中山大學
SUN YAT-SEN UNIVERSITY

运算符重载（上）

中山大学计算机学院

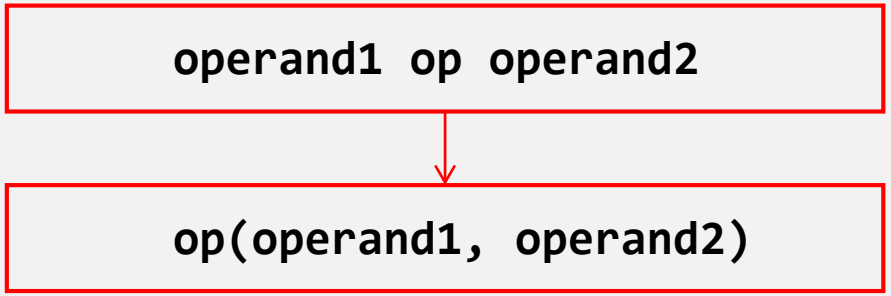
主讲人：XXX

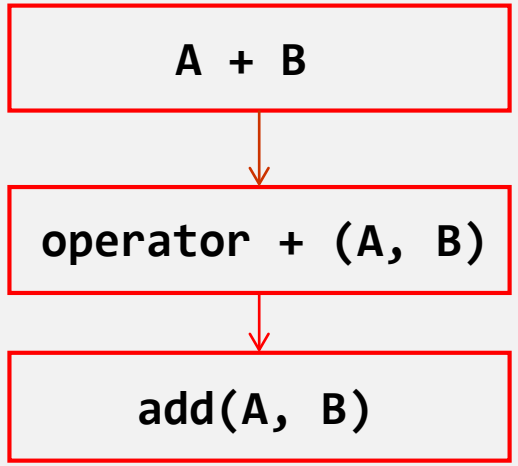
中山大学MOOC课程组

运算符重载

C++为了增强代码的可读性引入了运算符重载，运算符重载是具有特殊函数名的函数，也具有返回值类型，函数名和参数列表。

重载的运算符可以理解为带有特殊名称的函数，函数名是由关键字 operator 和其后要重载的运算符符号构成的。



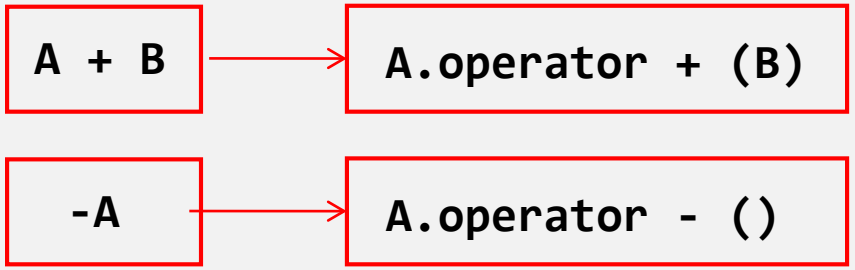


运算符重载的方法

运算符重载的两种方法：

- 类成员运算符重载
  - `return_type class_name::operator op([operand2]) {}`
  - 重载二元运算符时，成员运算符函数只需显式传递一个参数（即二元运算符的右操作数），而左操作数则是该类对象本身，通过 this 指针隐式传递。
  - 重载一元运算符时，成员运算符函数没有参数，操作数是该类对象本身，通过 this 指针隐式传递

- 友元运算符重载
  - `return_type operator op(operand1, operand2) {}`
  - 后面会详细说明



可以重载的运算符

运算符	操作符
双目算术运算符	+ (加)，-(减)，*(乘)，/(除)，% (取模)
关系运算符	==(等于)，!= (不等于)， < (小于)，> (大于)， <=(小于等于)，>=(大于等于)
逻辑运算符	||(逻辑或)，&&(逻辑与)，!(逻辑非)
单目运算符	+ (正)，-(负)，*(指针)，&(取地址)
自增自减运算符	++(自增)，--(自减)
位运算符	| (按位或)，& (按位与)， ~(按位取反)，^(按位异或)， << (左移)，>>(右移)
赋值运算符	=, +=, -=, *=, /= , % = , &=, |=, ^=, <<=, >>=
空间申请与释放	new, delete, new[ ] , delete[]
其他运算符	()(函数调用)，->(成员访问)，(逗号)，[](下标)

