实验一 C++的初步认识

题目一：输出个人信息

【问题描述】

从键盘输入自己的班级、学号、姓名，将信息输出到屏幕上。班级、学号、姓名都按字符串做处理。

【实验输出】

实验输出效果如图1-1所示。

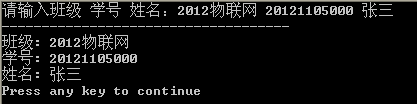


图1-1

【实验思路】

本实验主要考查C++标准输入/输出流对象cin、cout的基本用法，这两个对象声明于C++标准头文件<iostream>中。

题目要求从键盘输入三个字符串分别代表班级、学号和姓名。C++通过标准头文件<string>定义了字符串类型string，因此C++中字符串变量（更准确地说，是字符串对象）的定义语法为：

**#include <string>**

using namespace std;

int main()

{

string cla,id,name;

…… ……

return 0;

}

#include<iostream>

#include<string>

using namespace std;

int main()

{

string cla, num, name;

cout<<"请输入班级 学号 姓名：";

cin>>cla>>num>>name;

cout<<"----------------------------------"<<endl;

cout<<"班级："<<cla<<endl;

cout<<"学号："<<num<<endl;

cout<<"姓名："<<name<<endl;

return 0;

}

题目二：输入/输出基本类型数据

【问题描述】

从键盘输入5个整数，按照从大到小的顺序进行排序，并求其平均值。

【实验输出】

实验输出效果如图1-2所示。



图1-2

【实验思路】

本实验主要考查C++中整型、浮点型、数组等数据的输入/输出。首先，定义有5个元素的整型数组，利用cin循环键入元素值，并在循环中累加元素值。接下来采用任意一种排序方法对数组进行排序，并输出排序结果和所求平均值。

#include<iostream>

using namespace std;

int main()

{

int arr[5], i, j, t;

double ave = 0;

cout<<"排序前："<<endl;

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

{

cout<<"arr["<<i+1<<"] = ";

cin>>arr[i];

ave += arr[i];

}

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

for(j = i+1; j < 5; j++)

if(arr[i] < arr[j]){

t = arr[i];

arr[i] = arr[j];

arr[j] = t;

}

cout<<"排序后："<<endl;

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

cout<<"arr["<<i+1<<"] = "<<arr[i]<<endl;

cout<<"数组平均值为："<<ave/5<<endl;

return 0;

}

题目三：输出素数

【问题描述】

输出100以内的所有素数。

【实验输出】

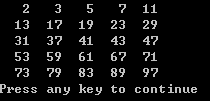
实验输出效果如图1-3所示。

图1-3

【实验思路】

找出1——100的所有的素数。素数为除了1和本身之外不被任何数整除的整数。每五个为一行的输出素数。

#include<iostream>

#include<cmath>

#include<iomanip>

using namespace std;

int main()

{

int i, j, k, count = 0;

for(i = 2; i < 101; i++)

{

k = sqrt(i);

for(j = 2; j <= k; j++)

if(i % j == 0)

break;

if(j > k)

{

cout<<setw(4)<<i;

count++;

if(count % 5 == 0)

cout<<endl;

}

}

return 0;

}

题目四：字符串排序

【问题描述】

输入n（2 <= n <= 20）个字符串，按英文字典排序，由小到大顺序输出。

【实验输出】

实验输出效果如图1-4所示。



图1-4

【实验思路】

定义一个整数n，长度为20的字符串。首先输入一个整数n，再输入n个字符串。使用任意一种排序方法对n个字符串排序。

#include<iostream>

#include<string>

using namespace std;

int main()

{

string s[30], t;

int n, i, j;

cin>>n;

cout<<"请输入 "<<n<<" 个字符串"<<endl;

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

cin>>s[i];

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

for(j = i+1; j < n ; j++)

if(s[i]>s[j])

{

t=s[i];

s[i]=s[j];

s[j]=t;

}

cout<<"排序后的字符串为："<<endl;

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

cout<<s[i]<<endl;

return 0;

}

题目五：求阶乘之和

【问题描述】

计算n（2 <= n <= 20）的阶乘。

【实验输出】

实验输出效果如图1-5所示。



图1-5

【实验思路】

输入一个整数n，求该数的阶乘。

#include<iostream>

using namespace std;

int main()

