C++程序设计

主讲：王老师

尚德机构

学习是一种信仰

教材介绍

C++程序设计

（2008年版）

编著：刘振安

机械工业出版社

考试题型

单选题　　1分×20题 = 20分

填空题　　1分×20题 = 20分

程序改错题　4分×5题 = 20分

完成程序题　4分×5题 = 20分

程序分析题　5分×2题 = 10分

程序设计题 10分×1题 = 10分

第九章　运算符重载及流类库

目录

第9章

运算符重载及流类库

§9.1 运算符重载

一、重载对象的赋值运算符

编译器在默认情况下为每个类生成一个默认的赋值操作，用于同类的两个对象之间相互赋值，默认的含义是逐个为成员赋值，即将一个对象的成员的值赋给另一个对象相应的成员，但这种赋值方式对于某些类可能是不正确的，如类str的数据成员是“char *st”，则语句“str s1(“hello”),s2(“world”); s2=s1;”经赋值后，s2.st和s1.st是同一块存储地址，当s2和s1的生存期结束时，存储”hello”的变量将被删除两次，这是个严重的错误。另外，对于”s1=s1;”的情况，也应不执行赋值操作。

§9.1 运算符重载

因此，程序必须为str类定义其自己的赋值操作，这个操作应能保证s1.st=s2.st，但两者各自具有自己的存储地址，同时如果发现自己为自己赋值，则不执行赋值操作，这个操作可用下面的方法实现：

```
str& str::operator=(str& a)
{
    if(this==&a)              //防止a=a这样的赋值
        return *this;         //a=a,退出
    delete st;
    st=new char [strlen (a.st) +1];    //申请内存
    strcpy(st, a.st);         //将对象a的数据复制一份到申请的内存区
    return *this; }           //返回this指针指向的对象
```

§9.1 运算符重载

程序中语句“str& str::operator=(str& a)”表示函数operator=(str&)返回str类对象a的引用。

C++中关键字operator和运算符连用就表示一个运算符函数，如本例中的”operator =”表示重载赋值运算符“=”，又如”operator+”表示重载“+”运算符，所以应将“operator =”和“operator +”从整体上视为一个（运算符）函数名。

§9.1 运算符重载

上面语句中当str类定义了赋值运算符函数operator=(str&)之后，“operator＝”就是str类的成员函数名，对象s2调用这个函数，参数是s1，完成用s1为s2赋值的操作，写成正规的函数调用语句为“s2.operator=(s1);”，但也可按照原来的书写习惯写成“s2=s1;”此时C++将自动将其解释为正规的函数调用语句“s2.operator=(s1);”，即s2调用成员函数str::operator=(str&)完成赋值操作。因为该成员函数返回一个引用，所以它也可以用于连续赋值的操作，如“s3=s2=s1;”,C++在编译过程中将自动将其解释为正规的函数调用语句“s3.operator= （s2.operator=(s1)）；”。

§9.1 运算符重载

由此可见，运算符函数虽然是函数，但完全可以不写成函数调用的形式，而是采用原来的书写习惯，系统将会自动按照其真正的含义执行，运算符重载就是函数重载，这就是运算符重载的实质。

例9.1 完整实现str类的例子

```
#include <iostream>
#include <string>
using namespace std;
class str{
private:
        char *st;
public:
        str( char *s );    //使用字符指针的构造函数
        str( str& s);    //使用对象引用的构造函数
        str& operator=( str& a );    //重载使用对象引用的赋值运算符
        str& operator=( char *s );    //重载使用指针的赋值运算符
        void print( ) {cout<<st<<endl;}    //输出字符串
        ~str( ) {delete st;}
};
str::str(char *s)
{
        st=new char[strlen(s)+1];      //为st申请内存
        strcpy(st,s);    //将字符串s复制到内存区st
}
```

```
str::str(str& a)
{
        st=new char[strlen(a.st)+1];      //为st申请内存
        strcpy(st,a.st);        //将对象a的字符串复制到内存区st
}
str& str::operator=(str& a)
{
        if (this==&a)    //防止出现a = a这样的赋值
                return *this;     //a = a，退出
        delete st;        //不是自身，先释放内存空间
        st=new char[strlen(a.st)+1];    //重新申请内存
        strcpy(st,a.st);    //将对象a的字符串复制到申请的内存区
        return *this;    //返回this指针指向的对象
}
str& str::operator=(char *s)
{
        delete st;    //是字符串直接赋值，先释放内存空间
        st=new char[strlen(s)+1];    //重新申请内存
        strcpy(st,s);    //将字符串s复制到内存区st
        return *this;
}
```

```
void main() {
        str s1("We"),s2("They"),s3(s1);    //调用构造函数和复制构造函数
        s1.print();s2.print();s3.print();    //显示We They We
        s2=s1=s3;    //调用赋值操作符
        s3="Go home!";    //调用字符串赋值操作符
        s3=s3;    //调用赋值操作符但不进行赋值操作
        s1.print();s2.print();s3.print();    //显示We We Go home!
}
```

```
We
They
We
We
We
Go home!
```

程序分析

程序中定义两个赋值操作符，是因为程序中经常要进行形如s1=”hello”;这样的赋值，因此为str类定义了一个成员函数str& str::operator=(char *s)，以绕过类型转换所带来的运行时间开销。

