

# C++第四天课堂笔记

## 一、回顾知识点

### 1.1 构造函数的作用与分类

构造函数：用于初始化对象的数据，在新对象创建存储空间之后，自动调用的。

构造函数的分类：

- 1) 有参与无参的构造函数
- 2) 普通的构造函数与拷贝构造函数

拷贝构造函数，用于类的对象为右值时使用，也就是说将一个对象赋值给另一个类定义的对象时，会调用拷贝构造函数。

```
A(const A &obj);
```

普通的构造函数，可以重载（即，可以存在多个构造函数，只不过参数列表不同）

### 1.2 析构函数的作用

析构函数的作用：回收资源，并释放空间

当删除（释放）对象时，会执行析构函数。

一个类中只能存在一个析构函数。 `~A();`

### 1.3 默认的构造与析构函数

编译器默认为类提供的函数有：

- 1) 无参的构造函数
- 2) 拷贝构造函数
- 3) 析构函数

### 1.4 类对象创建方式

即调用构造函数的方式创建对象，对象的空间存在栈区的。

- 1) `A a;`
- 2) `A a(参数列表)`
- 3) `A a = 常量值;` // 将右值隐式转化为 `A(常量值)`，可以`explicit`关键可以禁止隐式转化
- 4) `A(参数列表);` // 匿名对象的创建方式
- 5) `A a = A(参数列表)`

## 1.5 动态创建对象

动态创建的对象的空间是在堆区。

使用new创建和delete释放

```
A *a = new A(参数列表); // 创建A类对象的空间，并调用构造函数进行初始化数据
delete a; // 调用a对象的析构函数，释放a对象的空间
```

使用malloc创建和free释放【不建议】

需要手动调用初始化函数和析构函数

```
A *a = (A *)malloc(sizeof(A));
a->init(参数列表);

a->clean(); // 回收对象的成员变量的资源
a->~A();
free(a);
```

使用new创建数组和delete删除对象数组

```
A *p = new A[N]; // 默认调用A类的无参构造函数创建N个对象
A *p = new A[N]{A(参数列表), ...} // 指定构造函数创建N个类对象

delete[] p; // new 时带有[], delete时也要带[], 表示创建时指定的对象个数，删除时也要释放对应个数的对象。
```

## 二、类的静态成员

类的成员声明时，前面（最左边）可以使用static关键字，将其声明为类的静态成员。

静态成员：静态成员变量和静态成员函数。

【注意】

类的静态成员属于类的，不占类对象的空间。是所有类对象共享的。

类的静态成员函数，只能访问静态成员变量。

### 2.1 类的静态成员变量的声明、初始化与访问

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

class A
{
public:
    static int x;
};

int A::x = 20; // 静态变量初始化，静态成员变量初始化时不需要static
int main()
{
```

```

A a1, a2;
// 访问类的静态成员: 1) A::x, 2) 对象名.x
cout << "A::x=" << A::x << endl;
A::x = 30;
cout << "a1.x=" << a1.x << endl;
cout << "a2.x=" << a2.x << endl;

return 0;
}

```

```

disen@qfxa:~/code2/day04$ ./a.out
A::x=20
a1.x=30
a2.x=30

```

## 2.2 静态成员函数，内部只能访问静态成员变量

```

#include <iostream>
using namespace std;

class A
{
public:
    int m;
    static int x;
    static void add(int n)
    {
        // m = 10; // error, 静态成员函数不能访问非静态成员
        x += n;
        cout << "static x=" << x << endl;
    }
};

int A::x = 20; // 静态变量初始化
int main()
{
    A a1;
    A::add(10);
    cout << "a1.x=" << a1.x << endl;

    return 0;
}

```

```

disen@qfxa:~/code2/day04$ ./a.out
static x=30
a1.x=30

```

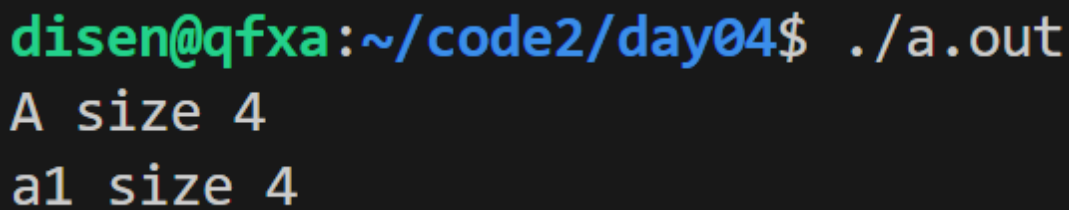
## 2.3 静态成员变量不占类对象的空间

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

class A
{
public:
    int m;
    static int x;
    A(int m)
    {
        this->m = m;
    }
};

// int A::x = 20; // 静态变量初始化
int main()
{
    A a1(3);
    cout << "A size " << sizeof(A) << endl;
    cout << "a1 size " << sizeof(a1) << endl;

    return 0;
}
```



```
disen@qfxa:~/code2/day04$ ./a.out
A size 4
a1 size 4
```

