9.
```
#include <iostream>
using namespace std;

class coord
{
    int x, y;

public:
    coord(int i = 0, int j = 0);
    void print();
    coord operator++();
    coord operator++(int);
    coord operator-();
};
coord::coord(int i, int j)
{
    x = i;
    y = j;
}
void coord::print()
{
    cout << "x=" << x << "  y=" << y << endl;
}
```

```
coord coord::operator++() // 重载前置运算符++
{
    ++x;
    ++y;
    return *this;
}
coord coord::operator++(int) // 重载后置运算符++
{
    coord temp(*this);
    ++x;
    ++y;
    return temp;
}

coord coord::operator-() // 重载负号“-”
{
    coord temp(-x, -y);
    return temp;
}
int main()
{
    coord ob1(10, 20), ob2(20, 40), ob;
    ob1.print();
    ob2.print();
    ++ob1;
    ob2.operator++(0); // 显式调后后置运算符
    ob1.print();
    ob2.print();
    ob = -ob1;
    ob.print();
    return 0;
}
```

程序5: exp_805.cpp

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

class MyString
{
    char *ptr;
public :
    MyString(char *s)
    {
        ptr = new char[strlen(s) + 1];
        strcpy(ptr, s);
    }
    ~MyString()
```

```
    {
        delete [] ptr;
    }
    void print()
    {
        cout << ptr << endl;
    }
    MyString &operator=(const MyString &s);
};

MyString &MyString::operator=(const MyString &s) //重载“=”运算符
{
    if (this == &s)
        return *this;                    //当用“ob1=ob1;”时，直接返回
    delete[] ptr;                        //释放被赋值对象的空间
    ptr = new char[strlen(s.ptr) + 1];   //重新为被赋值对象分配空间
    strcpy(ptr, s.ptr);
    return *this;
}

int main()
{
    MyString p1("chen");
    {
        MyString p2("    ");
        p2 = p1;
        p2.print();
    }
    p1.print();
}
```

程序设计实验 (user_date.h & exp_806.cpp)

- user_date.h:

```
#ifndef USER_DATE_H
#define USER_DATE_H

#include "date.h"

class user_date : date
{
public:
    user_date(int y, int m, int d) : date()
    {
        set_date(y, m, d);
    }
    bool operator==(user_date &d);
    bool operator!=(user_date &d);
```

```
    bool operator>(user_date &d);
    bool operator<(user_date &d);
    bool operator>=(user_date &d);
    bool operator<=(user_date &d);
};

bool user_date::operator==(user_date &d)
{
    if (get_year() == d.get_year() && get_month() == d.get_month() &&
get_day() == d.get_day())
        return true;
    else
        return false;
}

bool user_date::operator!=(user_date &d)
{
    if (get_year() != d.get_year() || get_month() != d.get_month() ||
get_day() != d.get_day())
        return true;
    else
        return false;
}

bool user_date::operator>(user_date &d)
{
    if (get_year() > d.get_year())
        return true;
    else if (get_year() == d.get_year() && get_month() > d.get_month())
        return true;
    else if (get_year() == d.get_year() && get_month() == d.get_month() &&
get_day() > d.get_day())
        return true;
    else
        return false;
}

bool user_date::operator<(user_date &d)
{
    if (get_year() < d.get_year())
        return true;
    else if (get_year() == d.get_year() && get_month() < d.get_month())
        return true;
    else if (get_year() == d.get_year() && get_month() == d.get_month() &&
get_day() < d.get_day())
        return true;
    else
        return false;
}

bool user_date::operator>=(user_date &d)
{
    if (get_year() > d.get_year())
```

```
        return true;
    else if (get_year() == d.get_year() && get_month() > d.get_month())
        return true;
    else if (get_year() == d.get_year() && get_month() == d.get_month() &&
get_day() > d.get_day())
        return true;
    else if (get_year() == d.get_year() && get_month() == d.get_month() &&
get_day() == d.get_day())
        return true;
    else
        return false;
}

bool user_date::operator<=(user_date &d)
{
    if (get_year() < d.get_year())
        return true;
    else if (get_year() == d.get_year() && get_month() < d.get_month())
        return true;
    else if (get_year() == d.get_year() && get_month() == d.get_month() &&
get_day() < d.get_day())
        return true;
    else if (get_year() == d.get_year() && get_month() == d.get_month() &&
get_day() == d.get_day())
        return true;
    else
        return false;
}
#endif
```

- exp_806.cpp:

```
#include "date.h"
#include "user_date.h"
using namespace std;

int main()
{
    user_date u0(2020, 4, 1);
    user_date u1(2021, 3, 1);
    user_date u2(2021, 3, 1);
    user_date u3(2022, 3, 1);
    user_date u4(2022, 3, 6);

    cout << "u0 == u1: " << ((u0 == u1) ? "true" : "false") << endl;
    cout << "u1 == u2: " << ((u1 == u2) ? "true" : "false") << endl;
    cout << "u1 != u2: " << ((u1 != u2) ? "true" : "false")<< endl;
    cout << "u2 < u3:  " << ((u2 < u3) ? "true" : "false") << endl;
    cout << "u3 < u4:  " << ((u3 < u4) ? "true" : "false") << endl;
}
```