1. **定义一个复数类Complex，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。编写程序，求两个复数之和。初值自拟。**

#include <iostream>

using namespace std;

class Complex

{public:

Complex(){real=0;imag=0;}

Complex(double r,double i){real=r;imag=i;}

double get\_real();

double get\_imag();

void display();

private:

double real;

double imag;

};

double Complex::get\_real()

{return real;}

double Complex::get\_imag()

{return imag;}

void Complex::display()

{cout<<"("<<real<<","<<imag<<"i)"<<endl;}

Complex operator + (Complex &c1,Complex &c2)

{

return Complex(c1.get\_real()+c2.get\_real(),c1.get\_imag()+c2.get\_imag());

}

int main()

{ Complex c1(3,4),c2(5,-10),c3;

c3=c1+c2;

cout<<"c3=";

c3.display();

return 0;

}

**2.定义一个复数类Complex，重载运算符“**+**”，“**-**”，使之能用于复数的加，减运算，运算符重载函数作为Complex类的成员函数。编程序，分别求出两个复数之和，差。初值自拟。**

using namespace std;

class Complex

{public:

Complex(){real=0;imag=0;}

Complex(doubler,double i){real=r;imag=i;}

Complex operator+(Complex &c2);

Complex operator-(Complex &c2);

void display();

private:

double real;

double imag;

};

Complex Complex::operator+(Complex &c2)

{ Complex c;

c.real=real+c2.real;

c.imag=imag+c2.imag;

return c;}

Complex Complex::operator-(Complex &c2)

{ Complex c;

c.real=real-c2.real;

c.imag=imag-c2.imag;

return c;}

void Complex::display()

{cout<<"("<<real<<","<<imag<<"i)"<<endl;}

int main()

{

Complex c1(3,4),c2(5,-10),c3;

c3=c1+c2;

cout<<"c1+c2=";

c3.display();

c3=c1-c2;

cout<<"c1-c2=";

c3.display();

return 0;

}

**3.定义一个复数类Complex，重载运算符 “\*”，“/”，使之能用于复数的乘，除。运算符重载函数作为Complex类的成员函数。编程序，分别求出两个复数之积和商。初值自拟。提示：两复数相乘的计算公式为：(a+bi)\*(c+di)=(ac-bd)+(ad+bc)i。两复数相除的计算公式为：(a+bi)/(c+di)=(ac+bd)/(c\*c+d\*d)+(bc-ad) /(c\*c+d\*d)i。**

#include <iostream>

using namespace std;

class Complex

{public:

Complex(){real=0;imag=0;}

Complex(double r,double i){real=r;imag=i;}

Complex operator\*(Complex &c2);

Complex operator/(Complex &c2);

void display();

private:

double real;

double imag;

};

Complex Complex::operator\*(Complex &c2)

{ Complex c;

c.real=real\*c2.real-imag\*c2.imag;

c.imag=imag\*c2.real+real\*c2.imag;

return c;}

Complex Complex::operator/(Complex &c2)

{ Complex c;

c.real=(real\*c2.real+imag\*c2.imag)/(c2.real\*c2.real+c2.imag\*c2.imag);

c.imag=(imag\*c2.real-real\*c2.imag)/(c2.real\*c2.real+c2.imag\*c2.imag);

return c;}

void Complex::display()

{cout<<"("<<real<<","<<imag<<"i)"<<endl;}

int main()

{ Complex c1(3,4),c2(5,-10),c3;

c3=c1\*c2;

cout<<"c1\*c2=";

c3.display();

c3=c1/c2;

cout<<"c1/c2=";

c3.display();

return 0;

}

**4.定义一个复数类Complex，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可以都是类对象，也可以其中有一个是整数，顺序任意。例如：c1+c2，i+c1，c1+i均合法（设i为整数，c1，c2为复数）。编程序，分别求两个复数之和、整数和复数之和。初值自拟。**

#include <iostream.h>

class Complex

{public:

Complex(){real=0;imag=0;}

Complex(double r,double i){real=r;imag=i;}

Complex operator+(Complex &c2);

Complex operator+(int &i);

friend Complex operator+(int&,Complex &);

void display();

private:

double real;

double imag;

};

Complex Complex::operator+(Complex &c)

{return Complex(real+c.real,imag+c.imag);}

Complex Complex::operator+(int &i)

{return Complex(real+i,imag);}

void Complex::display()

{cout<<"("<<real<<","<<imag<<"i)"<<endl;}

Complex operator+(int &i,Complex &c)

{return Complex(i+c.real,c.imag);}

int main()

{ Complex c1(3,4),c2(5,-10),c3;

int i=5;

c3=c1+c2;

cout<<"c1+c2=";

c3.display();

c3=i+c1;

cout<<"i+c1=";

c3.display();

c3=c1+i;

cout<<"c1+i=";

c3.display();

return 0;

}

**5.有两个矩阵a和b，均为2行3列。求两个矩阵之和。重载运算符“+”，使之能用于矩阵相加。如c=a+b。初值自拟。**

#include <iostream.h>

class Matrix

{public:

Matrix();

friend Matrix operator+(Matrix &,Matrix &);

void input();

void display();

private:

int mat[2][3];

};

Matrix::Matrix()

{ for(int i=0;i<2;i++)

for(int j=0;j<3;j++)

mat[i][j]=0;

}

Matrix operator+(Matrix &a,Matrix &b)

{Matrix c;

for(int i=0;i<2;i++)

for(int j=0;j<3;j++)