## 2.4 类的单例设计模式

私有化构造函数和拷贝函数，提供一个public的静态函数返回类的指针对象，声明一个静态的类的指针对象。

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

class A // 单例的类：类的对象运行期间有且只有一个对象。
{
private:
    int x;
    A(int x) { this->x = x; }
    A(const A &other)
    {
        x = other.x;
    }

public:
    static A *getInstance(int x = 0)
```

```

{
    if (instance == NULL)
    {
        instance = new A(x);
    }
    return instance;
}
void show()
{
    cout << "x=" << x << endl;
}

public:
    static A *instance;
};

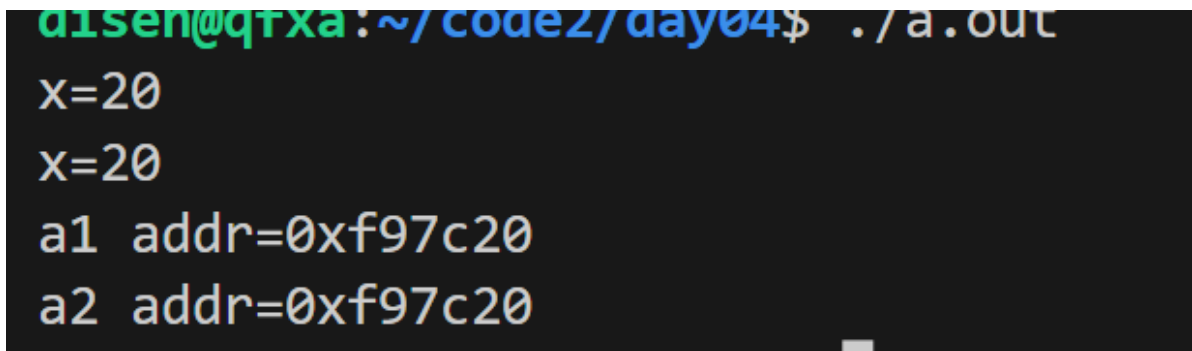
A *A::instance = NULL; // 定义初始化，在静态全局区

int main()
{
    A *a1 = A::getInstance(20);
    a1->show();

    A *a2 = A::getInstance();
    a2->show();
    cout << "a1 addr=" << a1 << endl;
    cout << "a2 addr=" << a2 << endl;

