山东大学 软件 学院

c++程序设计 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201800301108 | 姓名：牛翔宇 | | 班级：18.4 |
| 实验题目：实验四 | | | |
| 实验学时：4 | | 实验日期： 5.11 | |
| 实验目的：  熟悉c++的类、对象。  熟悉c++的重载、继承。 | | | |
| 硬件环境：  PC机 | | | |
| 软件环境：  Dev-c++ | | | |
| 实验步骤与内容：  1、建立一个学生类，该类含有两个私有成员变量：学号、成绩，三个公有方法void set(int newID,int newGrade)用于设置学号和成绩，int getID()获得学号，int getGrade()获得成绩。  class Student{  private:  int ID,grade;  public:  void set(int newID,int newGrade){ID=newID;grade=newGrade;};  int getID(){return ID;};  int getGrade(){return grade;};  };  在main方法建立一个对象数组，内放5个Student类对象，用指针指向数组首元素，输出第1,3,5学生的数据。  int main(){  Student \*s=new Student[5];  s[0].set(1,70);s[1].set(2,80);s[2].set(3,100);s[3].set(4,90);s[4].set(5,85);  Student \*p;  p = &s[0] ;  //for(int i=0,j=1;i<=4;i+=2,j++){  for(int i=0;i<5;i++){  if(i!=1&&i!=3){  cout<<"第" << i+1 << "名学生的学号： "<<(p+i)->getID()<< " 成绩： "<<(p+i)->getGrade()<<endl;  }  }  bestStudent(s);  delete[] s;  return 0;  }运行结果：    建立一个函数bestStudent，用指向对象的指针作函数参数，在函数中找出5个学生中成绩最高者，并输出其学号。  void bestStudent(Student \*s){  Student stu; //建立一个Student对象，用于接下来比较时作交换媒介  for(int i=0;i<5;i++){  if(s[i].getGrade()>s[i+1].getGrade()){  stu=s[i];s[i]=s[i+1];s[i+1]=stu; //利用for循环遍历交换  } //成绩最高者为s[4]  }  cout << "成绩最高的学生的学号是： " << s[4].getID();  }  在主函数main()中，调用bestStudent。  int main(){  Student \*s=new Student[5];  s[0].set(1,70);s[1].set(2,80);s[2].set(3,100);s[3].set(4,90);  s[4].set(5,85);  bestStudent(s);  delete []s;  return 0;  }  运行结果：    --begin--cpp—code--  class Student{  private:  int ID,grade;  public:  void set(int newID,int newGrade){ID=newID;grade=newGrade;};  int getID(){return ID;};  int getGrade(){return grade;};  };  void bestStudent(Student \*s){  Student stu; //建立一个Student对象，用于接下来比较时作交换媒介  for(int i=0;i<5;i++){  if(s[i].getGrade()>s[i+1].getGrade()){  stu=s[i];s[i]=s[i+1];s[i+1]=stu; //利用for循环遍历交换  } //成绩最高者为s[4]  }  cout << "成绩最高的学生的学号是： " << s[4].getID();  }  int main(){  Student \*s=new Student[5];  s[0].set(1,70);s[1].set(2,80);s[2].set(3,100);s[3].set(4,90);s[4].set(5,85);  Student \*p;  p = &s[0] ;  //for(int i=0,j=1;i<=4;i+=2,j++){  for(int i=0;i<5;i++){  if(i!=1&&i!=3){  cout<<"第" << i+1 << "名学生的学号： "<<(p+i)->getID()<< " 成绩： "<<(p+i)->getGrade()<<endl;  }  }  bestStudent(s);  delete[] s;  return 0;  }  --end--cpp--code--  2、定义一个复数类Complex。私有成员变量：实部real，虚部imag，都是double类型。  class Complex{  private:  double real;  double imag;  public:  Complex(){real=0;imag=0;}  Complex(double r){real=r;imag=0;}  Complex(double r,double i){real=r;imag=i;}//三个构造函数  void show(){ //以复数形式输出的函数  if(real==0){  if(imag<0&&imag!=-1)  cout<<imag<<"i"<<endl;  else if(imag==-1)  cout<<"-i"<<endl;  else if(imag>0&&imag!=1)  cout<<imag<<"i"<<endl;  else if(imag==1)  cout<<"i"<<endl;  else  cout<<0<<endl;  }  else{  if(imag<0&&imag!