{c.mat[i][j]=a.mat[i][j]+b.mat[i][j];}

return c;

}

void Matrix::input()

{cout<<"input value of matrix:"<<endl;

for(int i=0;i<2;i++)

for(int j=0;j<3;j++)

cin>>mat[i][j];

}

void Matrix::display()

{for (int i=0;i<2;i++)

{for(int j=0;j<3;j++)

{cout<<mat[i][j]<<" ";}

cout<<endl;}

}

int main()

{ Matrix a,b,c;

a.input();

b.input();

cout<<endl<<"Matrix a:"<<endl;

a.display();

cout<<endl<<"Matrix b:"<<endl;

b.display();

c=a+b;

cout<<endl<<"Matrix c = Matrix a + Matrix b :"<<endl;

c.display();

return 0;}

**6.定义一个字符串类String，用来存放不定长的字符串，重载运算符“＝＝”，，用于两个字符串的等于比较运算。初值自拟。**

#include <iostream.h>

#include <string.h>

class String

{public:

String(){p=NULL;}

String(char \*str);

friend bool operator==(String &string1,String &string2);

void display();

private:

char \*p;

};

String::String(char \*str)

{p=str;}

void String::display()

{cout<<p;}

bool operator==(String &string1,String &string2)

{if(strcmp(string1.p,string2.p)==0)

return true;

else

return false;

}

void compare(String &string1,String &string2)

{if(operator==(string1,string2)==1)

{string1.display();cout<<"=";string2.display();}

cout<<endl;

}

int main()

{ String string1("Hello"),string2("Hello");

compare(string1,string2);

return 0;

}

1. **定义一个描述学生基本情况的类，数据成员包括姓名、学号、C++成绩、英语和数学成绩，成员函数包括输出数据，求出总成绩和平均成绩。数据自拟。**

#include"string.h"

#include <iostream.h>

class CStuScore

{public:

char strName[12];

char strStuNO[9];

void SetScore( char sname[12], char NO[9],float s0, float s1, float s2)

{

strcpy(strName, sname);

strcpy(strStuNO, NO);

fScore[0] = s0;

fScore[1] = s1;

fScore[2] = s2; }

void print()

{ cout<<

cout<<"姓名:"<<strName;

cout<<"学号:"<<strStuNO;

cout<<" C++成绩:"<<fScore[0]<<"英语成绩："<<fScore[1]<<"数学成绩："<<fScore[2]<<endl; }

float GetSUM()

{ return (float)((fScore[0] + fScore[1] + fScore[2])); }

Float GetAverage();

private:

Float fScore[3];

};

float CStuScore::GetAverage()

{ return (float)((fScore[0] + fScore[1] + fScore[2])/3.0); }

void main()

{ CStuScore one;

float a,b,c;

char Name[12];

char StuNO[9];

cout<<"姓名:";

cin>>Name;

cout<<"学号:";

cin>>StuNO;

cout<<" 成绩1:"<<" 成绩2： "<<" 成绩3： "<<"\n";

cin>>a>>b>>c;

one.SetScore(Name,StuNO,a,b,c);

one.print();

cout<<"平均成绩为 "<<one.GetAverage()<<"\n";

cout<<"总成绩"<<one.GetSUM()<<"\n";}

1. **先建立一个Point（点）类，包含数据成员x，y（坐标点）。以它为基类，派生出一个Circle（圆）类，增加数据成员r（半径），再以Circle类为直接基类，派生出一个Cylinder（圆柱体）类，在增加数据成员h（高）。编写程序，重载运算符“<<”和“>>”，使之能够用于输出以上类对象。**

#include <iostream.h>

class Point

{public:

Point(float=0,float=0);

void setPoint(float,float);

float getX() const {return x;}

float getY() const {return y;}

friend ostream & operator<<(ostream &,const Point &);

protected:

float x,y;

};

Point::Point(float a,float b)

{x=a;y=b;}

void Point::setPoint(float a,float b)

{x=a;y=b;}

ostream & operator<<(ostream &output,const Point &p)

{output<<"["<<p.x<<","<<p.y<<"]"<<endl;

return output;

}

class Circle:public Point

{public:

Circle(float x=0,float y=0,float r=0);

void setRadius(float);

float getRadius() const;

float area () const;

friend ostream &operator<<(ostream &,const Circle &);

protected:

float radius;

};

Circle::Circle(float a,float b,float r):Point(a,b),radius(r){}

void Circle::setRadius(float r)

{radius=r;}

float Circle::getRadius() const {return radius;}

float Circle::area() const

{return 3.14159\*radius\*radius;}

ostream &operator<<(ostream &output,const Circle &c)

{output<<"Center=["<<c.x<<","<<c.y<<"], r="<<c.radius<<", area="<<c.area()<<endl;

return output;

}

class Cylinder:public Circle

{public:

Cylinder (float x=0,float y=0,float r=0,float h=0);

void setHeight(float);

float getHeight() const;

float area() const;

float volume() const;

friend ostream& operator<<(ostream&,const Cylinder&);

protected:

float height;

};

Cylinder::Cylinder(float a,float b,float r,float h)

:Circle(a,b,r),height(h){}

void Cylinder::setHeight(float h){height=h;}

float Cylinder::getHeight() const {return height;}

float Cylinder::area() const

{ return 2\*Circle::area()+2\*3.14159\*radius\*height;}

float Cylinder::volume() const

{return Circle::area()\*height;}

ostream &operator<<(ostream &output,const Cylinder& cy)