另外，主程序也可写成下面形式，效果一样：

```
void main() {
        str s1("We"),s2("They"),s3(s1);    //调用构造函数和复制构造函数
        s1.print();s2.print();s3.print();    //显示We They We
        s2.operator=((s1).operator=(s3));    //调用赋值操作符
        s3="Go home!";    //调用字符串赋值操作符
        s3.operator=(s3);    //调用赋值操作符但不进行赋值操作
        s1.print(); s2.print(); s3.print(); } //显示We We Go home!
```

二、运算符重载的实质

C++的任何运算都是通过函数来实现的，所以运算符重载其实就是函数重载，这就是运算符重载的实质，要重载某个运算符，只要重载相应的函数就可以了，与以往稍有不同的是，需要使用新的关键字operator，它和C++的一个运算符连用，可以构成一个运算符函数名，如“operator+”，通过这种构成方法就可以像重载普通函数那样重载运算符函数operator+（）。由于C++已经为各种基本数据类型定义了该运算符函数，所以只需要为自己定义的类型重载operator+（）就可以了。

不能重载的运算符

一般地，为用户定义的类型重载运算符都要求能够访问这个类型的私有成员，为此需通过两种方法实现，即将运算符重载为这个类型的成员函数或友元，前者称类运算符，后者称友元运算符。

C++的运算符大部分都可以重载，除了. :: * 和？:五种运算符，前三者是因为有特殊含义，后两者是因为不值得重载；另外，sizeof和#不是运算符，故不能重载；而= （） [ ] ->这四个运算符只能用类运算符来重载。

三、<<、>>和++运算符重载实例

插入符“<<”和提取符“>>”的重载与其他运算符重载一样，不同的是它们必须作为类的友元重载。

1、插入符函数的一般形式如下：

```
ostream &operator<<(ostream & output, 类名 & 对象名)
{    ……  //函数代码
     return output; }
```

output是类ostream对象的引用，它是cout的别名，即Ostream & output=cout。调用参数时，output引用cout（即cout的别名），插入符函数的第二个参数可以使用对象名，也可使用对象的引用。

2、提取符函数的一般形式如下：

```
istream &operator>>(istream & input, 类名 & 对象名)
{    ……  //函数代码
     return input;
}
```

input是类istream对象的引用，它是cin的别名，即istream & input=cin。调用参数时，input引用cin（即cin的别名）。

另外，提取符函数需要返回新的对象值，所以应该使用引用，即“类名&对象名”，不能使用“类名 对象名”，插入符函数不改变对象的值，所以两种方法都可以。

例9.2、使用友元函数重载运算符<<和>>

```
#include <iostream.h>
class test {
private:
        int i;
        float f;
        char ch;
public:
        test(int a=0,float b=0,char c='\0') {i=a;f=b;ch=c;}
        friend ostream &operator<<(ostream &,test);
        friend istream &operator>>(istream &,test &);
};
ostream &operator<<(ostream & stream,test obj)
{
        stream<<obj.i<<",";  //stream是cout的别名
        stream<<obj.f<<",";
        stream<<obj.ch<<endl;
        return stream;
}
```

```
istream &operator>>(istream & t_stream,test&obj)
{
        t_stream>>obj.i;    //t_stream是cin的别名
        t_stream>>obj.f;
        t_stream>>obj.ch;
        return t_stream;
}
void main( ) {
        test A(45,8.5,'W');
        operator<<(cout,A);
        test B,C;
        cout<<"Input as i f ch:";
        operator>>(cin,B);  operator>>(cin,C);
        operator<<(cout,B);  operator<<(cout,C);
}
```

```
45,8.5,W
Input as i f ch:5 5.8 A 2 3.4 a
5,5.8,A
2,3.4,a
```

使用友元函数重载运算符<<和>>

上面的主程序写成函数调用形式，是为了演示运算符就是函数重载，一般在使用时，则直接使用运算符，如下面的正规使用方法：

```
void main( ) {
        test A(45,8.5,'W');
        cout<<A;
        test B,C;
        cout<<"Input as i f ch:";
        cin>>B>>C;    //输入5 5.8 A 2 3.4 a
        cout<<B<<C;   //输出5 5.8 A      2 3.4 a
}
```

使用友元函数重载运算符<<和>>

显然，运算符“<<”重载函数有两个参数，第一个是ostream类的一个引用，第二个是自定义类型的一个对象，重载方式为友元重载，函数的返回类型是一个ostream类型的引用，在函数中实际返回的是该函数的第一个参数，这样作是为了使得“<<”能够连续使用。

例9.3、使用类运算符重载“++”运算符

```
#include <iostream>
using namespace std;
class number {
        int num;
public:
        number (int i) {num=i;}
        int operator++();       //前缀：++n
        int operator++(int);    //后缀：n++
        void print() {cout<<"num="<<num<<endl;}
};
int number::operator ++()   {   num++;      return num;}
int number::operator ++(int){   int i=num;          num++; return i;}  //不用给出形参名
void main() {
        number n(10);
        int i=++n;    // i=11,n=11，使用函数调用方式的语句为int i=n.operator++();
        cout<<"i="<<i<<endl;  //输出i=11
        n.print();  //输出n=11;
        i=n++;    // i=11,n=12, 使用函数调用方式的语句为i=n.operator++(0);
        cout<<"i="<<i<<endl;  //输出i=11
        n.print();  //输出n=12
}
```

```
i=11
num=11
i=11
num=12
```