{

int n, i = 1, sum = 1;

cin>>n;

do

{

sum = sum \* i;

i++;

}while(i<=n);

cout<<n<<"! = "<<sum<<endl;

return 0;

}

题目六：指针和引用

【问题描述】

定义一个函数，比较两个数的大小，形参分别使用指针和引用。

【实验输出】

实验输出效果如图1-6所示。



图1-6

【实验思路】

定义一个名为max1，返回值为整型，形参使用指针。该子函数定义为：

int max1(int \*a, int \*b);

定义一个名为max2，返回值为引用，形参使用引用。该子函数定义为：

int & max2(int &a, int &b);

#include<iostream>

using namespace std;

int main()

{

int max1(int \*a, int \*b);

int & max2(int &a, int &b);

int a, b;

cout<<"请输入俩个数：";

cin>>a>>b;

cout<<"行参使用指针，比较后较大的数是："<<max1(&a,&b)<<endl;

cout<<"行参使用引用，比较后较大的数是："<<max2(a,b)<<endl;

return 0;

}

int max1(int \*a, int \*b)

{

return \*a>\*b?\*a:\*b;

}

int & max2(int &a, int &b)

{

return a>b?a:b;

}

题目七：重载函数、函数模版、带默认参数的函数

【问题描述】

求不同类型的3个数的最大值，要求：

（1）使用重载函数和函数模版两种方法。

（2）使用带默认参数的函数。

【实验输出】

使用重载函数和函数模版输出结果如图1-7所示。

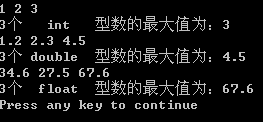


图1-7

使用带缺省参数的函数输出结果如图1-8所示。

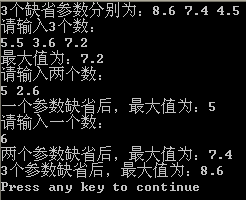


图1-8

【实验思路】

使用重载函数时子函数名相同，根据形式参数和返回值的类型来确定使用哪一个子函数。求int型数据最大值的子函数声明为：

int max(int a,int b,int c);

求double型数据最大值的子函数声明为：

double max(double a,double b,double c);

求float型数据最大值的子函数声明为：

float max(float a,float b,float c);

使用模版时需使用通用函数定义。通用函数定义和模版声明语句为：

template <class T>

T max(T x, T y, T z) ;

使用带缺省参数的函数时须在函数定义时对形式参数赋初值（该例子设的数据类型为double型）。函数定义语句为：

double max(double x = 8.6, double y = 7.4, double z = 4.5) ;

重载函数

#include<iostream>

using namespace std;

int main()

{

int max(int a,int b,int c);

double max(double a,double b,double c);

float max(float a,float b,float c);

int a1, b1, c1;

double a2, b2, c2;

float a3, b3, c3;

cin>>a1>>b1>>c1;

cout<<"3个 int 型数的最大值为："<<max(a1,b1,c1)<<endl;

cin>>a2>>b2>>c2;

cout<<"3个 double 型数的最大值为："<<max(a2,b2,c2)<<endl;

cin>>a3>>b3>>c3;

cout<<"3个 float 型数的最大值为："<<max(a3,b3,c3)<<endl;

return 0;

}

int max(int a,int b,int c)

{

int t = b;

if(a > b)

t = a;

if(t < c)

t = c;

return t;

}

double max(double a,double b,double c)

{

double t = b;

if(a > b)

t = a;

if(t < c)

t = c;

return t;

}

float max(float a,float b,float c)

{

float t = b;

if(a > b)

t = a;

if(t < c)

t = c;

return t;

}

函数模板

#include<iostream>

using namespace std;

template<typename T>

T max(T a,T b,T c)

{

T t = b;

if(a > b)

t = a;

if(t < c)

t = c;

return t;

}

int main()

{

int a1, b1, c1;

double a2, b2, c2;

float a3, b3, c3;

cin>>a1>>b1>>c1;

cout<<"3个 int 型数的最大值为："<<max(a1,b1,c1)<<endl;

cin>>a2>>b2>>c2;

cout<<"3个 double 型数的最大值为："<<max(a2,b2,c2)<<endl;

cin>>a3>>b3>>c3;

cout<<"3个 float 型数的最大值为："<<max(a3,b3,c3)<<endl;

return 0;

