day06

一、结构体在C语言在的操作

结构体和类包括函数一样，但是，结构体里面的函数都是公开的

二、友元函数

类当中的方法默认是私有的

```
#include <iostream>
using namespace std;
class car{
    string color = "黑色";
    // 标记modifyCar是car的好朋友
    friend void modifyCar(car & cc);
public:
    string getColor(){
        return color;
    }
};
void modifyCar(car & cc){
    cc.color = "白色";
}
int main() {
    car c;
    cout << c.getColor() << endl;
    // 调用友元函数
    modifyCar(c);
    cout << c.getColor() << endl;
    return 0;
```

```
}
// 运行结果:
黑色
白色
```

友元类

```
#include <iostream>
using namespace std;
class Car{
    string color = "黑色";
public:
    string getColor(){
        return color;
    }
    friend class SSSS;
};
class SSSS{
public:
    void modify(Car &c){
        c.color = "红色";
    }
};
int main() {
    Car c;
    cout << c.getColor() << endl;
    SSSS s;
    s.modify(c);
    cout << c.getColor() << endl;
    return 0;
}
// 运行结果:
黑色
红色
```

三、运算符重载

1.运算符重载意义

运算符重载，就是为了，让运算符有更多的特征，可以让运算符匹配更多的类型，让代码看起来更加优雅

运算符重载有两种方式，一种是成员函数，一直是全局函数

2.运算符重载—成员函数定义

```
// 运算符重载
    // s1 + s2 谁在前面，就是谁调用的这个运算符重载的函数，谁在后面，谁就是参数
返回值类型 operator符号(后一个对象){
    return 返回值;
}
```

```
class stu{
public:
    int money;
    stu(int money):money(money){

    }
    // 运算符重载
    // s1 + s2 谁在前面，就是谁调用的这个运算符重载的函数，谁在后面，谁就是参数
    int operator+(stu & s){
        return this->money + s.money;
    }
};
int main() {
    stu s1(10);
    stu s2(20);
    int a = s1 + s2;
    cout << a << endl;
    return 0;
}
// 运行结果:
30
```

用引用，防止数据复制，增加内存，提高运行效率

3.运算符重载—全局函数方式

在类中时，只需要传一个参数，当为全局时，需要传两个参数

```
int operator+(stu s1,stu s2){
    return s1.money + s2.money;
}
```

4.运算符重载—输出运算符重载<<

```
class stu{
public:
    string name;
    int age;
    stu(){
    }
    stu(string name,int age):name(name),age(age){}
};
// ostream 中将拷贝禁用了，所以必须要用引用
void operator<<(ostream& o,stu& s1){
    cout << s1.name << ":" << s1.age << endl;
```

```
}
int main() {

    stu s2("李四",20);
    cout <<s2;
    return 0;
}
// 运行结果:
李四:20
```

```
class stu{
public:
    string name;
    int age;
    stu(){
    }
    stu(string name,int age):name(name),age(age){}
};
// ostream 中将拷贝禁用了，所以必须要用引用
ostream& operator<<(ostream& o,stu& s1){
    cout << s1.name << ":" << s1.age << endl;
    return o;
}
int main() {
    stu s1("张三",18);
    stu s2("李四",20);
    cout << s1 << s2;
    return 0;
}
// 运行结果:
张三:18
李四:20
```

5.运算符重载—输入运算符重载

```
class stu{
public:
    string name;
    int age;
    stu(){
    }
    stu(string name,int age):name(name),age(age){}
};
// >> 输出运算符重载
void operator>>(istream& in,stu& s1){
    cout << "请输入你的姓名:" << endl;
    in >> s1.name;
    cout << "请输入你的年龄:" << endl;
    in >> s1.age;
}
```

```
int main() {
    stu s1("张三",18);
    cin >> s1;
    cout << s1.name << endl;
    cout << s1.age << endl;
    return 0;
}
// 运行结果:
请输入你的姓名:
李四
请输入你的年龄:
25
李四
25
```

补

1. 赋值运算符

```
stu s1("张三",18);
stu s2("李四",20); // 走的是有参构造
stu s3 = s1;    // 走的是拷贝构造
s2 = s1; // 此处用了拷贝赋值运算符
```

默认情况下，类里面有赋值运算符

2. 调用运算符()

四、lambda表达式

1.lambda表达式的介绍

提高代码运行效率，但是阅读性差，理解难度高

实现方法

```
// 匿名函数
[]()->int{};
```

`[]`: **必须要有这是捕获列表**

`()`: **表示函数参数，如果没有参数，小括号可以不用写**

`->`: **连接函数体，可以不写，会推断出来**

`int`: **函数的返回值，可以不写，会推断出来**

`{}`: **函数体**

```
[]()->void{cout << "你好" << endl;};
[]{
    cout << "你好" << endl;
};
```

2.lambda的调用

1. 定义函数指针来接收

```
// 定义函数指针来接收
void (*p)() = []{cout << "你好" << endl;};
```

2. 使用auto来调用

```
auto a = []{cout << "你好,lamdba" << endl;};
```

3. 使用调用运算符 `()`

```
[]{cout << "你好,lamdba111" << endl;}();
```

使用

```
int c = [](int a,int b){return a+b;}(10,20);
cout << c << endl;
// 运行结果:
30
```