例9.4、使用友元运算符重载++运算符

```
#include <iostream>
using namespace std;
class number {
        int num;
public:
        number (int i) {num=i;}
        friend int operator++(number&);    //前缀：++n
        friend int operator++(number&,int);    //后缀：n++
        void print() {cout<<"num="<<num<<endl;}
};
int operator ++(number& a) { a.num++;   return a.num;}
int operator ++(number& a,int){   int i=a.num++;   return i;} //不用给出形参名
void main() {
        number n(10);
        int i=++n;    //i=11,n=11
        cout<<"i="<<i<<endl;  //输出i=11
        n.print();  //输出n=11;
        i=n++;    //i=11,n=12
        cout<<"i="<<i<<endl;  //输出i=11
        n.print();  //输出n=12
}
```

```
i=11
num=11
i=11
num=12
```

例9.4、使用友元运算符重载++运算符

本例中，因为运算符需要修改操作数，所以必须使用引用参数。

有的C++运算符编译器不区分前缀或后缀运算符，这是只能通过对运算符函数进行重载来反映其为前缀或后缀运算符。

注意不能自己定义新的运算符，只能是把C++原有的运算符用到自己设计的类上面去，同时，经过重载，运算符并不改变原有的优先级，也不改变它所需的操作数数目。当不涉及到定义的对象时，它仍然执行系统预定义的运算，只有用到自己定义的对象上，才执行新定义的操作。

四、类运算符和友元运算符的区别

如果运算符所需的操作数（尤其是第一个操作数）希望进行隐式类型转换，则运算符应通过友元来重载；另一方面，如果一个运算符的操作需要修改对象的状态，则应当使用类运算符，这样更符合数据封装的要求。但参数是使用引用还是对象，则要根据运算符在使用中可能出现的情况而定。

C++同时具有类运算符和友元运算符，参数也可以使用对象或引用，这就给编程带来了极大的方便，但也要注意各自适用的特定场合。

例9.5、使用对象作为友元函数参数来定义运算符+的例子

```
#include <iostream.h>
class complex {
private:
        double real,imag;
public:
        complex(double r=0,double i=0) {real=r;imag=i;}  //构造函数
        friend complex operator+(complex,complex);
        void show() {cout<<real<<"+"<<imag<<"i"<<endl;}
};
complex operator+(complex a,complex b)
{
        double r=a.real+b.real;
        double i=a.imag+b.imag;
        return complex(r,i);
}
void main() {
        complex x(5,3),y;
        y=x+7;  //相当于"y=operator+(x,7);"，通过调用构造函数将x和7初始化，有y=12+3i
        y=7+y;  //相当于"y=operator+(7,y);"，通过调用构造函数将7和y初始化，有y=19+3i
        y.show();  //输出19+3i
}
```

四、类运算符和友元运算符的区别

如果将上例中的友元运算符换为类运算符，假设类运算符为如下形式：

```
complex operator+(complex a)
{
    double r=a.real+real;
    double i=a.imag+imag;
    return complex(r,i);
}
```

四、类运算符和友元运算符的区别

因为语句“y=x+7;”等价于“y=x.operator+(7);”,而“7”可通过构造函数转换成complex类型的对象，所以使其参数匹配，从而保证正常工作。而语句“y=7+y;”等价于“y=7.operator+(y);”,因常量不能作为对象名，系统将无法解释7.operator而出错。如果参数使用引用形式，则不仅语句“y=7+y;”，连语句“y=x+7;”也因引用不产生临时对象，常量又不能通过构造函数转化为complex类型，从而不能通过编译。

类运算符比友元运算符少一个参数，这是因为类运算符函数作为成员函数时具有this指针，complex &a是形参表，a是类complex的引用对象，所以使用对象名，a.real代表对象a的数据成员real，而real则为当前对象的数据成员，它用于完成两个复数的加法运算。

五、下标运算符[ ]的重载

C++中，运算符[ ]只能用类运算符来重载。下例中的一位数组iArray完全按照C++关于数组的规定：数组a[n]，其下标从0开始，到n-1结束。

例9.6、设计类iArray，对其重载下标运算符[ ]，并能在进行下标访问时检查下标是否越界。

```
#include <iostream>
#include <iomanip>
using namespace std;
class iArray {
        int _size;
        int *data;
public:
        iArray(int);
        int& operator[](int);  //声明"[]"运算符函数，函数返回整型数首地址
        int size()const{return _size;}
        ~iArray(){delete []data;}
};
iArray::iArray(int n)  //构造函数中，且其中n>1
{
        if(n<1){cout<<"错误";exit(1);}
        _size=n;
        data=new int[_size];    //申请内存空间
}
```

```
int&iArray::operator [](int i)  //定义[]运算符函数，合理范围0～_size-1
{
        if(i<0||i>_size-1) { cout<<"数值越界";  delete[]data; exit(1); }
                //检查数值是否越界，如越界则释放数组所占内存空间，退出
        return data[i];
}
void main() {
        iArray a(10);
        cout<<"数组元素个数 = "<<a.size()<<endl;
        for(int i=0;i<a.size();i++)
                a[i]=10*(i+1);
        for(i=0;i<a.size();i++)
                cout<<setw(5)<<a[i];
}
```

```
数组元素个数 = 10
   10   20   30   40   50   60   70   80   90  100
```