不可以重载的运算符

.：成员访问运算符

.*, ->*：成员指针访问运算符

::：域运算符

?:：条件运算符

sizeof：长度运算符　（包括除 new，delete 外的关键字运算符，如 alignof，typeid 等等）

#：　预处理符号

重载运算符的一些规则

- 不能创建新运算符，例如 **、<> 或 &|。
- 重载运算符不能改变运算符的优先级与结合性或操作数个数
- 重载运算符不可使用缺省参数
- 运算符=、()、[]和->可作为类成员运算符，但不可作为友元运算符。
- 以成员函数进行重载的运算符，使用时左操作数必须是类对象
- 除了new，new[],delete，delete[],不能重载没有类对象为操作数的运算符，也就是：要重载运算符，至少其中的一个操作数必须是类对象。
- 运算符 && 与 || 的重载失去短路求值。
- 重载的运算符 -> 必须要么返回裸指针，要么（按引用或值）返回同样重载了运算符 -> 的对象

例子

```
class Integer {
    int x;
public:
//转换构造函数是指带有一个参数的构造函数，可以将非本类对象的参数转换成本类对象
    Integer(int x=0):x(x)  { //包含了2个构造函数，默认构造函数和转换构造函数 }
    Integer operator + (const Integer& Int) {
        return Integer(x+Int.x);
    }
    Integer operator - (const Integer& Int) {
        return Integer(x-Int.x);      }
    Integer operator - () {
        return Integer(-x);           }
    void print() {
        cout<<x<<endl;                }
};
```

```
int main()  {
    Integer a=3,b=4;
    a.print();
    b.print();
    Integer c=a+b;
    c.print();
    Integer d=a-b;
    d.print();
    Integer e=-a;
    e.print();
    Integer f= a + 3;
// 相当于 f = a.operator+(3),3通过调用
//转换构造函数变成了一个Integer对象
    f.print();
    return 0;
}
```

输出结果：
3
4
7
-1
-3
6

自增运算符 重载

重载自增运算符时需注意：

- 若为前缀自增运算符，直接重载（以++举例）：
  - `return_type& class_name::operator ++() {}`
  - 前缀自增运算符以引用方式返回自增之后的对象本身
- 若为后缀自增运算符，该函数有一个 int 类型的虚拟形参，这个形参在函数的主体中是不会被使用的，这只是一个约定，它告诉编译器递增运算符正在后缀模式下被重载：
  - `return_type class_name::operator ++(int) {}`
  - 后缀自增运算符以值的方式返回自增之前的对象副本

自增运算符 重载

```
class Integer {
    int x;
public:
    Integer(int x=0):x(x)  {
    }
    Integer& operator ++ () { //前加
        cerr<<"prefix is invoked"<<endl;
        ++x;//这个++调用的是谁?
        return *this; //返回的是加了之后的对象本身
    }
    Integer operator ++ (int) {//后加
        Integer temp = *this;
        x++; //这个++调用的是谁?
        cerr<<"suffix is invoked"<<endl;
        return temp; //返回的是未加之前的对象副本
    }
    void print()   {
        cout<<x<<endl;    }
};
```

```
int main()  {
    int x = 0;
  //以下演示的是：++x这个表达式的值就是自增之后的x
  //本身，所以可以放在运算符的左边
   //仅仅为了演示前加的返回值，并不鼓励这样使用
    ++x = 10000;
    //x++ = 10000;  //会报错，因为x++表达式的值是未加
//之前的x的值
            cout << "普通变量 x = " << x << endl;

    Integer a=3;
    a.print();
    Integer b=++a;
    a.print();        b.print();
    Integer c=a++;
    a.print();        c.print();
    ++a = 10000; //由于前加返回的是对象本身，所以可以
//放在赋值运算符左边，不要这样用，只是为了明白概念
    a.print();
}
```