    delete A::instance; // 释放单例对象的空间
    return 0;
}

```



```

disen@qtxa:~/code2/day04$ ./a.out
x=20
x=20
a1 addr=0xf97c20
a2 addr=0xf97c20

```

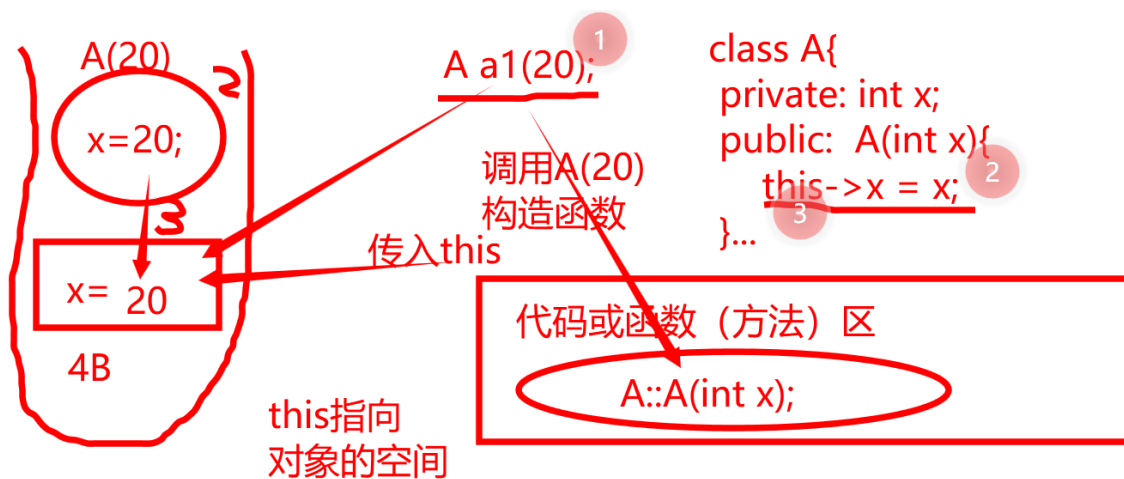
## 三、对象的存储

### 3.1 类的成员变量与函数的存储

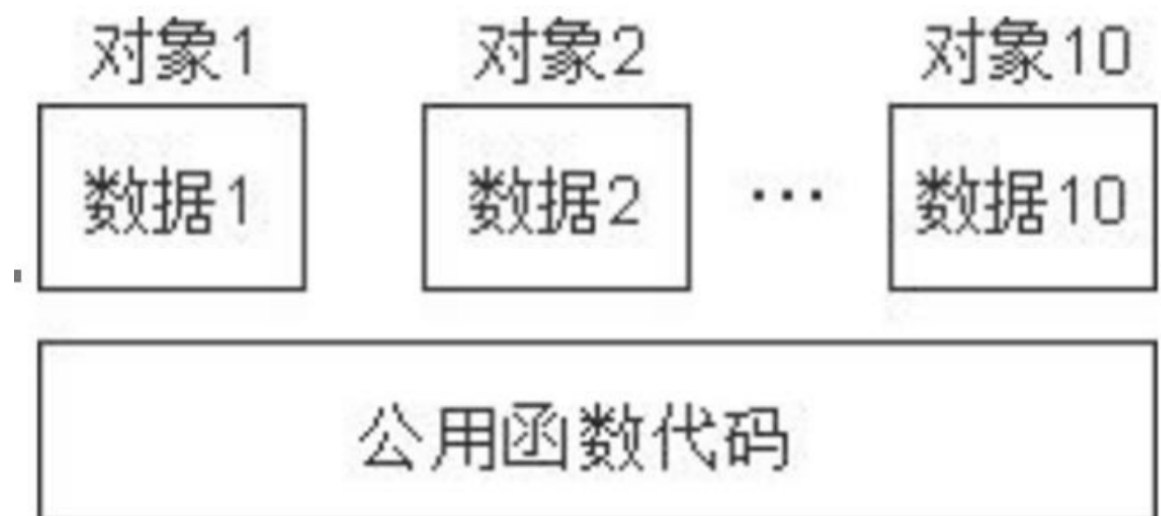
类中的非静态成员存储与类的对象存在一起的，类的成员函数存储在类对象（代码区）。

一个类的大小由类的非静态成员决定的。

当前调用成员函数时，从类存储的位置（代码区）中调入的栈中，为函数的形参变量分配空间（局部变量），同时也会将调用成员函数的对象封装成当前类的指针，这个指针即为this。



### 3.2 this指针



当某一个对象调用成员函数时，将对象转化为 this 指针，传入到成员函数中，以区分另一个对象调用的成员函数，方便操作自己对象的成员变量。当形参和成员变量同名时，也可用 this 指针来区分。

在类的非静态成员函数中返回对象本身，可使用 return \*this，可以通过这种实现对象的链式编程 (a1.a().b().c()....;)，如cout 属于链式编程风格 (cout << "" << "" << " ")。

如：

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

class A
{
private:
    int a, b, c;

public:
    void setA(int a)
    {
        this->a = a;
    }
    void setB(int b)
    {
        this->b = b;
    }
}
```

```

void setC(int c)
{
    this->c = c;
}
void show()
{
    cout << "a=" << a << ",b=" << b << ",c=" << c << endl;
}
};

class ABuilder
{
public:
    ABuilder()
    {
        instance = new A();
    }
    ~ABuilder()
    {
        if (instance != NULL)
            delete instance;
    }
    ABuilder &a(int a)
    {
        instance->setA(a);
        return *this;
    }
    ABuilder &b(int b)
    {
        instance->setB(b);
        return *this;
    }
    ABuilder &c(int c)
    {
        instance->setC(c);
        return *this;
    }
    A *build()
    {
        return instance;
    }

private:
    A *instance;
};

int main()
{
    // 构建器设计模式： 构造产品，需要构造器一步一步的完成构造，最后build()出产品
    ABuilder builder;
    A *a1 = builder.a(1).b(2).c(3).build();
    a1->show();
    return 0;
}