§9.2 流类库

一、流类库的基础类

在C++中，输入输出是通过**流**完成的，把接收输出数据的地方称为目标，输入数据的出处称为源，而输入输出操作可以看成字符序列在源、目标以及对象之间的流动。

C++将与输入和输出有关的操作定义为一个类体系，放在一个系统库里，以备用户调用，这个类体系称为**流类**，提供这个流类实现的系统库称为**流类库**。

C++的流类库由几个进行I/O操作的基础类和几个支持特定种类的源和目标的I/O操作的类组成。这些基础类在头文件<iostream>中说明，其中**streambuf类**管理一个流的缓冲区，普通用户一般不涉及它，而只使用**ios类、istream类和ostream类中提供的公有接口**，完成流的提取和插入操作。

一、流类库的基础类

在C++中，如果在多条继承路径上有一个汇合处，则称该汇合处的基类为公共基类，如ios类。因为可通过不同路径访问这个基类，从而使其产生多个实例，有时会引起二义性，如果想使这个公共的基类只产生一个实例，就需要将它说明为虚基类。ios类就是istream类和ostream类的虚基类，用来提供对流进行格式化I/O操作和错误处理的成员函数。用关键字virtual可将公共基类说明为虚基类。从ios类公有派生的istream和ostream两个类分别提供对流进行提取操作和插入操作的成员函数，而iostream类通过组合istream类和ostream类来支持对一个流进行双向操作，它并没有提供新的成员函数。

9.2.1 流类库的基础类

当在多条继承路径上有一个公共的基类,在这些路径中的某几条汇合处，这个公共的基类就会产生多个实例(或多个副本)，若只想保存这个基类的一个实例，可以将这个公共基类说明为虚基类。

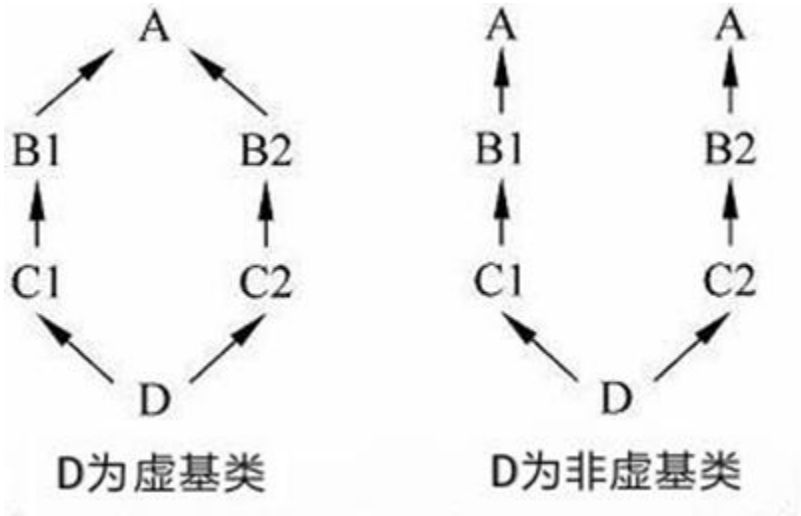
在继承中产生歧义的原因有可能是继承类继承了基类多次，从而产生了多个拷贝，即不止一次的通过多个路径继承类在内存中创建了基类成员的多份拷贝。虚基类的基本原则是在内存中只有基类成员的一份拷贝。这样，通过把基类继承声明为虚拟的，就只能继承基类的一份拷贝，从而消除歧义。

用virtual限定符把基类继承说明为虚拟的。

在派生类继承基类时，加上一个virtual关键词则为虚拟基类继承，如：

虚基类主要解决在多重继承时，基类可能被多次继承，虚基类主要提供一个基类给派生类，如：

```
class derive:virtual public base
{};
```



```cpp
#include <iostream>
using namespace std;
class B0// 声明为基类B0
{
    int nv;//默认为私有成员
public://外部接口
    B0(int n){ nv = n; cout << "Member of B0" << endl; }//B0类的构造函数
    void fun(){ cout << "fun of B0" << endl; }
};
class B1 :virtual public B0
{
    int nv1;
public:
    B1(int a) :B0(a){ cout << "Member of B1" << endl; }
};
class B2 :virtual public B0
{
    int nv2;
public:
    B2(int a) :B0(a){ cout << "Member of B2" << endl; }
};
class D1 :public B1, public B2
{
    int nvd;
public:
    D1(int a) :B0(a), B1(a), B2(a){ cout << "Member of D1" << endl;
    void fund(){ cout << "fun of D1" << endl; }
};
```

```cpp
int main(void)
{
    D1 d1(1);
    d1.fund();
    d1.fun();
    return 0;
}
```

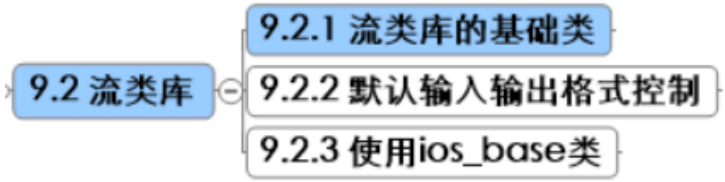
执行结果：

```
Member of B0
Member of B1
Member of B2
Member of D1
fun of D1
fun of B0
```

