

XI`AN TECHNOLOGICAL UNIVERSITY

实验报告

实验课程名称 运算符重载

专 业：计算机科学与技术

班 级： 16060104

姓 名： 杨 旭

学 号： 16060104117

实验学时： 2

指导教师： 刘白林

成 绩：

2017 年 6 月 18 日

西安工业大学实验报告

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 专业 | 计算机科学与技术 | 班级 | 计科四班 | 姓名 | 杨旭 | 学号 | 16060104117 |
| 实验课程 | C++ | 指导教师 | 刘白林 | 实验日期 |  | 同实验者 |  |
| 实验项目 | 运算符重载 | | | | | | |
| 实验设备及器材 | 微型 | | | | | | |

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、实验学时

课内实验：2课时 课外练习：2课时

三 本实验涉及的新知识

㈠ 多态性的概念及分类

1．多态性的概念

多态性是指不同对象在收到相同的消息时，产生不同的动作。即用同一个名字定义不同的函数，执行不同但相似的操作，从而实现“一个接口，多种方法”。

2．多态性的分类及实现方法

⑴ 静态多态性。亦称编译时多态性，即静态联编支持的多态性，在C++中通过函数重载及运算符重载实现。

⑵ 动态多态性。亦称运行时多态性，即动态联编支持的多态性，在C++中通过虚函数实现。

3．函数重载的方法

由前面实验中所涉及的知识可知，函数重载主要有以下方式：

⑴ 同一个类中的同名函数，可用参数个数不一样或参数类型不一样或参数个数及类型不一样实现重载。

⑵ 同一类中，普通成员函数与同名的常量成员函数，可通过“const”实现重载。

⑶ 不同类中的同名函数，可通过类名调用或类的对象调用实现重载。

㈡ 运算符重载

1．运算符重载的意义及作用

C++中的运算符与函数一样，也可以实现重载，从而扩充C++预定义的运算符的作用范围。编程者可以通过重载运行符扩充运算符的功能，使编写的程序更符合人们的思维习惯，更易于理解。

2．重载的运算符

在C++中，大部分运算符可以重载，包括算术运算符、关系运算符、逻辑运算符、增量运算符、赋值运算符等。如：+、-、\*、/、%、&、=、+=、-=、&&、||、！、++、[]、new、delete等（除不能重载的几个运算符外，均可以重载）。

不能重载的运算符有：

：：—— 作用域运算符

. —— 成员访问运算符

.\* —— 成员指针访问运算符

?: —— 条件运算符

3．用成员函数重载运算符

⑴ 格式

<返回类型> <类名>::operator@(参数表)

{ 函数体 }

其中：返回类型通常为类名（也可以为其它类型）；operator是重载运算符必须使用的定义符；@ 表示所重载的运算符，如+、-、\*、/等。

⑵ 重载双目运算符后的调用格式

① 显示调用：obj1.operator@ (obj2) ;

② 隐式调用：obj1@obj2 ;

⑶ 重载单目运算符后的调用格式

① 显示调用：obj.operator@( ) ;

② 隐式调用：@obj ;

⑷ 说明

① 重载双目运算符时，运算符的左分量即为对象本身，用this指针表示，因此参数表中只有一个参数。

② 重载单目运算符时，惟一的运算分量是对象本身，用this指针表示，因此参数表中没有参数。

4．用友员函数重载运算符

⑴ 格式

friend <返回类型> operator@(参数表)

{ 函数体 }

⑵ 重载双目运算符后的调用格式

① 显示调用：operator @ (obj1,obj2) ;

② 隐式调用：obj1@obj2 ;

⑶ 重载单目运算符后的调用格式

① 显示调用： operator@( obj ) ;

② 隐式调用：@obj ;

⑷ 说明

由于友元运算符不是类的成员函数，无this指针，因此，当重载双目运算符时，参数表中有两个参数；当重载单目运算符时，参数表中有一个参数。

5．重载运算符应注意的问题

① 重载运算符不能改变原有运算符的含义。

② 重载运算符不能改变运算符的原有的优先级和运算分量的个数。

③ 只能重载已有的运算符，即不能新造一个运算符。

④ 同一个运算符可以重载多次，调用时可根据参数的个数及类型决定调用哪个重载函数。

⑤ 通常，单目运算符的重载用成员函数，双目运算符的重载用友元函数。

⑥ 运算符“=、( )、[ ]、->”的重载，只能用成员函数，不能用友元函数。

⑦ 当双目运算符的运算分量有一个基本类型数据时，通常用友元运算符重载。

⑧ 增量运算符重载时，前置运算符与前面介绍的知识相同，而后置运算符增加一个int作为形参。

⑨ 赋值运算符“=”通常不需要重载，但当类的数据成员中有指针变量，需要用new分配内存，用delete释放内存时，必须重载。

四、实验内容

㈠ 验证及认知实验

按要求调试下列程序，并回答相关问题。

程序1（exp\_801.cpp）

//用成员函数重载前置运算符“++”