```

```
disen@qfxa:~/code2/day04$ ./a.out
a=1,b=2,c=3
```

### 3.3 const修饰成员函数与类对象

当const修饰成员函数时，函数内不能修改普通成员变量，但可以修改mutable修饰的成员变量。

当const修饰类对象时，只能读成员变量，不能修改成员变量的值。可以调用const修饰的成员函数，不能调用普通成员函数。

```
#include <iostream>

using namespace std;
class A
{
public:
    int x;
    mutable int y;

    void setXY(int x, int y) const
    {
        // this->x = x; // error
        this->y = y;
    }
    void show()
    {
        cout << x << "," << y << endl;
    }
};

int main(int argc, char const *argv[])
{
    A a1;
    a1.setXY(1, 2); // const函数，内部不修改非mutable的成员变量
    a1.show();

    // const A a2; 报错 error，const 变量必须存在初始值（右值）
    const A a2 = A();
    a2.setXY(0, 30); // const对象可以访问const的成员函数
    // a2.show(); // const对象不能访问非const成员函数
    return 0;
}
```

```
disen@qfxa:~/code2/day04$ ./a.out
4196944,2
```

## 四、友元



## 4.1 友元语法

使用 friend关键字，可以修饰 其他类、类成员函数、全局函数都可声明为友元

【注意】

友元函数不是类的成员，不带 `this` 指针  
友元函数可访问对象任意成员属性，包括私有属性

## 4.2 全局友元函数

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

class B
{
    // 声明全局函数的友元
    friend void show(B &b); // 将全局的show()函数设置为当前类的友元函数
private:
    int x;

public:
    B(int x)
    {
        this->x = x;
    }
    void show()
    {
        cout << x << endl;
    }
};

// 全局的友元函数
void show(B &b)
{
    // b的私有成员
    cout << "b.x=" << b.x << endl;
}

int main()
{
    B b(100);
    show(b);
    return 0;
}
```

## 4.3 友元的类成员函数

【注意】类的某一个成员函数作为友元函数时，不能在类定义，只能声明。

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

class B;
class C
```

```

{
public:
    // 在此函数内，要访问B类中所有成员，将此函数在B类中声明友元
    // 先声明，不能实现
    void showB(B &b);
};

class B
{
    // 声明友元的成员函数
    friend void C::showB(B &b);

private:
    int x;

public:
    B(int x)
    {
        this->x = x;
    }
    void show()
    {
        cout << x << endl;
    }
};

// 定义或实现友元的成员函数
void C::showB(B &b)
{
    cout << b.x << endl;
}

int main()
{
    B b(100);
    C c;
    c.showB(b);
    return 0;
}

```

## 4.4 友元类

```

#include <iostream>
using namespace std;

class B;
class C
{
public:
    void showB(B &b);
};

class B
{
    friend class C;
}

```

```

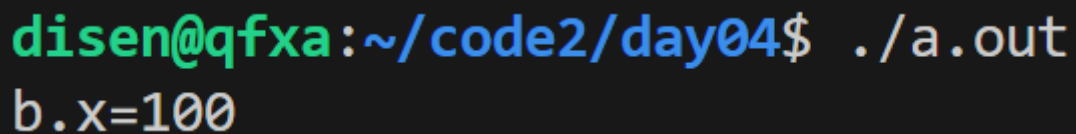
private:
    int x;

public:
    B(int x)
    {
        this->x = x;
    }
    void show()
    {
        cout << x << endl;
    }
};

// 定义或实现友元的成员函数
void C::showB(B &b)
{
    cout << "b.x=" << b.x << endl;
}

int main()
{
    B b(100);
    C c;
    c.showB(b);
    return 0;
}

```



```

disen@qfxa:~/code2/day04$ ./a.out
b.x=100

```

## 4.5 课堂练习

请编写电视机类，电视机有开机和关机状态，有音量，有频道，提供音量操作的方法，频道操作的方法。由于电视机只能逐一调整频道，不能指定频道，增加遥控类，遥控类除了拥有电视机已有的功能，再增加根据输入调台功能。