52.
```
#include <iostream>
using namespace std;
class x
{protected:int a;
public: x(){a=1;}
};
class x1:virtual public x
{public:
    x1(){a+=1;cout<<"x1:"<<a<<",";}
};

class x2:virtual public x
{public:
    x2(){a+=2;cout<<"x2:"<<a<<",";}
};

class y:public x1,public x2
{public:
    y(){cout<<"y:"<<a<<endl;}
};
void main()
{y obj;}
```



一、流类库的基础类

C++的流类库预定义了4个流：cin、cout、cerr和clog，事实上，可以将cin视为istream的一个对象，将cout视为ostream的一个对象。

流是一个抽象概念，在实际进行I/O操作时，必须将流和一种具体的物理设备联接起来，如cin联接键盘，cout、cerr和clog联接显示终端。

二、默认输入输出格式控制

1、数值数据

关于数值数据，默认方式是自动识别浮点数并用最短的方式输出，如将5.0作为5输出，输入3.4e+2，输出340等，还可以将定点数分成整数和小数部分，

例如“int a；double b；cin>>a>>b;”，

当用键盘输入23.45时，a为23，b为0.45；

二、默认输入输出格式控制

2、字符数据

①单字符：默认方式是舍去空格，直到读到字符为止。例如“char a,b,c;”定义单字符对象a、b和c，“cin>>a>>b>>c;”能将连续的3个字符分别正确的赋给相应对象；

②字符串：默认方式是从读到第一个字符开始，到空格符结束。例如“char a[10];”定义含10个字符位置的字符串对象a，“cin>>a;”

输入的如果是“abc def”，则a内为abc；

③字符数组：默认方式是使用数组名来整体读入。例如“char a[30]”，使用“cin >>a;”读入，“cout<<a;”输出。

二、默认输入输出格式控制

④**字符指针**：尽管为其动态分配了地址，也只能采取逐个赋值的方法，不仅不以空格符结束，反而舍去空格，读到字符才计数。

⑤因为字符串没有结束位，所以将其作为整体输出时，有效字符串的后面将出现乱码，这时可用手工增加字符串结束符“\0”来消除乱码，例如：

```
char *p=new char[5];  //申请5个字符所需的内存地址
for(int i=0;i<4;i++)    cin>>*(p+i);  //假设输入w e andf
p[4]= '\0' ;   //将结束符“\0”赋给最后一个字符
cout<<p;     //输出前四个字符wean
```

二、默认输入输出格式控制

⑥当用键盘同时给一个单字符对象和一个字符串对象赋值时，不要先给字符串赋值，如果先给它赋值，应强行使用结束符，例如：

```
char b[3],c;  //定义含3个字符的字符串对象b和单字符对象c
cin>>b>>>c;
b[3]='\0'; //在b的第三个字符处强行使用结束符"\0"
```

假如输入“we a”，因字符串读入的默认方式是从读到第一个字符开始，到空格符结束，所以b内为we，c内为a；同理，如果输入“w e a”，则b内为w，c为e。

二、默认输入输出格式控制

3、Bool（布尔型）数据

默认方式把输入的0识别为false，其他的值均识别为1。输出时，只有0和1两个值，

例如：

```
int a;  double b;  char c;  char str[50];  bool r;
cin>>a>>b>>c>>str>>r;
cout<<a<<" "<<b<<" "<<c<<" "<<str<<" "<<r<<" "<<endl;
```

输入数据	输出数据
23.567 wty56.78 yu	23 0.567 w ty56.78 1
234 3647586.7897354 dfghe 0	234 3.64759e+006 d fghe 0
456 3.4e+2 5 ty675 w	456 340 5 ty675 1

4、如果默认输入输出格式不能满足自己的要求，就必须重载它们。

三、使用ios_base类

1、ios_base类简介：

ios_base类派生ios类，ios类又是istream类和ostream类的虚基类。

§9.3 文件流

在C++中，文件操作是通过流来完成的，C++共有**输入文件流、输出文件流和输入输出文件流3种**，并已通过头文件<fstream>实现标准化。

要打开一个输入文件流，需定义一个**ifstream**类型的对象；要打开一个输出文件流，需定义一个**ofstream**类型的对象；如要打开输入输出文件流，则要定义一个**fstream**类型的对象。这三种类型都定义在头文件**<fstream>**中。

一、使用文件流

流类具有支持文件的能力，在建立和使用文件时，就像使用cin和cout一样方便，以写文件为例，如建立一个类似cout的输出文件流myFile，只要它是ofstream类的对象，就可以像使用cout一样，使用myFile将指定内容写入文件；同理，如建立一个类似cin的输入文件流getText，只要它是ifstream类的对象，就可像使用cin一样，用其读取文件内容。

例9.9、演示文件流的概念

abcdefgGoodBye!

```
#include <iostream>
#include <fstream>        //输入输出文件流头文件
using namespace std;
void main() {
        char ch[15],*p="abcdefg";
        ofstream myFile;      //建立输出流myFile
        myFile.open("myText.txt");      //建立输出流myFile和文件myText.txt之间的关联
        myFile<<p;      //使用输出流myFile将指针p所指字符串流向文件
        myFile<<"GoodBye!";      //使用输出流myFile直接将字符串流向文件
        myFile.close();      //关闭文件myText.txt
        ifstream getText("myText.txt");   //建立输入流getText及其和文件myText.txt的关联并打开
        for(int i=0;i<strlen(p)+8;i++)  //使用输入流getText每次从文件myText.txt读入1个字符
                getText>>ch[i];      //将每次读入的1个字符赋给数组的元素ch[i]
        ch[i]='\0';      //设置结束标志
        getText.close();      //关闭文件
        cout<<ch;      //使用cout使数组元素流向屏幕，输出“abcdefgGoodBye!”
}
```