自增运算符 重载

输出:

普通变量 x = 10000
3
prefix is invoked
4
4
suffix is invoked
5
4
prefix is invoked
10000

自增运算符 重载

```
#include <iostream>
#include <iomanip>
using namespace std;
class Clock{
public:
    Clock(int =12,int=0,int=0);
    Clock& tick();
    friend ostream&
operator<<(ostream&,const Clock&);
    Clock& operator++();  //前加
    Clock  operator++(int); //后加
private:
    int hour;
   int min;
   int ap; // 0 is AM, 1 is PM
};
```

```
Clock& Clock::tick(){
    ++min; //调用的是?
   if(min == 60){
        hour++; //调用的是?
        min = 0;  }
       if(hour == 13)
            hour = 1;
      if(hour ==12 && min == 0)
            ap = !ap;
      return *this;   }
Clock& Clock::operator++(){
            return tick();  }
Clock  Clock::operator++(int i){
            Clock temp = *this;
            tick();
            return temp;
}
```

```
ostream& operator<<(ostream& out,
const Clock& c){
  out << setfill('0')<<setw(2)
<<c.hour<<":"<<setw(2)<<c.min;
   if(c.ap)
       out << " PM";
   else
       out << " AM";
    return out;
}
int main(){
   Clock c,d;
   c = d++;
   cout << "Clock c: " << c << endl;
   cout << "Clock d: " << d << endl;
   d = ++c;
   cout << "Clock c: " << c << endl;
   cout << "Clock d: " << d << endl;
}
```

自增运算符 重载

输出:

```
Clock c: 12:00 AM
Clock d: 12:01 AM
Clock c: 12:01 AM
Clock d: 12:01 AM
```

赋值运算符 重载

C++允许重载赋值运算符:

```
class Integer {
    int x;
public:
    Integer(int x=0):x(x)   {
    }
    Integer& operator = (const Integer& Int)    {
        x=Int.x;
        cout<<"function is invoked"<<endl;
        return *this;
    }
    void print()    {
        cout<<x<<endl;
    }
};
```

输出结果:

```
3
function is invoked
3
```

赋值运算符 重载

但会默认有一个缺省的赋值运算符，每个成员变量会直接拷贝值：

```
class Integer {
    int x;
public:
    Integer(int x=0):x(x)   {
    }
    void print()    {
        cout<<x<<endl;
    }
};
int main()  {
    Integer a=3, b;
    a.print();
    b=a; // default operator ++
    b.print();
    return 0;
}
```

输出结果：
3
3

赋值运算符　重载

所以当成员变量包含指针类型的时候，需要注意浅拷贝和深拷贝的区别：

```
class IntArray {
    int *a, n;
public:
    IntArray(int n=1):n(n) {
        a=new int[n];
    }
    ~IntArray() {
        delete[] a;
    }
    int& operator [] (const int& i) {
        assert(0<=i&&i<n);
        return a[i];
    }
    void print()  {
        for (int i=0;i<n;++i)
            cout<<a[i]<<" ";
        cout<<endl;
    }
};
```

```
int main() {
    IntArray a(4), b;
    for (int i=0;i<4;++i) a[i]=i;
    a.print();
    b=a; // default operator = is ok, but not recommended
    b.print();
    for (int i=0;i<4;++i) a[i]=-i;
    a.print();
    b.print(); // would be same with a
    a.~IntArray();   尽量不要显示调用析构函数
    b.print(); // undefined behavior, may cause segmentation fault
    return 0; }
```

Segmentation fault!!!