{output<<"Center=["<<cy.x<<","<<cy.y<<"], r="<<cy.radius<<", h="<<cy.height

<<"\narea="<<cy.area()<<", volume="<<cy.volume()<<endl;

return output;

}

int main()

{ Cylinder cy1(3.5,6.4,5.2,10);

cout<<"\noriginal cylinder:\nx="<<cy1.getX()<<", y="<<cy1.getY()<<", r="

<<cy1.getRadius()<<", h="<<cy1.getHeight()<<"\narea="<<cy1.area()

<<", volume="<<cy1.volume()<<endl;

cy1.setHeight(15);

cy1.setRadius(7.5);

cy1.setPoint(5,5);

cout<<"\nnew cylinder:\n"<<cy1;

Point &pRef=cy1;

cout<<"\npRef as a point:"<<pRef;

Circle &cRef=cy1;

cout<<"\ncRef as a Circle:"<<cRef;

return 0;

}

1. **写一个程序，定义抽象类型Shape，由他派生三个类：Circle（圆形），Rectangle（矩形），Trapezoid（梯形），用一个函数printArea分别输出三者的面积，3个图形的数据在定义对象是给定。**

#include <iostream>

using namespace std;

class Shape

{public:

virtual double area() const =0; };

class Circle:public Shape

{public:

Circle(double r):radius(r){}

virtual double area() const {return 3.14159\*radius\*radius;

protected:

double radius; };

class Rectangle:public Shape

{public:

Rectangle(double w,double h):width(w),height(h){}

virtual double area() const {return width\*height;} protected:

double width,height;

};

class Trapezoid:public Shape

{public:

Trapezoid(double w,double h,double len):width(w),height(h),length(len){}

virtual double area() const {return 0.5\*height\*(width+length);}

protected:

double width,height,length;

};

void printArea(const Shape &s)

{cout<<s.area()<<endl;}

int main()

{

Circle circle(12.6);

cout<<"area of circle =";

printArea(circle);

Rectangle rectangle(4.5,8.4);

cout<<"area of rectangle =";

printArea(rectangle);

Trapezoid trapezoid(4.5,8.4,8.0);

cout<<"area of trapezoid =";

printArea(trapezoid);

return 0;

}

1. **定义一个人员类Cperson，包括数据成员：姓名、编号、性别和用于输入输出的成员函数。在此基础上派生出学生类CStudent(增加成绩)和老师类Cteacher(增加教龄)，并实现对学生和教师信息的输入输出。**

#include <iostream.h>

#include <string.h>

class CPerson

{

public:

void SetData(char \*name, char \*id, bool isman = 1)

{ int n = strlen(name);

strncpy(pName, name, n); pName[n] = '\0';

n = strlen(id);

strncpy(pID, id, n);

pID[n] = '\0';

bMan = isman;

}

void Output()

{

cout<<"姓名："<<pName<<endl;

cout<<"编号："<<pID<<endl;

char \*str = bMan?"男":"女";

cout<<"性别："<<str<<endl;

}

private:

char pName[20];

char pID[20];

bool bMan;

};

class CStudent: public CPerson

{

public:

void InputScore(double score1, double score2, double score3)

{

dbScore[0] = score1;

dbScore[1] = score2;

dbScore[2] = score3;

}

void Print()

{

Output();

for (int i=0; i<3; i++)

cout<<"成绩"<<i+1<<"："<<dbScore[i]<<endl;

}

private:

double dbScore[3];

};

class Cteacher: public CPerson

{

public:

void Inputage(double age)

{tage = age;}

void Print()

{

Output();

cout<<"教龄："<<tage<<endl;

}

private:

double tage;

};

void main()

{

CStudent stu;

Cteacher tea;

stu.SetData("LiMing", "21010211");

stu.InputScore( 80, 76, 91 );

stu.Print();

tea.SetData("zhangli","001");

tea.Inputage(12);

tea.Print();

}

1. **某商店经销一种货物，货物成箱购进，成箱卖出，购进和卖出时以重量为单位，各箱的重量不一样，因此，商店需要记下目前库存货物的总量，要求把商店货物购进和卖出的情况模拟出来。**

#include<iostream>

using namespace std;

class Goods

{ public :

Goods ( int w) { weight = w; totalWeight += w ; } ; ~ Goods ( ) { totalWeight -= weight ; } ;

int Weight ( ) { return weight ; } ;

static int TotalWeight ( ) { return totalWeight ; }

private : int weight ;

static int totalWeight ;

} ;

int Goods :: totalWeight = 0 ;

main ( )

{ int w ;

cin >> w ;

Goods \*g1 = new Goods( w ) ;

cin >> w ;

Goods \*g2 = new Goods( w ) ;

cout << Goods::TotalWeight ( ) << endl ;

delete g2 ;

cout << Goods::TotalWeight ( ) << endl ;

}

1. **设计一个Time类，包括三个私有数据成员：hour,minute,sec，用构造函数初始化,内设公用函数display(Date &d),设计一个Date类，包括三个私有数据成员：month,day,year，也用构适函数初始化；分别定义两个带参数的对象t1(12,30,55),d1(3,25,2010)，通过友员成员函数的应用，输出d1和t1的值。**

#include <iostream>

using namespace std;

class Date;

class Time

{public:

Time(int,int,int);

void display(const Date&);

private:

int hour;

int minute;

int sec;

};

Time::Time(int h,int m,int s)