#include <iostream.h>

class coord

{ int x,y;

public :

coord(int i=0,int j=0);

void print();

coord operator ++();

};

coord::coord(int i,int j)

{x=i;y=j;}

void coord::print()

{ cout << "x="<<x<<" y="<<y<<endl;}

coord coord::operator ++()

{ ++x;

++y;

return \*this;

}

void main()

{ coord ob(10,20);

ob.print();

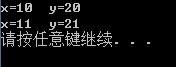
++ob;

ob.print();

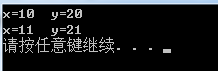
}

问题：

1. 编译运行程序的结果是：



1. 程序中前置运算符“++”采用的重载方式是 单目运算符重载 。
2. 将重载函数“coord coord::operator ++()”中的“++x; ++y; ”改为：“++this->x; ++this->y ; ” 重新运行程序。



1. 将“main()”函数中的“++ob;”改为“ob.operator++( );”，重新运算程序。

.

程序2（exp\_802.cpp）

//用友元函数重载两个复数相加：“+”

#include <iostream.h>

class complex

{ private :

double real ;

double imag;

public :

complex(double r=0,double i=0)

{real=r;imag=i;}

void print();

complex operator+(complex c1,complex c2);

};

complex operator+(complex c1,complex c2)//重载“+”

{ complex temp;

temp.real=c1.real+c2.real;

temp.imag=c1.imag+c2.imag;

return temp;

}

void complex::print()

{cout <<real;

if (imag>0) cout<<"+";

if (imag!=0) cout <<imag<<"i"<<endl;

}

void main()

{ complex A1(2.3,4.6),A2(3.6,2.8),A3;

A3=A1+A2;

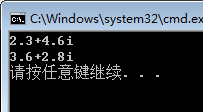
A1.print();A2.print();

}

问题：

1. 编译程序时，将会出现编译错误，其原因是：成员的运算符重载只能有一个参数 。

⑹ 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：



。

⑺ 将“main()”中的“A3=A1+A2;”改为“A3=operator(A1,A2);”，重新运行程序。

㈡ 知识应用实验

按要求调试、完善程序。

程序3（exp\_803.cpp）

//用成员函数重载两个复数相加、减

#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.imag = c.imag + imag;

temp.real = c.real + real;

return temp;

}

complex complex ::operator-(complex c)//重载“-”

{

complex temp;

temp.imag = c.imag - imag;

temp.real = c.real - real;

return temp;

}

void complex::print()

{

cout << real;

if (imag>0) cout << "+";

if (imag != 0) cout << imag << "i" << endl;

}

void 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();

}程序4（exp\_804.cpp）

//用成员函数重载前置、后置“++”及负号“-”

#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++

{

x++;

y++;

return \*this;

}

coord coord::operator -()//重载负号“-”

{

coord temp;

temp.x = 0 - x;

temp.y = 0 - y;

return temp;

}

void 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

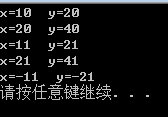
ob1.print();

ob2.print();

ob = -ob1;

ob.print();

}



程序5（exp\_805.cpp）

//先编译运行程序，分析程序运行时的出错原因，再将重载“=”的部分恢复为正常语句，

//并完善程序

#include <string.h>

class string

{ char \*ptr;

public :

string(char \*s)

{ptr=new char[strlen(s)+1];

strcpy(ptr,s);

}

~string()

{delete []ptr;}

void print()

{cout <<ptr<<endl;}

// string &operator=(const string&s);

};

/\*

string &string::operator=(const string&s)//重载“=”运算符

{ if(this==&s) return \*this; //当用“ob1=ob1;”时

//首先释放被赋值对象的空间

//重新为被赋值对象分配空间

strcpy(ptr,s.ptr);

return \*this;

}

\*/

void main()

出错原因，当析构函数销毁空间后，出现了野指针。

{string p1("chen");

{string p2(" ");

p2=p1;

p2.print();