有指针的话要提供三个函数：赋值运算符，拷贝构造函数，析构函数

输出结果：

```
0 1 2 3
0 1 2 3
0 -1 -2 -3
0 -1 -2 -3
2008160928 0 2008154448 0
```

赋值运算符 重载

所以当成员变量包含指针类型的时候，需要注意浅拷贝和深拷贝的区别：

```
class IntArray  {
    int *a, n;
public:
 ……
IntArray& operator = (const IntArray& A)
{
        delete[] a;
        n=A.n;
        a=new int[n];
        memcpy(a,A.a,sizeof(int)*n);
        return *this;
    }
  ……
    }
```

```
int main()  {
    IntArray a(4), b;
    for (int i=0;i<4;++i) a[i]=i;
    a.print();
    b=a; // deep copy is good
    b.print();
    for (int i=0;i<4;++i) a[i]=-i;
    a.print();
    b.print(); // would be different from a
    a.~IntArray();
    b.print(); // nothing happend
    return 0;
}
```

输出结果：

```
0 1 2 3
0 1 2 3
0 -1 -2 -3
0 1 2 3
0 1 2 3
```

赋值运算符 重载

所以当成员变量包含指针类型的时候，需要注意浅拷贝和深拷贝的区别：

```
class IntArray  {
    int *a, n;
public:
 ……
IntArray& operator = (const IntArray& A)
{
        delete[] a;
        n=A.n;
        a=new int[n];
        memcpy(a,A.a,sizeof(int)*n);
        return *this;
    }
  ……
    }
```

```
int main()  {
    IntArray a(4), b;
    for (int i=0;i<4;++i) a[i]=i;
    a.print();
    b=a; // deep copy is good
    b.print();
    for (int i=0;i<4;++i) a[i]=-i;
    a.print();
    b.print(); // would be different from a
    a.~IntArray();
    b.print(); // nothing happend
    return 0;
}
```

输出结果：

```
0 1 2 3
0 1 2 3
0 -1 -2 -3
0 1 2 3
0 1 2 3
```

转型运算符 重载

前面讲了类C的转换构造函数，带有一个参数（注意不是C&类型的），假设这个参数的类型是paramtype,转换构造函数讲paramtype类型转换为C类型

转换构造函数可以讲其他类型转换为所需的类类型。如果要进行相反的转型动作，将类的类型转换成其他类型，可以重载转型操作符，语法如下：

```
operator othertype();
```

```
class Clock{
        public:
                Clock(int =12,int=0,int=0);
                Clock& tick();
                friend ostream&
operator<<(ostream&,const Clock&);
                Clock& operator++(); //前加
                Clock  operator++(int); //后加
                operator int();
        private:
                int hour;
                int min;
                int ap; // 0 is AM, 1 is PM
};
```

转型运算符 重载

前面讲了类C的转换构造函数，带有一个参数（注意不是C&类型的），假设这个参数的类型是paramtype,转换构造函数讲paramtype类型转换为C类型

转换构造函数可以讲其他类型转换为所需的类类型。如果要进行相反的转型动作，将类的类型转换成其他类型，可以重载转型操作符，语法如下：

```
operator othertype();
```

```
class Clock{
 public:
    Clock(int =12,int=0,int=0);
    Clock& tick();
    friend ostream& operator<<(ostream&,const Clock&);
    Clock& operator++();  //前加
    Clock  operator++(int); //后加
    operator int();
 private:
    int hour;
    int min;
    int ap; // 0 is AM, 1 is PM
};
```

转型运算符 重载

```
Clock::operator int(){
//用整数表示的时间称为军事时间，其规则是:
//1300代表1:00PM,1400代表2:00PM,0代表12：00AM,
//时间如果是8：34，则对应整数值是834
    cout << "Converting from Clock to int...." <<endl;
   int time = hour;
   if(time == 12)
            time = 0;
   if(ap==1)
            time += 12;
   time *= 100 ;
   time += min;
   return time;
}
```

```
int main(){
    Clock c,d;
    Clock getUp(8,55,0);
    int getUp_Int;
    c = d++;
    cout << "Clock c: " << c << endl;
    cout << "Clock d: " << d << endl;
    d = ++c;
    cout << "Clock c: " << c << endl;
    cout << "Clock d: " << d << endl;
    getUp_Int = (int) getUp; //调用转型函数 gteUp.int()
    cout << "getUp_Int: " << getUp_Int <<endl;
    getUp_Int = getUp; //调用转型函数 gteUp.int()
    cout << "getUp_Int: " << getUp_Int <<endl;
   return 0;
}
```