1、对文件进行操作的方法：

①打开一个相应的文件流，如“ofstream myStream;”打开一个输出文件流。

②把这个流和相应的文件关联起来，如“myStream.open(“myText.txt”);”,就是通过open函数把两者关联起来，以后对文件流的操作就是对文件的操作。

因为ifstream、ofstream和fstream这3个类都具有自动打开文件的构造函数，而这个构造函数本身就具有open函数的功能，所以可用一条语句

“ofstream myStream("myText.txt");”来完成上面建立文件流及其和文件关联的两步。如果指定文件路径，路径中的”\”号必须使用转义字符表示，如

“ifstream getText("f:\\text\\myText.txt");”表示建立输入流文件ifstream和路径为"f:\text\myText.txt"的文件的关联。

③操作文件流，想对文件进行什么操作，对相应的文件流进行相应操作即可。

④及时关闭不再使用的文件，格式为：“文件流名.close();”

2、小结

综上所述，流是I/O流类的中心概念，它是一种抽象，负责在数据的生产者和消费者之间建立联系并管理数据的流动，程序将流对象看作是文件对象的化身。

二、几个典型流成员函数

1、输出流的open函数

①函数原型：void open(char const *,int filemode,int=filebuf::openprot);

②参数：第一个参数是要打开的函数名；第二个是文件打开方式；第三个是文件的保护方式，一般都使用默认值。

③参数具体设置：第二个参数filemode可取以下值：

ios_base::in	打开文件进行读操作，这样可避免删除现存文件的内容
ios_base::out	打开文件进行写操作，这是默认方式
ios_base::ate	打开一个已有的输入或输出文件并查找到文件尾
ios_base::app	打开文件以便在文件尾部添加数据
ios_base::binary	指定文件以二进制方式打开，默认为文本方式
ios_base::trunc	如文件存在，将其长度截断为零并清除原有内容

上面的这几种方式也可通过“或”运算符“|”同时使用

2、输入流类的open函数

①使用默认构造函数建立对象，然后调用open成员函数打开文件，如：

```
ifstream inFile;                    //用构造函数建立文件流对象inFile
inFile.open(“filename”,iosmode);    //文件流对象打开文件filename
```

②也可以使用指针方式，如：

```
ifstream * pinFile;                 //动态建立文件流对象，获得对象指针
pinFile->open(“filename”,iosmode);  //用对象指针调用open函数打开文件filename
```

2、输入流类的open函数

③还可以直接使用有参数构造函数指定文件名和打开模式，如：

ifstream inFile(“filename”,iosmode);

④输入流的打开模式iosmode有两种参数：

ios_base::in　　//打开文件用于输入（默认方式）

ios_base::binary　　//指定文件以二进制方式打开，默认为文本方式

3、close成员函数

close成员函数用于关闭与一个文件流相关联的磁盘文件，如果没有关闭该文件，因文件流是对象，在其生存期结束时，也将自动关闭该文件，但要养成及时关闭不再使用的文件的习惯，以避免误操作或干扰而产生错误。

4、错误处理函数

①各函数功能：在对一个流对象进行I\O操作时，可能产生错误，这时可使用文件流的错误处理成员函数进行错误类型判别，各函数作用如下表：

函数	功能
bad()	如果进行非法操作，返回true，否则返回flase
clear()	设置内部错误状态，如果用缺省参量调用则清除所有错误位
eof()	如果提取操作已经到达文件尾，则返回true，否则返回flase
good()	如果没有错误条件和没有设置文件结束标志，返回ture，否则返回flase
fail()	与good相反，操作失败返回flase，否则返回true
is_open	判定流对象是否成功的与文件关联，若是，返回ture，否则返回flase

4、错误处理函数

②可以使用上面这些函数检查当前流的状态，如：

if(cin.good()) cin>>data;