电视类与遥控器类：

```

// 声明遥控器类
class Remoter;

// 定义电视类
class Tv
{
    friend class Remoter;
    enum
    {
        OFF,
        ON
    };
    enum

```

```

{
    MIN_V,
    MAX_V = 100
};
enum
{
    MIN_CH = 1,
    MAX_CH = 255
};

private:
    int mStatus; // 电视状态
    int mVolume; // 声音
    int mChannel; // 频道
public:
    Tv()
    {
        mStatus = OFF; // 状态
        mVolume = 10; // 声音
        mChannel = MIN_CH; // 频道
    }

    void power() // 电源按钮
    {
        mStatus = mStatus ? OFF : ON;
    }

    void volumeUp()
    {
        if (mVolume != MAX_V)
            mVolume++;
        showInfo();
    }

    void volumeDown()
    {
        if (mVolume != MIN_V)
            mVolume--;
        showInfo();
    }

    void channelUp()
    {
        if (mChannel != MAX_CH)
            mChannel++;
        showInfo();
    }

    void channelDown()
    {
        if (mChannel != MIN_CH)
            mChannel--;
        showInfo();
    }

    void showInfo()
    {
        if (mStatus)
        {
            cout << "Status ON, Volume " << mVolume << ", Channel " << mChannel
<< endl;

```

```

    }
}
};

// 定义遥控器类
class Remoter
{
private:
    TV *mTv;

public:
    Remoter(TV *tv)
    {
        mTv = tv;
    }
    void power()
    {
        mTv->power();
    }
    void vUp()
    {
        mTv->volumeUp();
    }
    void vDown()
    {
        mTv->volumeDown();
    }
    void chUp()
    {
        mTv->channelUp();
    }
    void chDown()
    {
        mTv->channelDown();
    }
    void setChannel(int ch) // 指定频道
    {
        if (ch >= mTv->MIN_CH && ch <= mTv->MAX_CH)
        {
            mTv->mChannel = ch;
            mTv->showInfo();
        }
    }
};

```

测试1: t1()

```

void t1()
{
    // 电视类对象的操作
    TV tv;
    tv.power();
    tv.volumeUp();
    tv.volumeUp();
    tv.channelUp();
    tv.channelUp();
}

```

```

int main(int argc, char const *argv[])
{
    t1();
    return 0;
}

```

```

disen@qfxa:~/code2/day04$ ./a.out
Status ON, Volume 11, Channel 1
Status ON, Volume 12, Channel 1
Status ON, Volume 12, Channel 2
Status ON, Volume 12, Channel 3

```

测试2: t2()

```

void t2()
{
    TV *tv = new Tv();
    Remoter remoter(tv);
    remoter.power();
    remoter.setChannel(25);
    remoter.vUp();
    remoter.vUp();
    remoter.vUp();
    remoter.chDown();
    remoter.power();
    delete tv;
}

int main(int argc, char const *argv[])
{
    t2();
    return 0;
}

```

```

disen@qfxa:~/code2/day04$ ./a.out
Status ON, Volume 10, Channel 25
Status ON, Volume 11, Channel 25
Status ON, Volume 12, Channel 25
Status ON, Volume 13, Channel 25
Status ON, Volume 13, Channel 24

```

## 五、运算符重载

### 5.1 运算符重载基本概念

运算符重载即对运算的功能重新定义，从而使得运算符支持不同的数据类型。

运算符重载(operator overloading)只是一种“语法上的方便”，它是另一种函数调用的方式。

运算符重载的函数名字由关键字 `operator+运算符` 组成。它像任何其他函数一样，当编译器遇到适当的模式时，就会调用这个函数。

定义运算符重载函数时，依据运算符类别（单目、双目），定义函数的参数。

运算符重载函数是全局函数时，单目运算符需要一个参数，双目需要两个。

运算符重载函数是类成员函数时，单目运算符不需要参数（但++、-- 在后需要int占位参数），双目需要一个参数。

如：类成员函数的方式，定义运算符 + 重载函数

```
#include <iostream>

using namespace std;
class A
{
private:
    int x;

public:
    explicit A(int x)
    {
        this->x = x;
    }
    void show()
    {
        cout << x << endl;
    }
    // +运算符重载， 类成员函数， 双目运算符， 只一个参数
    A *operator+(A &other);
};

A *A::operator+(A &other)
{
    // this->x += other.x;
    A *tmp = new A(this->x + other.x);
    return tmp;
}

int main(int argc, char const *argv[])
{
    A a1(5), a2(6);
    a1.show();
    A *a3 = a1 + a2;
    a3->show();
    delete a3;
    return 0;
}
```

```
disen@qfxa:~/code2/day04$ ./a.out
5
11
```