转型运算符 重载

输出:
Clock c: 12:00 AM
Clock d: 12:01 AM
Clock c: 12:01 AM
Clock d: 12:01 AM
Converting from Clock to int....
getUp_Int: 855
Converting from Clock to int....
getUp_Int: 855

运算符的预置含义

1. 某些内置运算符的含义和接受相同参数的运算符组合的含义相同。
   如果有 int a; ++a等价于 a+=1 以及 a = a +1。
2. 但这一现象对于自定义的运算符并不适用。
   如编译器不会根据Z::operator+()和Z::operator=()的定义生成Z::operator+=()的定义
   需要重载 +=， -=等运算符
3. 语法格式
   C& operator+=(C c) //以值的方式传递参数，
   C& operator+=(const C &c)//以常引用方式传递参数

见C++程序设计语言（第1-3部分）本贾尼 斯特劳斯特鲁普著， 王刚 杨巨峰 译 机械工业出版社
Chapter 18.2.2

运算符+=的重载

```
class Integer {
    int x;
public:
        ......
    Integer& operator += (const Integer& Int){
        x += Int.x; //修改了当前操作数
        return *this;
    }
    Integer operator +(const Integer& Int) {
        return Integer(x+Int.x); //创建了临时对象，并不修改操作数
    }
     ......
};
```

运算符+=的重载

```
class Integer {
    int x;
public:
        ......
    Integer& operator += ( Integer Int){
        x += Int.x; //修改了当前操作数
        return *this;
    }
    Integer operator +(Integer Int) {
        return Integer(x+Int.x); //创建了临时对象，并不修改操作数
    }
     ......
};
```

运算符+=的重载

```
int main()  {
    Integer a=3,b=4;
    a.print();
    b.print();
    Integer c=a+b;
    c.print();
    cout << "====================" <<endl;
    Integer d=a-b;
    d.print();
    Integer e=-a;
    e.print();
    cout << "====================" <<endl;
    a += 3;
    a.print();
    return 0;
}
```

```
3
4
7
====================
-1
-3
====================
6
```

输入输出运算符的重载

- ✓ C++对右移运算符>>进行了重载，使其能输入信息到所有的c++内建数据类型，
  如 int i; cin >>i; 则翻译为 cin.operator>>(i)

- ✓ 程序员可对operator>>进行重载，以支持用户自定义的数据类型。

  >>的第一个操作数是系统类的对象（如上面的cin是系统类istream对象），
  >>重载为顶层函数

  语法格式：
  ① istream& operator>>(istream& , C &c) //注意 C&c 前面没有const, istream前面也没有const
  ② 并在相应的类内将该函数设为friend

由于左边是输出流而非本类对象，所以要用友元来调用私有成员

左移运算符<<类似，用以进行信息的输出
语法格式：
① ostream& operator<<(ostream& , const C &c)
② 并在相应的类内将该函数设为friend

输入输出运算符的重载

```
class Clock{
        public:
                Clock(int =12,int=0,int=0);
                Clock& tick();
                //在friend函数中可以访问类对象的私有数据
                friend ostream& operator<<(ostream&,const Clock&);
                friend istream& operator>>(istream&, Clock&);
                ……
};
```

```
ostream& operator<<(ostream& out, const Clock& c){
        out << setfill('0')<<setw(2)<<c.hour<<":"<<setw(2)<<c.min;
        if(c.ap)
            out << " PM";
         else
             out << " AM";
    return out;
}
```