③函数clear更多的是用于在已知流发生错误的情况下清除流的错误状态，也可用于设置流的错误状态。

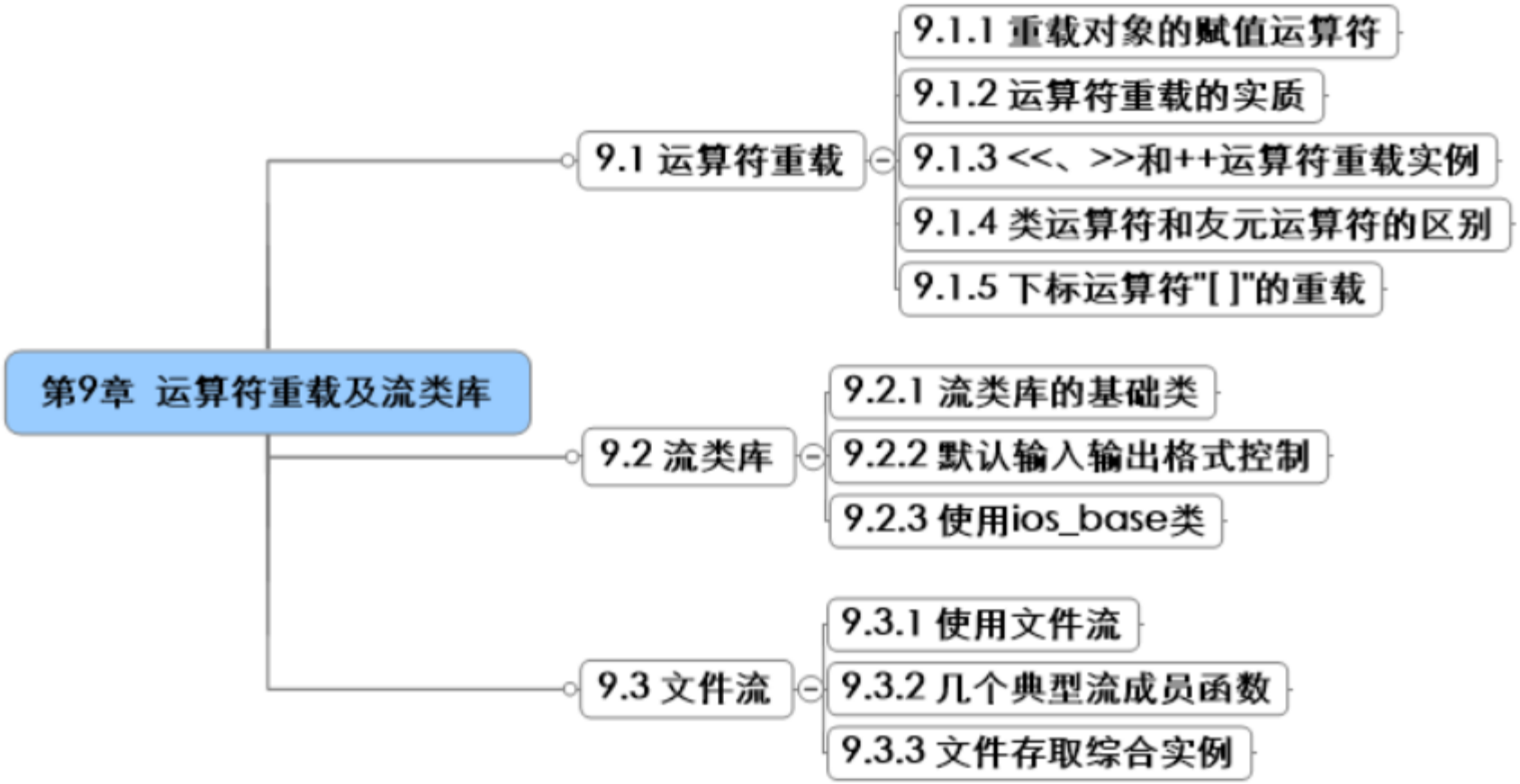
④“！”运算符经过了重载，它与fail函数执行相同的功能，因此表达式if(！cout)等价于if（cout.fail()）,if(cout)等价于if(! cout.fail())。

⑤如果需要测试文件是否存在，可以使用fail成员函数确定，如：

ifstream inFile（“filename”,ios_base::binary）；

if(inFile.fail()) cout<<”无错误”<<endl;

本章总结


第9章 运算符重载及流类库
9.1 运算符重载
9.1.1 重载对象的赋值运算符
9.1.2 运算符重载的实质
9.1.3 <<、>>和++运算符重载实例
9.1.4 类运算符和友元运算符的区别
9.1.5 下标运算符"[ ]"的重载
9.2 流类库
9.2.1 流类库的基础类
9.2.2 默认输入输出格式控制
9.2.3 使用ios_base类
9.3 文件流
9.3.1 使用文件流
9.3.2 几个典型流成员函数
9.3.3 文件存取综合实例

真题演练

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
   ifstream ifile;
   char fn[20],ch;
   cout<<"输入文件名";
   cin>>fn;
   ifile.open(fn)                    ;//打开指定的文件
   if (!ifile)
   {
      cout<<fn<<"文件不能打开"<<endl;
      return 0;
   }
   while ((ch=ifile.get())!=EOF)
      cout<<ch                       ;//输出字符
   cout<<endl;
   ifile.close();//关闭文件
   return 1;
}
```

18. 友元运算符@ obj 被 C++ 编译器解释为

A. operator@ (obj)　　B. operator@ (obj ,0)

C. obj. operator@ ( )　　D. obj. operator@ (0)

答案：A

47. 完成程序，使其输出结果为 x = 30

```
#include "stdafx.h"
#include <iostream>
using namespace std;
class Sample{
private:
   int x;
public:
   Sample(){}
   Sample(int a){x = a;}
   void disp(){
      cout << "x = " << x << endl;}
   friend Sample operator + (Sample &s1, Sample &s2);
};
_____(Sample &s1, Sample &s2){
   return Sample(s1.x + s2.x);
}
void main(){
   Sample obj1(10);
   Sample obj2(20);
   Sample obj3;
   _____;
   obj3.disp();
}
```

49. 在下面程序的横线处填上适当的字句,使输出为:0,2,10。

```
#include" stdafx. h"
#include <iostream>
#include" math. h"
using namespace std;
class Magic
{double x;
public:
Magic(double d =0.00) :x(fabs(d))
{}
Magic operator + (_____)
{
return Magic(sqrt(x * x + c. x * c. x));
}
_____ &operator << (ostream & stream,Magic & c)
{ stream << c. x;
return stream;
}
void main()
{Magic ma;
cout << ma << " , " << Magic(2) << " , " << ma + Magic( -6) + Magic( -8) << endl;
}
```

48.
```
#include <iostream>
using namespace std;
class F
{public:
```

_____