}

p1.print();

}

㈢ 程序设计实验

完善实验六中Date类的定义。要求增加成员函数实现：

⑴ 能比较两个日期的大小（重载“= =”、“>”、“<”、“>=”、“<=”、“！=”）；

⑵ 编写main( )函数测试，内容自定。

提示：两个日期比较，先比较年、再比较月、再比较日，两个日期的年、月、日均相同才相等。

（日期类的定义存放在user\_date.h中，测试程序存放在exp\_806.cpp中）

#include <iostream>

using namespace std;

class date

{

protected:

int year;

int month;

int day;

public:

void set\_date(int y, int m, int d);

void print\_date();

friend int operator ==(const date &op,const date &op1);

friend int operator >(const date &op,const date &op1);

friend int operator <(const date &op,const date &op1);

friend int operator >=(const date &op, const date &op1);

friend int operator <=(const date &op, const date &op1);

friend int operator !=(const date &op, const date &op1);

};

int operator == (const date &op, const date &op1)

{

int flag = 0;

if (op.year == op1.year&&op.month == op1.month&&op.day == op1.day)

{

flag = 1;

}

return flag;

}

int operator >(const date &op, const date &op1)

{

int flag = 0;

if (op.year > op1.year&&op.month > op1.month&&op.day > op1.day)

{

flag = 1;

}

return flag;

}

int operator <(const date &op, const date &op1)

{

int flag = 0;

if (op.year < op1.year&&op.month < op1.month&&op.day < op1.day)

{

flag = 1;

}

return flag;

}

int operator >=(const date &op, const date &op1)

{

int flag = 0;

if (op.year >= op1.year&&op.month >= op1.month&&op.day >= op1.day)

{

flag = 1;

}

return flag;

}

int operator <=(const date &op, const date &op1)

{

int flag = 0;

if (op.year <= op1.year&&op.month <= op1.month&&op.day <= op1.day)

{

flag = 1;

}

return flag;

}

int operator !=(const date &op, const date &op1)

{

int flag = 0;

if (op.year != op1.year&&op.month != op1.month&&op.day!= op1.day)

{

flag = 1;

}

return flag;

}

void date::set\_date(int y, int m, int d)

{

year = y;

month = m;

day = d;

}

void date::print\_date()

{

if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0))

{

cout << year << '-' << month << '-' << day << "（闰年）" << endl;

}

else

{

cout << year << '-' << month << '-' << day << endl;

}

}

#include"date.h"

void main()

{

date a, b, c, d;

d.set\_date(1998, 2, 3);

c.set\_date(1994, 5, 6);

b.set\_date(1997, 8, 9);

a.set\_date(1999, 11, 12);

if (a > b)

{

cout << '>' << endl;

a.print\_date();

b.print\_date();

cout << "a真的大于b" << endl;}

else

{

cout << '>' << endl;

a.print\_date();

b.print\_date();

cout << "a不大于b" << endl;

}

if (a < b)

{

cout << '<' << endl;

a.print\_date();

b.print\_date();

cout << "a真的大于b" << endl;

}

else

{

cout << '<' << endl ;

a.print\_date();

b.print\_date();

cout << "a不大于b" << endl;

}

if (a == b)

{

cout <<"==" << endl;

a.print\_date();

b.print\_date();

cout << "a真的等于b" << endl;

}

else

{

cout << "==" << endl;

a.print\_date();

b.print\_date();

cout << "a不等于于b" << endl;

}

if (a != b)

{

cout << "!=" << endl;

a.print\_date();

b.print\_date();

cout << "a真的等于b" << endl;

}

else

{

cout << "!=" << endl;

a.print\_date();

b.print\_date();

cout << "a不等于于b" << endl;

}

if (a <= b)

{

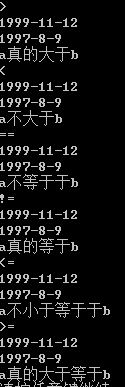
cout << "<=" << endl;

a.print\_date();

b.print\_date();

cout << "a真的小于等于b" << endl;

}



else

{

cout << "<=" << endl;

a.print\_date();

b.print\_date();

cout << "a不小于等于于b" << endl;

}

if (a >= b)

{

cout << ">=" << endl;

a.print\_date();

b.print\_date();

cout << "a真的大于等于b" << endl;

}

else

{

cout << ">=" << endl;

a.print\_date();

b.print\_date();

cout << "a不大于等于于b"<<endl;

}

system("pause");}