如：++的运算符重载的成员函数

```
#include <iostream>

using namespace std;
class A
{
private:
    int x;

public:
    explicit A(int x)
    {
        this->x = x;
    }
    void show()
    {
        cout << x << endl;
    }
    // ++运算符重载， 类成员函数，单目运算符，++在前不要参数，++需要一个占位参数
    // ++a
    A &operator++()
    {
        ++this->x;
        return *this;
    }

    // a++
    A &operator++(int)
    {
        this->x++;
        return *this;
    }
};

int main(int argc, char const *argv[])
{
    A a1(5);
    a1.show();
    ++a1; // ++a1;
    a1.show();
    a1++;
    a1.show();
    return 0;
}
```



```
disen@qfxa:~/code2/day04$ ./a.out
5
6
7
```

## 5.2 运算符重载碰上友元函数

友元函数是一个全局函数时，和我们之前写的全局函数类似，只是友元函数可以访问某个类私有数据。

如：重载 c++ 的 << 输出流运算符

标准输出流（控制台）： ostream, cout 是 ostream 类的对象

标准输入流（键盘）： istream, cin 是 istream 类的对象

如：

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

class A{
    friend ostream & operator<<(ostream &cout, A &a);
private:
    int x,y;
public:
    A(int x, int y){
        this->x = x;
        this->y = y;
    }
};

// << 输出运算符重载
ostream & operator<<(ostream &cout, A &a){
    // x,y是A类的私有成员
    cout << a.x << ", " << a.y;
    return cout;
}

int main(){
    A a1(1, 2);
    cout << a1 << endl;
    return 0;
}
```

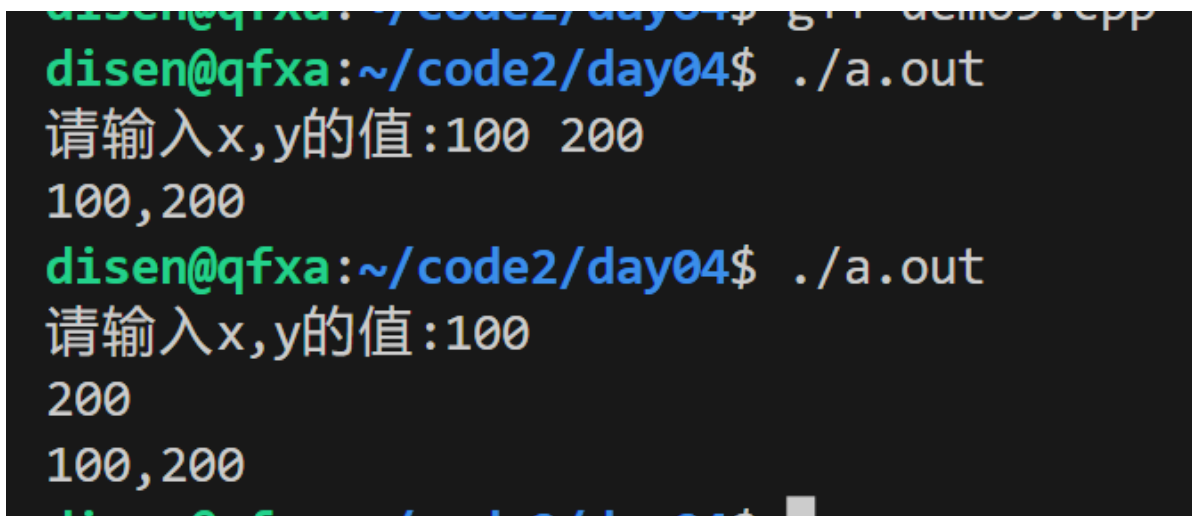
```
disen@qfxa:~/code2/day04$ ./a.out
1, 2
```

如：全局函数方式，实现 >> 输入流的重载函数