}

默认参数

#include<iostream>

using namespace std;

double max(double x = 8.6, double y = 7.4, double z = 4.5)

{

double t = y;

if(x > y)

t = x;

if(t < z)

t = z;

return t;

}

int main()

{

double a2, b2, c2;

cout<<"3个缺省参数分别为：8.6 7.4 4.5"<<endl;

cout<<"请输入3个数："<<endl;

cin>>a2>>b2>>c2;

cout<<"最大值为："<<max(a2,b2,c2)<<endl;

cout<<"请输入俩个数："<<endl;

cin>>a2>>b2;

cout<<"一个参数缺省后，最大值为："<<max(a2,b2)<<endl;

cout<<"请输入一个数："<<endl;

cin>>a2;

cout<<"俩个参数缺省后，最大值为："<<max(a2)<<endl;

cout<<"3个参数缺省后，最大值为："<<max()<<endl;

return 0;

}

实验二

题目一：求屏幕上两点间的距离

【问题描述】

输入两个点的横纵坐标，计算两点间距离。

【实验输出】

实验输出效果如图2-1所示。

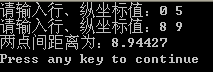


图2-1

【实验思路】

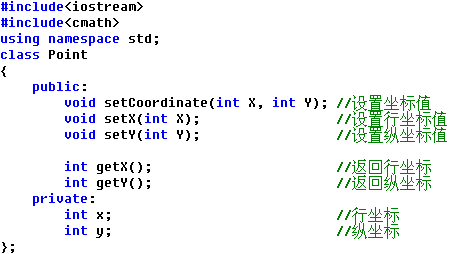
面向对象程序设计的一般方法是：

（1）确定问题中涉及的类；（2）找出每个类的静态特征（数据成员）和动态特征（成员函数），并将其实现。

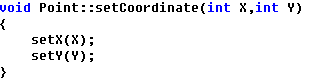
本题涉及的核心类是点Point类。该类的数据成员显然是行坐标和纵坐标。围绕这两个数据成员进行操作的成员函数应该有：设置行（纵）坐标，读取行（纵）坐标。

代码模版：

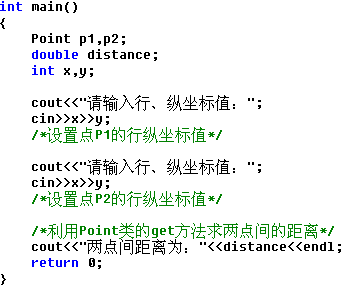
（1）Point类声明



（2）成员函数setCoordinate(int X,int Y)的实现



（3）main()的实现



#include<iostream>

#include<cmath>

using namespace std;

class Point

{

private:

int x;

int y;

public:

void setcoordinate(int X, int Y);

void setX(int X);

void setY(int Y);

int getx();

int gety();

};

void Point::setcoordinate(int X, int Y)

{

setX(X);

setY(Y);

}

void Point::setX(int X)

{

x = X;

}

void Point::setY(int Y)

{

y = Y;

}

int Point::getx()

{

return x;

}

int Point::gety()

{

return y;

}

int main()

{

Point p1, p2;

double distance;

int x, y;

cout<<"请输入行，纵坐标：";

cin>>x>>y;

p1.setcoordinate(x, y);

cout<<"请输入行，纵坐标：";

cin>>x>>y;

p2.setcoordinate(x, y);

distance = sqrt(((p1.getx()-p2.getx())\*(p1.getx()-p2.getx())) + ((p1.gety()-p2.gety())\*(p1.gety()-p2.gety())));

cout<<"两点间距离为："<<distance<<endl;

return 0;