{ hour=h;

minute=m;

sec=s;

}

class Date

{public:

Date(int,int,int);

friend void Time::display(const Date &);

private:

int month;

int day;

int year;

};

Date::Date(int m,int d,int y)

{ month=m;

day=d;

year=y;

}

void Time::display(const Date &da)

{cout<<da.month<<"/"<<da.day<<"/"<<da.year<<endl;

cout<<hour<<":"<<minute<<":"<<sec<<endl;

}

int main()

{ Time t1(12,30,55);

Date d1(3,25,2010);

t1.display(d1);

return 0;

}

1. **定义点类（Point），再定义一个类（Distance）描述两点之间的距离，其数据成员为两个点类对象，两点之间距离的计算可设计由构造函数来实现。**

#include<iostream>

#include<math.h>

using namespace std;

class Point

{public :

Point ( double xi , double yi ) { X = xi ; Y = yi ; }

double GetX( ) { return X ; }

double GetY( ) { return Y ; }

private : double X , Y ;

} ;

class Distance

{

public :

Distance(Point p,Point q);

double Getdist() {return dist;}

private:

Point a,b;

double dist;

};

Distance::Distance(Point q1,Point q2):a(q1),b(q2)

{ double dx=double(a.GetX()-b.GetX());

double dy=double(a.GetY()-b.GetY());

dist=sqrt(dx\*dx+dy\*dy);

}

int main ( )

{ Point p1 ( 3.0 , 5.0 ) , p2 ( 4.0 , 6.0 ) ;

Distance dis(p1,p2);

cout << "This distance is: "<< dis.Getdist()<<endl ;

return 0;}

1. **定义点类（Point），再定义一个友元函数（Distance）描述两点之间的距离，其数据成员为两个点类对象，将两点之间距离函数声明为Point类的友元函数。**

#include<iostream>

#include<math.h>

using namespace std;

class Point

{ public :

Point ( double xi , double yi ) { X = xi ; Y = yi ; }

double GetX( ) { return X ; }

double GetY( ) { return Y ; }

friend double Distance ( Point & a , Point & b ) ;

private : double X , Y ;

} ;

double Distance ( Point & a , Point & b )

{ double dx = a.X - b.X ;

double dy = a.Y - b.Y ;

return sqrt( dx \* dx + dy \* dy ) ;

}

int main ( )

{ Point p1 ( 3.0 , 5.0 ) , p2 ( 4.0 , 6.0 ) ;

double d = Distance ( p1 , p2 ) ;

cout << "This distance is " << d << endl ;

return 0;}

1. **有一个Time类，包含数据成员minute（分）和sec（秒），模拟秒表，每次走一秒，满60秒进一分钟，此时秒又从0开始算。要求输出分和秒的值。初值自拟。**

#include <iostream>

using namespace std;

class Time

{public:

Time(){minute=0;sec=0;}

Time(int m,int s):minute(m),sec(s){}

Time operator++();

void display(){cout<<minute<<":"<<sec<<endl;}

private:

int minute;

int sec;

};

Time Time::operator++()

{ if(++sec>=60)

{sec-=60;

++minute;}

return \*this;

}

int main()

{ Time time1(34,0);

for (int i=0;i<61;i++)

{++time1;

time1.display();}

return 0;

}

1. **声明一个教师（Teacher）类和一个学生（Student）类，用多重继承的方式声明一个研究生（Graduate）派生类。教师类中包括数据成员name（姓名），age（年龄），title（职称）。学生类中包括数据成员name（姓名），age（年龄），score（成绩）。在定义派生类对象时给出初始化的数据（自已定），然后输出这些数据。初值自拟。**

#include <iostream>

#include <string>

using namespace std;

class Teacher

{public:

Teacher(string nam,int a,string t)

{ name=nam;

age=a;

title=t;}

void display()

{cout<<"name:"<<name<<endl;

cout<<"age"<<age<<endl;

cout<<"title:"<<title<<endl;

}

protected:

string name;

int age;

string title; };

class Student

{public:

Student(string nam,char s,float sco)

{ name1=nam;

sex=s;

score=sco;} void display1()

{ cout<<"name:"<<name1<<endl;

cout<<"sex:"<<sex<<endl;

cout<<"score:"<<score<<endl;

}

protected:

string name1;

char sex;

float score;

};

class Graduate:public Teacher,public Student

{public:

Graduate(string nam,int a,char s,string t,float sco,float w):Teacher(nam,a,t),Student(nam,s,sco),wage(w) {}

void show( ) {cout<<"name:"<<name<<endl;

cout<<"age:"<<age<<endl;

cout<<"sex:"<<sex<<endl;

cout<<"score:"<<score<<endl;

cout<<"title:"<<title<<endl;

cout<<"wages:"<<wage<<endl;

}

private:

float wage;

};

int main( )

{ Graduate grad1("Wang-li",24,'f',"assistant",89.5,1234.5);

grad1.show( );

return 0;

}

1. **在上题的基础上，在Teacher类和Student类之上增加一个共同的基类Person，如下图所示。作为人员的一些基本数据都放在Person中，在Teacher类和Student类中再增加一些必要的数据（Student类中增加score,Teacher类中增加职称title,Graduate类中增加工资wages）。初值自拟。**

Person

Student

Teacher

Graduate

#include <iostream>

#include <string>

using namespace std;

class Person

{public:

Person(char \*nam,char s,int a)

{strcpy(name,nam);sex=s;age=a;}

protected:

char name[20];

char sex;

int age;