=-1)  cout<<real<<imag<<"i"<<endl;  else if(imag==-1)  cout<<real<<"-i"<<endl;  else if(imag>0&&imag!=1)  cout<<real<<"+"<<imag<<"i"<<endl;  else if(imag==1)  cout<<real<<"+i"<<endl;  else  cout<<real<<endl;  }  }  double getReal(){return real;}//获得实部  double getImag(){return imag;}//获得虚部  void setReal(double r){real = r;}//设置实部  void setImag(double i){imag = i;}//设置虚部  friend Complex operator-(Complex &a,Complex &b);//友元函数运算符-重载  Complex operator\*(Complex &a);//成员函数运算符\*重载  Complex operator/(Complex &a);//成员函数运算符/重载  Complex operator++();//成员函数运算符++（前置）重载  Complex operator++(int);//成员函数++（后置）重载  Complex operator--();//成员函数运算符--（前置）重载  Complex operator--(int); //成员函数运算符--（后置）重载  friend ostream& operator << (ostream& output, Complex& c);//友元函数  //<<重载  friend istream& operator>>(istream& input,Complex &c); //友元函数  //>>重载  operator double(){return real;}//double类型转换运算符重载  friend Complex operator+(int a,Complex &b);//友元函数int+复数，+重载  Complex operator+(int a);//成员函数，复数+int，+重载  };   1. 重载运算符“+”，使之能用于复数的加法运算。将运算符函数重载为非成员、非友元的普通函数。   Complex operator+(Complex &a,Complex &b){  Complex t;  t.setReal(a.getReal()+b.getReal());  t.setImag(a.getImag()+b.getImag());  return t;  }   1. 重载运算符“-”，使之能用于复数的减法运算。将运算符函数重载为类的友元函数。   Complex operator-(Complex &a,Complex &b){  Complex t;  t.real=a.real-b.real;  t.imag=a.imag-b.imag;  return t;  }  要在类中用friend声明，否则无法访问私有成员变量。   1. 重载运算符“\*”,“/”，使之能用于复数的乘、除。运算符重载函数作为类的成员函数。   Complex Complex::operator\*(Complex &a){  Complex t;  t.real=real\*a.real-imag\*a.imag;  t.imag=imag\*a.real+real\*a.imag;  return t;  }  Complex Complex::operator/(Complex &a){  Complex t;  t.real=(real\*a.real+imag\*a.imag)/(a.real\*a.real+a.imag\*a.imag);  t.imag=(imag\*a.real-real\*a.imag)/(a.real\*a.real+a.imag\*a.imag);  return t;  }   1. 重载运算符“+”，使之能用于复数的加法运算。参加运算的两个运算量有一个是类对象，有一个是整数，顺序任意。   Complex Complex::operator+(int a){  Complex t;  t.real=real+a;  t.imag=imag;  return t;  }  Complex operator+(int a,Complex &b){  Complex t;  t.real=a+b.real;  t.imag=b.imag;  return t;  }  第一个函数用于复数+int，因为复数在前，int在后，所以可以直接把函数作为成员函数。第二个函数用于int+复数，因为int在前，复数在后，所以可以把函数作为友元函数，在类中用friend声明。   1. 实现实部自增、自减（都可以前置、后置）的运算。前置与后置的区别在于函数参数，前置函数没有参数，后置函数有参数int。前置自加/自减先自加/自减再返回自加/自减后的对象，后置自加/自减，返回自加/自减之前的对象，再自加/自减。   Complex Complex::operator++(){  ++real;  return \*this;  }  Complex Complex::operator++(int){  Complex t;  t.real=real;  t.imag=imag;  ++real;  return t;  }  Complex Complex::operator--(){  --real;  return \*this;  }  Complex Complex::operator--(int){  Complex t;  t.real=real;  t.imag=imag;  --real;  return t;  }   1. 重载流插入运算符“<<”和流提取运算符“>>”，使之能用于复数的输入和输出。   