```
};
double F::operator()(double x,double y) const
{
return (x+5)*y;
}
```

_____

```
{F f;
cout<<f(1.5,2.2)<<endl;
}
```

51.
```
#include <iostream>
using namespace std;
void sort(int L[],int n)
{
    int j,k,flag,temp;
    flag=n-1;
    while(flag>0)
    {k=flag-1;flag=0;
    for(j=0;j<=k;j++)
        {if(L[j]>L[j+1])
            {temp=L[j];L[j]=L[j+1];
            L[j+1]=temp;flag=j;}
            }
    }
}
void main()
{int array[4]={7,2,3,4};
sort(array,4);
cout<<"The sorted numbers:";
for(int i=0;i<4;i++)
    cout<<array[i];
}
```

53. 声明一个 circle 类，有数据成员 Radius（半径，float 型），成员函数 GetArea( ) 计算圆的面积。在 main 函数中声明一个 circle 类的对象 c1，其半径为 5.6。调用 GetArea( ) 函数计算 c1 的面积，并显示该计算结果（cout << "圆的面积：" << c1. GetArea << endl;）。

```
#include <iostream. h >
class circle{
public:
  float GetArea( );
  circle(float r);
protected:
  float Radius;
};
circle::circle(float r){
  Radius = r;
}
float circle::GetArea( ){
  return Radius * Radius * 3. 14;
}
void main( ){
  circle c1(5. 6);
  cout << "圆的面积：" << c1. GetArea( ) << endl;
}
```

六、程序设计题:本大题共 1 小题,每小题 10 分,共 10 分。

53. 声明复数类 Complex,该类中有两个私有变量 real, image 分别表示一个复数的实部和虚部。为 Complex 类添加适当的构造函数,并使用友元函数 add 实现复数加法。

```
#include <iostream>
using namespace std;
class Complex
{
private:
double real, image;
public :
void setRI(double a, double b)
{
real = a;
image = b;
}
```

```
double getReal( )
{
return real;
}
double getImage( )
{
return image;
}
void print( ){
if( image >0)
cout <<"复数:" << real <<" + " << image <<"i" <<endl;
if( image <0)
cout <<"复数:" << real <<" - " << image <<"i" <<endl;
}
```

```
friend Complex add(Complex ,Complex);//声明友元函数
};
void main()
{
Complex c1(19, 0.864), c2, c3;
c2.setRI(90,125.012);
c3 = add(c1, c2);
cout <<"复数一:";c1.print();
cout <<"复数二:";c2.print();
cout <<"相加后:";c3.print();
}
```

53. 在类内添加：

```
Complex( ){}
Complex( double a,double b)
{
real = a;
image = b;
}
```

类外添加：

```
Complex add( Complex c1 , Complex c2)//定义友元函数
{
Complex c3;
c3. real = c1. real + c2. real;//访问 Complex 类中的私有成员
c3. image = c1. image + c2. image;
return c3;
```

53. 定义一个生日类，数据成员有年、月、日。定义一个人员类，数据成员有姓名、性别、生日。人员类中的生日是生日类的对象，两个类都有构造函数和显示函数。在主函数中声明一个人员类对象，屏幕显示其数据。

```
53. #include <iostream>
    using namespace std;
    class birth{
    private:
        int year,month,day;
    public:
        birth(int x,int y,int z){
            year=x;month=y;day=z;
        }
        void show(){
            cout<<"生日是"<<year<<"年"<<month<<"月"<<day<<"日"<<
    endl;
        }
    };
```

```
class person{
private:
    char name[8];
    char sex[2];
    birth birdy;
public:
    person(char *p,char *q,int x,int y,int z);
    void show(){
        cout<<"姓名为"<<name<<"性别为"<<sex;
        birdy.show();
    }
};
person::person(char *p,char *q,int x,int y,int z):birdy(x,y,z){
    strcpy(name,p); strcpy(sex,q);
}
void main()
{
    person prsn("张三","男",2000,4,28);
    prsn.show();
}
```

有一个Person类，私有数据成员name、age和sex分别表示人的姓名、年龄和性别。雇员类Employee是Person的派生类，新增数据成员部门department和薪水salary。请用C++代码描述这两个类，并用Employee类的成员函数Display实现雇员的姓名、年龄、性别、部门和薪水的输出。(要求编写派生类的构造函数)

```
#include <iostream>
using namespace std;
class Person
{public:
Person(char *s1="",int a=0,char *s2="")
{strcpy(name,s1);age=a;strcpy(sex,s2);}
protected:
char name[8];
int age;
char sex[2];
};
```

```
class Employee:public Person
{public:
Employee(char *s1="",int a=0,char *s2="",char *s3="",double s=0.
0):Person(s1,a,s2)
{strcpy(department,s3);
salary=s;
}
void Display()
{cout<<name<<" "<<age<<" "<<sex<<" "<<department<<" "<<salary<<endl;}
private:
char department[20];
double salary;
};
void main()
{Employee ee("张三",18,"男","计算机",2180.8);
ee.Display();
}
```

祝大家顺利通过考试！