}

题目二：计算房贷

【问题描述】

已知银行住房贷有两种还款方式：等额本息法和等额本金法，其中后一种还款方式的计算公式为：

每月本金 = 贷款总额÷还款次数（月数）

每月利息 = 上月剩余本金×月利率（即：年利率/12）

每月还款额 = 每月本金＋每月利息

设计一个房贷计算器，再输入贷款总额、贷款年利率以及借贷月数后，能根据上述公式计算出每月还款额。

【实验输出】

实验输出效果如图2-2所示。

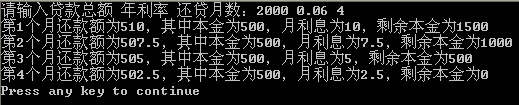
****

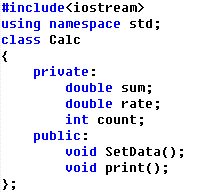
图2-2

【实验思路】

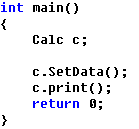
问题的关键是设计出房贷计算器类Calc。已知类设计的关键是确定该类应该具备哪些数据成员，围绕数据成员应实现哪些功能（成员函数）。因此应先找出Calc类的数据成员。根据题意可知，贷款总额、年利率、贷款月数是Calc类的数据成员，这些数据可通过方法SetData()输入、方法print()用于计算并打印每月还款额。

代码模块：

（1）Calc类声明



（2）mian()的实现



#include<iostream>

using namespace std;

class Calc

{

private:

double sum;

double rate;

int count;

public:

void Setdata();

void print();

};

void Calc::Setdata()

{

cout<<"请输入贷款总额 年利率 还款月数：";

cin>>sum>>rate>>count;

}

void Calc::print()

{

int i, bj=sum/count, sy=sum;

double hke=0, ylx=0;

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

{

ylx = sum \* 0.06 / 12;

sy = sum - bj;

hke = bj + ylx;

cout<<"第"<<i+1<<"个月还款额为"<<hke;

cout<<"，"<<"其中本金为"<<bj<<"，";

cout<<"月利息为"<<ylx<<"，";

cout<<"剩余本金为"<<sy<<endl;

sum = sy;

}

}

int main()

{

Calc c;

c.Setdata();

c.print();

return 0;

}

题目三：积分返券

【问题描述】

某商场年终举行“积分返券”活动，其规则为：每张会员卡积满1000分返礼券10元，积满3000分礼券30元，积满5000分返礼券100元。返礼券同时，消除卡上礼券对应的积分。编程帮助商场实现返券活动。

【实验输出】

实验输出效果如图2-3所示。

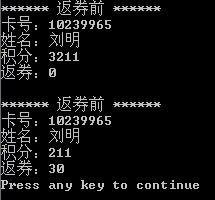
****

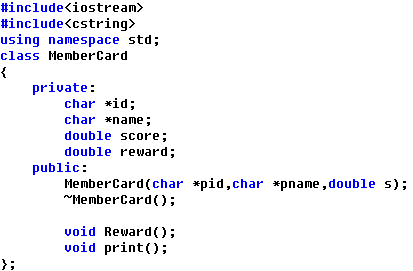
图2-3

【实验思路】

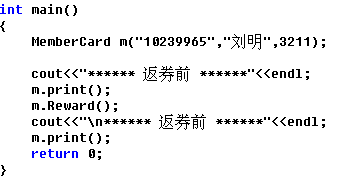
本实验涉及的类主要是会员卡类MemberCard。它具有卡号、姓名、积分额、礼券金额4个数据成员，主要成员函数包括构造函数和积分返现函数。

代码模版：

（1）MemberCard类声明



（2）main()实现



#include<iostream>

using namespace std;

class Membercard

{

private:

char \*id;

char \*name;

double score;

double reward;

public:

Membercard(char \*pid, char \*pname, double s);

~Membercard();

void Reward();

void print();

};

Membercard::Membercard(char \*pid, char \*pname, double s)

{

id=pid;

name=pname;

score=s;

reward = 0;

}

Membercard::~Membercard()

{

cout<<"该空间已释放！"<<endl;

}

void Membercard::Reward()

{

if(score>=1000 &&score<3000)

{

reward=10;

score-=1000;

}

else if(score>=3000 &&score<5000)

{

reward=30;

score-=3000;

}

else if(score>=5000)

{

reward=100;

score-=5000;

}

}

void Membercard::print()

{

cout<<"卡号"<<id<<endl;

cout<<"姓名"<<name<<endl;

cout<<"积分"<<score<<endl;

cout<<"返券"<<reward<<endl;

}

int main()