};

class Teacher:virtual public Person

{public:

Teacher(char \*nam,char s,int a,char \*t):Person(nam,s,a) {strcpy(title,t);

}

protected:

char title[10];

};

class Student:virtual public Person

{public:

Student(char \*nam,char s,int a,float sco):Person(nam,s,a),score(sco){}

protected:

float score;

};

class Graduate:public Teacher,public Student

{public:

Graduate(char \*nam,char s,int a,char \*t,float sco,float w):Person(nam,s,a),Teacher(nam,s,a,t),Student(nam,s,a,sco),wage(w){}

void show( ) {cout<<"name:"<<name<<endl;

cout<<"age:"<<age<<endl;

cout<<"sex:"<<sex<<endl;

cout<<"score:"<<score<<endl;

cout<<"title:"<<title<<endl;

cout<<"wages:"<<wage<<endl;

}

private:

float wage;

};

int main( )

{ Graduate grad1("Wang-li",'f',24,"assistant",89.5,1234.5);

grad1.show( );

return 0;

}

1. **声明一个Shape抽象类，在此基础上派生出Rectangle和Circle类，二者都有GetArea( )函数计算对象的面积，GetPerim( )函数计算对象的周长。**

#include<iostream>

using namespace std;

class Shape

{public:

Shape(){}

~Shape(){}

virtual float GetPerim()=0;

virtual float GetArea()=0;

};

class Rectangle:public Shape

{public:

Rectangle(float i,float j):L(i),W(j){}

~Rectangle(){}

float GetPerim(){return 2\*(L+W);}

float GetArea(){return L\*W;}

private:

float L,W;

};

class Circle:public Shape

{public:

Circle(float r):R(r){}

float GetPerim(){return 3.14\*2\*R;}

float GetArea(){return 3.14\*R\*R;}

private:

float R;

};

void main()

{ Shape \* sp;

sp=new Circle(10);

cout<<sp->GetPerim ()<<endl;

cout<<sp->GetArea()<<endl;

sp=new Rectangle(6,4);

cout<<sp->GetPerim()<<endl;

cout<<sp->GetArea()<<endl;

}

1. **定义一个国家基类Country，包含国名、首都、人口等属性，派生出省类Province，增加省会城市、人口数量属性。**

#include<iostream>

#include<string>

using namespace std;

class Country

{public:

Country(string nam,string c,long int cp)

{name=nam;capital=c;country\_population=cp;}

protected:

string name;

string capital;

long int country\_population;

};

class Province:public Country

{public:

Province(string nam,string c,long int cp,string pc,long int pp):Country(nam,c,cp)

{Province\_capital=pc;

Province\_population=pp;

};

void show()

{cout<<"name:"<<name<<endl;

cout<<"capital:"<<capital<<endl;

cout<<"country\_population:"<<country\_population<<endl;

cout<<"Province\_capital:"<<Province\_capital<<endl;

cout<<"Province\_population:"<<Province\_population<<endl;

}

private:

string Province\_capital;

long int Province\_population;

};

int main()

{ Province prov1("China","Beijing",1300000000,"Nanchang",45000000);

prov1.show();

return 0;

}

1. **定义一个车基类Vehicle，含私有成员speed，weight。派生出自行车类Bicycle，增加high成员；汽车类Car，增加seatnum(座位数)成员。从bicycle和car中派生出摩托车类Motocycle。**

#include<iostream>

using namespace std;

class Vehicle

{public:

Vehicle(float sp,float w){speed=sp;weight=w;}

void display(){cout<<"speed:"<<speed<<" weight"<<weight<<endl;}

private:

float speed;

float weight;

};

class Bicycle:virtual public Vehicle

{public:

Bicycle(float sp,float w,float h):Vehicle(sp,w){high = h;}

float high;

};

class Car:virtual public Vehicle

{public:

Car(float sp,float w,int num):Vehicle(sp,w)

{seatnum = num;

}

protected:

int seatnum;

};

class Motorcycle:public Bicycle,public Car

{public:

Motorcycle(float sp,float w,float h,int num):Vehicle(sp,w),Bicycle(sp,w,h),Car(sp,w,num){}

void display(){

Vehicle::display();

cout<<" high:"<<high<<" seatnum:"<<seatnum<<endl;

}

};

int main()

{ Motorcycle m(120,120,120,1);

m.display();

return 0; }

1. **声明一个哺乳动物Mammal类，再由此派生出狗Dog类，二者都定义Speak( )成员函数，基类中定义为虚函数。声明一个Dog类的对象，调用Speak()函数，观察运行结果。**

#include <iostream.h>

class Mammal

{

public:

Mammal():itsAge(1) { cout << "Mammal constructor"<<endl; }

~Mammal() { cout << "Mammal destructor"<<endl; }

virtual void Speak() const { cout << "Mammal speak!"<<endl; }

private:

int itsAge;

};

class Dog : public Mammal

{

public:

Dog() { cout << "Dog Constructor"<<endl; }

~Dog() { cout << "Dog destructor"<<endl; }

void Speak() const { cout << "Woof!"<<endl; }

};

int main()

{

Mammal \*pDog = new Dog;

pDog->Speak();

delete pDog;

return 0;

}

1. **商店销售某一商品，商店每天公布统一的折扣（discount）。同时允许销售人员在销售时灵活掌握售价（price），在此基础上，对一次购10件以上者，还可以享受9.8折优惠。现已知当天3名销货员的销售情况为：**