输入输出运算符的重载

```
class Clock{
        public:
                Clock(int =12,int=0,int=0);
                Clock& tick();
                //在friend函数中可以访问类对象的私有数据
                friend ostream& operator<<(ostream&,const Clock&);
                friend istream& operator>>(istream&, Clock&);
                ……
};
```

```
istream& operator>>(istream& in,  Clock& c){
        cout << "输入时钟的时 分和由0，1区分的上下午"<< endl;
        in >> c.hour >> c.min >> c.ap; //每一次输入后的返回值都是in
        return in;
}
```

输入输出运算符的重载

```
class Clock{
        public:
                Clock(int =12,int=0,int=0);
                Clock& tick();
                //在friend函数中可以访问类对象的私有数据
                friend ostream& operator<<(ostream&,const Clock&);
                friend istream& operator>>(istream&, Clock&);
                ……
};
```

```
int main(){
        Clock c,d;
        cin >> c >> d; //相当于调用 operator>>(cin, c) 之后再调用) operator>>(cin, d)
        d = c++;
        cout << “Clock c: ” << c << endl; //每一次输出后返回值都是cout
    //相当于调用operator(operator<<(operator<<(cout,"Clock c: "),c),endl)
        cout << "Clock d: " << d << endl;
}
```

输入输出运算符的重载

```
int main(){
        Clock c,d;
        cin >> c >> d; //相当于调用 operator>>(cin, c) 之后再调用) operator>>(cin, d)
        cout << “Clock c: ” << c << endl; //每一次输出后返回值都是cout
    //相当于调用operator(operator<<(operator<<(cout,"Clock c: "),c),endl)
        cout << "Clock d: " << d << endl;
}
```

```
输入时钟的时 分和由0，1区分的上下午
11 22 0
输入时钟的时 分和由0，1区分的上下午
10 20 1
Clock c: 11:22 AM
Clock d: 10:20 PM
```

下标运算符的重载

✓下标运算符必须以成员函数的形式进行重载
✓语法形式如下:

```
class C{
   ……
   returntype& operator[] (paramtype); //可以用于修改对象
   const returntype& operator[] (paramtype) const;//可以访问但不能修改对象
  …….
}
```

形参为整型
返回值类型为引用类型

例：整数数组 演示下标运算符[ ]的重载

```
class IntArray
{
public:
        int& operator[]( int );
        const int& operator[]( int ) const;
        IntArray( int s );
        int getSize() const { return size; }
private:
        int size;
        int* a;
};
```

例：整数数组 演示下标运算符[ ]的重载

```
int& IntArray::operator[]( int i )
{
        if( i<0 || i>=size )
                throw string( "OutOfBounds" );
        return a[i];
}
const int& IntArray::operator[]( int i ) const
{
        if( i<0 || i>=size )
                throw string( "OutOfBounds" );
        return a[i];
}
```

```
void print(const IntArray& c){
        cout << "In print ....." <<endl;
        for(int i =0; i <c.getSize();i++)
                cout << c[i] << endl; //这个调用的是哪个版本的[]运算符
}
```

例：整数数组 演示下标运算符[ ]的重载

```
int main()
{
        IntArray b(5);
        int i;
        try{
                for( i=0; i<b.getSize(); i++ )
                        b[i] = 2*i; //返回是引用，可以放在赋值运算符的左边
                for( i=0; i<5; i++ )
                        cout << b[i] << '\n';
                cout << b[6];
        } catch ( string s ) {
                cerr << s << '\n';
                cerr <<"i = "<< i << endl;
        }
        print(b); //这个函数里会用到重载的下标运算符
        return 0;
}
```

例：整数数组 演示下标运算符[ ]的重载

```
输出：
0
2
4
6
8
OutOfBounds
i = 5
In print ......
0
2
4
6
8
```

中山大學
SUN YAT-SEN UNIVERSITY

谢 谢

中山大学计算机学院

主讲人：XXX

中山大学MOOC课程组