ostream& operator<<(ostream& output, Complex &c){  output<<c.real<<"+"<<c.imag<<"i"<<endl;  return output;  }  istream& operator>>(istream& input,Complex &c){  cout<<"请输入复数的实部和虚部："<<endl;  input>>c.real>>c.imag;  }  注意，重载流运算符不能声明为类成员函数，因为运算符左侧为流对象，所以声明为友元函数。   1. 在成员函数中包含重载类型转换运算符: operator double( ) { return real;} 在主程序中分别调用以上复数运算，并输出运算结果。在调用的地方加上注释，说明要验证什么运算。其中处理一个复数与一个double数相加的运算(与复数的实部相加)，结果存放在一个double型的变量d1中，输出d1的值。   int main(){  Complex a(1,1),b(2,2),c,d;  cout<<"复数加法运算"<<endl;  c=a+b;  c.show();  cout<<"复数减法运算"<<endl;  c=a-b;  c.show();  cout<<"复数乘法运算"<<endl;  c=a\*b;  c.show();  cout<<"复数除法运算"<<endl;  c=a/b;  c.show();  cout<<"复数+int"<<endl;  c=a+1;  c.show();  cout<<"int+复数"<<endl;  c=1+a;  c.show();  cout<<"复数实部后置自加"<<endl;  d=c++;  d.show();  cout<<"复数实部前置自加"<<endl;  d=++c;  d.show();  cout<<"复数实部后置自减"<<endl;  d=c--;  d.show();  cout<<"复数实部前置自减"<<endl;  d=--c;  d.show();  cout<<"输出复数"<<endl;  cout<<c;  cout<<"输入复数"<<endl;  cin>>c;  c.show();  double d1;  d1=double(a)+1.5; //double类型转换重载  cout<<d1;  return 0;  }运行结果：    3、①编写一个程序，声明抽象基类Shape，内含函数printArea()用于输出面积。  class Shape{  public:  virtual void printArea(){};  };  ②由Shape类派生出3个派生类： Circle(圆形)、Rectangle(矩形)、Triangle(三角形),重载printArea()方法。  class Circle:public Shape{  private:  double r;  public:  Circle(double radius){r=radius;}  void printArea(){cout<<3.14\*r\*r<<endl;}  };  class Rectangle:public Shape{  private:  double l;  double w;  public:  Rectangle(double length,double width){l=length;w=width;}  void printArea(){cout<<l\*w<<endl;}  };  class Triangle:public Shape{  private:  double h;  double w;  public:  Triangle(double height,double width){h=height;w=width;}  void printArea(){cout<<0.5\*h\*w<<endl;}  };  main方法创建三个图形的对象，并输出面积。  int main(){  Circle c(3);  Rectangle r(2,5);  Triangle t(4,6);  c.printArea();  r.printArea();  t.printArea();  }  --begin--cpp--code--  class Shape{  public:  virtual void printArea(){};  };  class Circle:public Shape{  private:  double r;  public:  Circle(double radius){r=radius;}  void printArea(){cout<<3.14\*r\*r<<endl;}  };  class Rectangle:public Shape{  private:  double l;  double w;  public:  Rectangle(double length,double width){l=length;w=width;}  void printArea(){cout<<l\*w<<endl;}  };  class Triangle:public Shape{  private:  double h;  double w;  public:  Triangle(double height,double width){h=height;w=width;}  void printArea(){cout<<0.5\*h\*w<<endl;}  };  int main(){  Circle c(3);  Rectangle r(2,5);  Triangle t(4,6);  c.printArea();  r.printArea();  t.printArea();  }  --end--cpp--code--  运行结果： | | | |
| 结论分析与体会：  1、实验一注意用new动态建立对象数组时，最后要delete分配的内存空间，否则这块内存空间将一直存在，浪费资源。  2、实验二中重载运算符要注意有的运算符只能重载为类的友元函数，比如流运算符>>和<<，因为他们左侧总是流对象，相当于是流对象调用运算符函数。还要注意参数的顺序，一般第一个参数对应于运算符左侧的对象，第二个函数对应于右侧的对象，在成员函数中一般只需要一个参数。注意前置自加自减和后置自加自减的区分，参数列表是否有int。  3、c++可以多继承，这一点与java不同，要注意区分。另外要注意对基类成员可见性的声明。 | | | |