**销货员 销货件 数销售单价**

**101 5 23.5**

**102 12 24.56**

**103 100 21.5**

**请编程序，计算出当日此商品的总销售款sum，以及每件商品的平均售价。要求用静态数据成员和静态成员函数。（提示：将折扣discount、总销售款sum和商品销售总件数n声明为静态数据成员，再定义静态成员函数average（求平均售价）和display（输出结果）。**

#include <iostream>

using namespace std;

class Product

{public:

Product(int n,int q,float p):num(n),quantity(q),price(p){};

void total();

static float average();

static void display();

private:

int num;

int quantity;

float price;

static float discount;

static float sum;

static int n;

};

void Product::total()

{float rate=1.0;

if(quantity>10) rate=0.98\*rate;

sum=sum+quantity\*price\*rate\*(1-discount);

n=n+quantity;

}

void Product::display()

{cout<<sum<<endl;

cout<<average()<<endl; }

float Product::average()

{return(sum/n);}

float Product::discount=0.05;

float Product::sum=0;

int Product::n=0;

int main()

{Product rod[3]={ Product(101,5,23.5),Product(102,12,24.56),Product(103,100,21.5)};

for(int i=0;i<3;i++)

Prod[i].total();

Product::display();

return 0; }

1. **定义一个Teacher（教师）类和一个Student（学生）类，二者有一部分数据成员是相同的，例如num（号码），name（姓名），sex（性别）。编写程序，将一个Student对象（学生）转换为Teacher（教师）类，只将以上3个相同的数据成员移植过去。可以设想为：一位学生大学毕业了，留校担任教师，他原有的部分数据对现在的教师身份来说仍然是有用的，应当保留并成为其教师的数据的一部分。**

#include <iostream>

using namespace std;

class Student

{public:

Student(int,char[],char,float);

int get\_num(){return num;}

char \* get\_name(){return name;}

char get\_sex(){return sex;}

void display()

{cout<<"num:"<<num<<"\nname:"<<name<<"\nsex:"<<sex<<"\nscore:"<<score<<"\n\n";}

private:

int num;

char name[20];

char sex;

float score;

};

Student::Student(int n,char nam[],char s,float so)

{ num=n;

strcpy(name,nam);

sex=s;

score=so;}

class Teacher

{public:

Teacher(){}

Teacher(Student&);

Teacher(int n,char nam[],char sex,float pay);

void display();

private:

int num;

char name[20];

char sex;

float pay; };

Teacher::Teacher(int n,char nam[],char s,float p)

{ num=n;

strcpy(name,nam);

sex=s;

pay=p;}

Teacher::Teacher(Student& stud)

{ num=stud.get\_num();

strcpy(name,stud.get\_name());

sex=stud.get\_sex();

pay=1500;}

void Teacher::display()

{cout<<"num:"<<num<<"\nname:"<<name<<"\nsex:"<<sex<<"\npay:"<<pay<<"\n\n";}

int main()

{ Teacher teacher1(10001,"Li",'f',1234.5),teacher2;

Student student1(20010,"Wang",'m',89.5);

cout<<"student1:"<<endl;

student1.display();

teacher2=Teacher(student1);

cout<<"teacher2:"<<endl;

teacher2.display();

return 0;}

**24**、**分别定义Teacher（教师）类和Cadre（干部）类，采用多重继承方式有这两个类派生出新类Teacher\_Cadre（教师兼干部）。要求：**

**(1)、在两个基类中的包含姓名、年龄、性别、地址、电话、等数据成员。**

**(2)、在Teacher类中包含数据成员title（职称），在Cadre类中还包含数据成员post（职务），在Teacher\_Cadre类中还包含数据成员wages（工资）。**

**(3)、对两个基类中的姓名、年龄、性别、职称、地址、电话等数据成员用相同的名字，在引用数据成员时制定作用域。**

**(4)、在类中声明成员函数，在类外定义成员函数 。**

**(5)、在派生类Teacher\_cadre的成员函数show中调用Teacher类中的display函数。输出姓名，年龄，性别，职称，地址，电话，然后再用cout语句输出职务与工资。**

#include<string>

#include <iostream>

using namespace std;

class Teacher

{public:

Teacher(string nam,int a,char s,string tit,string ad,string t);

void display();

protected:

string name;

int age;

char sex;

string title;

string addr;

string tel;

};

Teacher::Teacher(string nam,int a,char s,string tit,string ad,string t):

name(nam),age(a),sex(s),title(tit),addr(ad),tel(t){ }

void Teacher::display()

{cout<<"name:"<<name<<endl;

cout<<"age"<<age<<endl;

cout<<"sex:"<<sex<<endl;

cout<<"title:"<<title<<endl;

cout<<"address:"<<addr<<endl;

cout<<"tel:"<<tel<<endl; }

class Cadre

{public:

Cadre(string nam,int a,char s,string p,string ad,string t);

void display();

protected:

string name;

int age;

char sex;

string post;

string addr;

string tel; };

Cadre::Cadre(string nam,int a,char s,string p,string ad,string t):

name(nam),age(a),sex(s),post(p),addr(ad),tel(t){}

void Cadre::display()

{cout<<"name:"<<name<<endl;

cout<<"age:"<<age<<endl;

cout<<"sex:"<<sex<<endl;

cout<<"post:"<<post<<endl;

cout<<"address:"<<addr<<endl;

cout<<"tel:"<<tel<<endl; }

class Teacher\_Cadre:public Teacher,public Cadre

{public:

Teacher\_Cadre(string nam,int a,char s,string tit,string p,string ad,string t,float w);

void show( );

private:

float wage; };

Teacher\_Cadre::Teacher\_Cadre(string nam,int a,char s,string t,string p,string ad,string tel,float w):

Teacher(nam,a,s,t,ad,tel),Cadre(nam,a,s,p,ad,tel),wage(w) {}

void Teacher\_Cadre::show( )

{Teacher::display();

cout<<"post:"<<Cadre::post<<endl;

cout<<"wages:"<<wage<<endl; }

int main( )

{Teacher\_Cadre te\_ca("Wang-li",50,'f',"prof.","president","135 Beijing Road,Shanghai","(021)61234567",1534.5);

te\_ca.show( );

return 0;}

**25、写一个程序，定义抽象类型Shape，由他派生五个类：Circle（圆形），Square（正方形），Rectangle（矩形），Trapezoid（梯形），Triangle（三角形）。用虚函数分别计算几种图形的面积，并求它们的和。要求用基类指针数组，使它的每一个元素指向一个派生类的对象。**

#include <iostream>

using namespace std;

class Shape

{public:

virtual double area() const =0; };

class Circle:public Shape

{public:

Circle(double r):radius(r){}

virtual double area() const {return 3.14159\*radius\*radius;};

protected:

double radius; };

class Square:public Shape

{public:

Square(double s):side(s){}

virtual double area() const {return side\*side;}

protected:

double side;};

class Rectangle:public Shape

{public:

Rectangle(double w,double h):width(w),height(h){}

virtual double area() const {return width\*height;}

protected:

double width,height; };

class Trapezoid:public Shape

{public:

Trapezoid(double t,double b,double h):top(t),bottom(t),height(h){}

virtual double area() const {return 0.5\*(top+bottom)\*height;}

protected:

double top,bottom,height; };

class Triangle:public Shape

{public:

Triangle(double w,double h):width(w),height(h){}

virtual double area() const {return 0.5\*width\*height;}

protected:

double width,height; };

int main()

{Circle circle(12.6);

Square square(3.5);

Rectangle rectangle(4.5,8.4);

Trapezoid trapezoid(2.0,4.5,3.2);

Triangle triangle(4.5,8.4);

Shape \*pt[5]={&circle,&square,&rectangle,&trapezoid,&triangle};

double areas=0.0;

for(int i=0;i<5;i++)

{areas=areas+pt[i]->area();}

cout<<"totol of all areas="<<areas<<endl; //输出总面积

return 0;}

**26、编程序实现以下功能：**

**(1)按职工号由小到大的顺序将5个员工的数据(包括号码,姓名,年龄,工资)输出到磁盘文件中保存**

**(2)从键盘输入两个员工的数据(职工号大于已有的职工号)，增加到文件的末尾。**

**(3)输入文件中全部职工的数据**

**(4)从键盘输入一个号码，从文件中查找有无此职工号，如有则显示此职工是第几个职工，以及此职工的全部数据。如没有，就输出“无此人”。可以反复多次查询，如果输入查找的职工号是0，就结束查询。**

#include <iostream>

#include <fstream>

using namespace std;

struct staff

{ int num;

char name[20];

int age;

double pay;};

int main()

{staff staf[7]={2101,"Li",34,1203,2104,"Wang",23,674.5,2108,"Fun",54,778,3006,"Xue",45,476.5,5101,"Ling",39,656.6},staf1;

fstream iofile("staff.dat",ios::in|ios::out|ios::binary);

if(!iofile)

{cerr<<"open error!"<<endl;

abort(); }

int i,m,num;

cout<<"Five staff :"<<endl;

for(i=0;i<5;i++)

{cout<<staf[i].num<<" "<<staf[i].name<<" "<<staf[i].age<<" "<<staf[i].pay<<endl;

iofile.write((char \*)&staf[i],sizeof(staf[i]));}

cout<<"please input data you want insert:"<<endl;

for(i=0;i<2;i++)

{cin>>staf1.num>>staf1.name>>staf1.age>>staf1.pay;

iofile.seekp(0,ios::end);

iofile.write((char \*)&staf1,sizeof(staf1));}

iofile.seekg(0,ios::beg);

for(i=0;i<7;i++)

{iofile.read((char \*)&staf[i],sizeof(staf[i]));

cout<<staf[i].num<<" "<<staf[i].name<<" "<<staf[i].age<<" "<<staf[i].pay<<endl; }

bool find;

cout<<"enter number you want search,enter 0 to stop.";

cin>>num;

while(num)

{find=false;

iofile.seekg(0,ios::beg);

for(i=0;i<7;i++)

{iofile.read((char \*)&staf[i],sizeof(staf[i]));

if(num==staf[i].num)

{m=iofile.tellg();

cout<<num<<" is No."<<m/sizeof(staf1)<<endl;

cout<<staf[i].num<<" "<<staf[i].name<<" "<<staf[i].age<<" "<<staf[i].pay<<endl;

find=true;

break;

} }

if(!find)

cout<<"can't find "<<num<<endl;

cout<<"enter number you want search,enter 0 to stop.";

cin>>num; }

iofile.close();

return 0;