```
#include <iostream>

using namespace std;

int main(int argc, char const *argv[])
{
    int x, y;
    cout << "请输入x,y的值:";
    cin >> x >> y; // 从键盘接收x,y的变量的值，两个变量之间输入时，可以使用空格或回车分隔
    cout << x << "," << y << endl;
    return 0;
}
```



```
disen@qfxa:~/code2/day04$ g++ demos.cpp
disen@qfxa:~/code2/day04$ ./a.out
请输入x,y的值:100 200
100,200
disen@qfxa:~/code2/day04$ ./a.out
请输入x,y的值:100
200
100,200
```

```
#include <iostream>

using namespace std;
class Worker
{
    // 全局函数的友元
    friend ostream &operator<<(ostream &cout, Worker &obj);
    friend istream &operator>>(istream &cin, Worker &obj);

private:
    string name;
    int salary;
};

ostream &operator<<(ostream &cout, Worker &obj)
{
    cout << "Worker name is " << obj.name << ", salary is " << obj.salary;
    return cout;
}

istream &operator>>(istream &cin, Worker &obj)
{
    cout << "Name:";
    cin >> obj.name;
    cout << "Salary:";
    cin >> obj.salary;
    return cin;
}
```

```

}

int main(int argc, char const *argv[])
{
    worker w1;
    cin >> w1;
    cout << w1 << endl;
    return 0;
}

```

```

Name:小马
Salary:900000
Worker name is 小马, salary is 900000

```

## 5.3 可重载的运算符

可以重载的操作符

+	-	*	/	%	^	&		~
!	=	<	>	+=	-=	*=	/=	%=
^=	&=	=	<<	>>	>>=	<<=	=	!=
<=	>=	&&		++	--	->*	'	->
[]	()	new	delete	new[]	delete[]			

不能重载的算符

. :: .\* ?: sizeof

如：>或<关系 运算符的重载

```

#include <iostream>
#include <cstdlib>

using namespace std;

class Num
{
private:
    int n;

public:
    explicit Num(int n)
    {
        this->n = n;
    }
    bool operator>(int other) // 成员函数的运算符重载
    {

```

```

        return this->n > other;
    }
    bool operator>(Num &other)
    {
        return this->n > other.n;
    }
    bool operator<(Num &other)
    {
        return this->n < other.n;
    }
    bool operator<(int other)
    {
        return this->n < other;
    }
};
int main(int argc, char const *argv[])
{
    Num n1(atoi(argv[1])), n2(atoi(argv[2]));
    if (n1 > 20)
    {
        cout << "n1 大于20" << endl;
        if (n1 < n2)
        {
            cout << "n1 小于 n2" << endl;
        }
    }
    if (n1 > n2)
    {
        cout << "n1 大于 n2" << endl;
    }
    return 0;
}

```

```

Segmentation fault (core dumped)
disen@qfxa:~/code2/day04$ ./a.out 30 10
n1 大于20
n1 大于 n2
disen@qfxa:~/code2/day04$ ./a.out 10 30
disen@qfxa:~/code2/day04$ ./a.out 50 30
n1 大于20
n1 大于 n2

```

## 5.4 自增自减(++/--)运算符重载

++或--在前的重载函数，成员函数不需要参数：

```

类名 &operator++(){}
类名 &operator--(){}

```

++或--在后的重载函数，成员函数需要int占位参数：

```
类名 &operator++(int){}
类名 &operator--(int){}
```

案例：见5.1。

## 5.5 指针运算符(\*、->)重载

如果对一个对象取指针时，要实现->重载函数，函数内部返回引用的其它类对象的指针。

如果对一个对象取值\*时，要实现\*重载函数，函数内部返回引用其它类对象指针的值。

```
#include <iostream>

using namespace std;

class A
{
private:
    int x;

public:
    explicit A(int x) { this->x = x; }
    void printA()
    {
        cout << "x = " << x << endl;
    }
};

class B
{
private:
    A *mA;

public:
    B(int x)
    {
        mA = new A(x);
    }
    A *operator->()
    {
        return mA;
    }
    A &operator*()
    {
        return *mA;
    }
};

int main(int argc, char const *argv[])
{
    B b(20);
    // b-> 执行B类的->重载函数，返回A类的对象的指针
    b->printA();
    // *b 执行B类的*重载函数，返回 A类的对象
    (*b).printA();
    return 0;
}
```

```
disen@qfxa:~/code2/day04$ ./a.out  
x = 20  
x = 20
```

【注意】\*和->运算符的重载函数不能有参数。