}

27、

**（1）定义一个商品类CGoods，其中包含商品号(long no)、商品名(char \*p\_name)、商品价格(double price)**

**三个数据成员，以及相应的构造函数、拷贝构造函数、析构函数、打印数据成员的成员函数。**

**（2）为CGoods类增加一个商品总数(int count)数据成员，并增加一个成员函数getCount()获取count的值，编写一个友元函数getName()获取商品名称p\_name；**

**（3）为CGoods类定义小于运算符(‘<’)和不小于运算符(‘>=’)两个运算符重载函数。**

**CGoods类对象大小的比较是根据其商品价格(price)的值的大小来实现的。**

**（4）以CGoods类为基类，派生出服装类CClothes和食品类CFood两个派生类，并在这两个类中分别增加一个表示品牌的指针数据成员(char \*p\_brand)和表示用途的成员函数(void usedFor()——可分别输出一条表示服装和食品用途的信息)。写出CClothes类和CFood类的完整定义(包括构造、析构和usedFor()成员函数的实现)。**

#include <iostream>

#include <string.h>

using namespace std;

class CGood

{

private:

long no;

char \*p\_name;

double price;

int count;//CGoods类增加一个商品总数(int count)数据成员

public:

CGood (long no\_val,char \*p\_val,double price\_val,int count\_val);

CGood (const CGood &r\_course);

~ CGood (){delete p\_name;};

void print() const;

int getCount() const;//增加一个成员函数getCount()获取count的值

friend char \*getName(CGood &cgoods);//编写一个友元函数getName()获取商品名称p\_name

bool operator <(const CGood &cg)const//为CGoods类定义小于运算符(‘<’)和不小于运算符(‘>=’)两个运算符重载函数

{

if(price<cg.price)return true;

return false;

}

bool operator >=(const CGood &cg)const

{

if(price>=cg.price)return true;

return false;

}

};

CGood:: CGood (long no\_val,char \*p\_val,double price\_val,int count\_val)

{

no=no\_val;

p\_name=new char[strlen(p\_val)+1];

strcpy(p\_name,p\_val);

price=price\_val;

count=count\_val;

}

CGood:: CGood (const CGood &r\_course)

{

no=r\_course.no;

p\_name=new char[strlen(r\_course.p\_name)+1];

strcpy(p\_name,r\_course.p\_name);

price=r\_course.price;

count=r\_course.count;

}

void CGood::print() const

{

cout<<" CGood number:"<<no<<endl;

cout<<" CGood name:"<<p\_name<<endl;

cout<<" CGood price:"<<price<<endl;

cout<<" CGood count:"<<count<<endl;

}

int CGood::getCount() const

{

return count;

}

char \*getName(CGood &cgoods)

{

return cgoods.p\_name;

}

//以CGoods类为基类，派生出服装类CClothes和食品类CFood两个派生类

class CClothes : public CGood

{

private:

char \*p\_brand;

public:

CClothes(long no\_val,char \*p\_val,double price\_val,int count\_val,char \*brand):CGood (no\_val,p\_val,price\_val,count\_val)

{

p\_brand=new char[strlen(brand)+1];

strcpy(p\_brand,brand);

}

~CClothes(){delete p\_brand;}

void usedFor()

{

cout<<"This is used for clothes"<<endl;

}

};

class CFood : public CGood

{

private:

char \*p\_brand;

public:

CFood(long no\_val,char \*p\_val,double price\_val,int count\_val,char \*brand):CGood (no\_val,p\_val,price\_val,count\_val)

{

p\_brand=new char[strlen(brand)+1];

strcpy(p\_brand,brand);

}

~CFood(){delete p\_brand;}

void usedFor()

{

cout<<"This is used for food"<<endl;

}

};

int main()

{

//1

char name[10]="no.1";

CGood a(1,name,100,50);//NO:1 Name:yes price:100 count:50

a.print();

//2

int acount=a.getCount();

char \*aname=getName(a);

cout<<acount<<' '<<aname<<endl;

//3

CGood b(2,"no.2",70,50);//直接用“……”(string类型)赋值会有warning，但不影响运行

if(b<a) cout<<"b<a"<<endl;//很明显70<100

else cout<<"b>a"<<endl;

//4

CClothes cl(3,"clothes",10,10,"Clothes");

CFood fd(4,"food",10,10,"FOODS");

cl.usedFor();

fd.usedFor();

}

**28、编写一程序，统计一篇英文文章中单词的个数与行数。**

#include<iostream>

#include<fstream>

using namespace std;

bool isalph(char);

int main(){

ifstream ifile("E:\\lenovo\\english\\daily.doc");

char text[1000];

bool inword=false;

int rows=0,words=0;

int i;

while(!ifile.eof()){

ifile.getline(text,999);

rows++;

i=0;

while(text[i]!=0){

if(!isalph(text[i]))

inword=false;

else if(isalph(text[i]) && inword==false){

words++;

inword=true;

}

i++;

}

}

cout<<"rows= "<<rows<<endl;

cout<<"words= "<<words<<endl;

ifile.close ();

return 0;

}

bool isalph(char c){

return ((c>='A' && c<='Z') || (c>='a' && c<='z'));

}

**29、编写一程序，将C++源程序每行前加上行号与一个空格。**

#include<iostream>

#include<fstream>

using namespace std;

int main(){

int i=1;

char c[1000];

ifstream ifile("D:\\10\_4\_3.cpp");

ofstream ofile("D:\\r10\_4\_3.cpp");

while(!ifile.eof()){

ofile<<i++<<": ";

ifile.getline(c,999);

ofile<<c<<endl;

}

ifile.close();

ofile.close();

return 0;

}