{

Membercard m("10239965","刘明",3211);

cout<<"\*\*\*\*\*\* 返券前 \*\*\*\*\*\*"<<endl;

m.print();

m.Reward();

cout<<"\n\*\*\*\*\*\* 返券后 \*\*\*\*\*\*"<<endl;

m.print();

return 0;

}

题目四：设计实时更新的时间类Time

【问题描述】

设计能按照系统时间更新的Time类。它有数据成员时、分、秒，其主要成员函数有：带参数构造函数（根据用户输入的值进行初始化）、无参构造函数（根据当前的系统时间进行初始化）、使时间增加1秒的方法tick（）、以格林尼治时间格式（24小时制）输出时间的方法printUniversal()，以标准时间格式（12小时制）输出时间的方法printStandard()。

【实验输出】

实验输出效果如图2-4所示。

****

图2-4

【实验思路】

题目已清楚说明Time类应具备的主要成员函数和数据成员。

Time类的设计难点在于以下几方面。

（1）如何获取计算机上当前的系统时间并完成无参初始化。

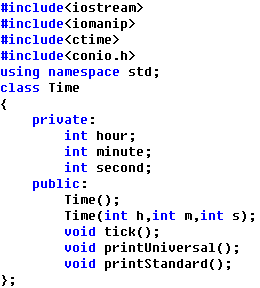
（2）成员函数tick()应确保时、分、秒的正确进位。如时间23:59:59，在调用tick()后，应为00:00:00。

（3）在主函数中构造循环，每隔1秒应调用一次成员函数tick()和printUniversal()，保证时间的实时更新。在循环中如何统计当前时差为1秒？

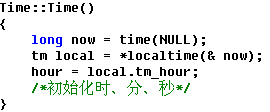
（4）时间输出函数应格式化地输出时间内，如时间是9点0分0秒时，应输出“09:00:00”。

代码模版：

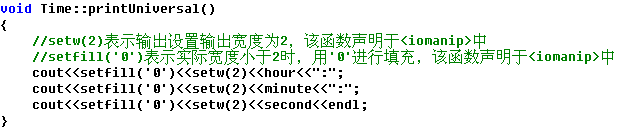
（1）Time类声明



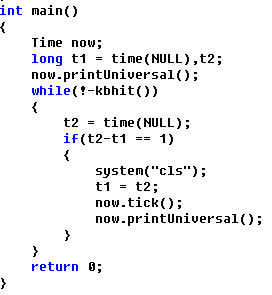
（2）缺省构造函数



（3）打印格林尼治时间



（4）main()的实现



#include<iostream>

#include<iomanip>

#include<ctime>

#include<conio.h>

using namespace std;

class Time

{

public:

Time();

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

void tick();

void printUniversal();

void printStandard();

private:

int hour;

int minute;

int second;

};

Time::Time()

{

long now = time(NULL);

tm local = \*localtime(& now);

hour = local.tm\_hour;

minute = local.tm\_min;

second = local.tm\_sec;

}

void Time::tick()

{

second++;

minute = minute + second / 60;

second = second % 60;

hour = hour + minute / 60;

minute = minute % 60;

hour = hour % 24;

}

void Time::printUniversal()

{

cout<<setfill('0')<<setw(2)<<hour<<":";

cout<<setfill('0')<<setw(2)<<minute<<":";

cout<<setfill('0')<<setw(2)<<second<<endl;

}

void Time::printStandard()

{

if(hour > 12)

{

cout<<setfill('0')<<setw(2)<<hour-12<<":";

cout<<setfill('0')<<setw(2)<<minute<<":";

cout<<setfill('0')<<setw(2)<<second<<endl;

}

}

int main()

{

Time now;

long t1 = time(NULL), t2;

now.printUniversal();

now.printStandard();

while(!-kbhit())

{

t2 = time(NULL);

if(t2 - t1 == 1)

{

system("cls");

t1 = t2;

now.tick();

now.printUniversal();

now.printStandard();

}

}

return 0;

}

题目五：管理个人活期账号

【问题描述】

个人储蓄活期帐户包括帐号、户名、密码、余额、活期年利率等信息。要求能够对个人账户进行存钱、取钱、计算年利息、打印账户相关信息等操作。编写主函数测试账户相关功能。

【实验输出】

实验输出效果如图2-5所示。

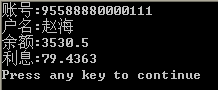
****

图2-5

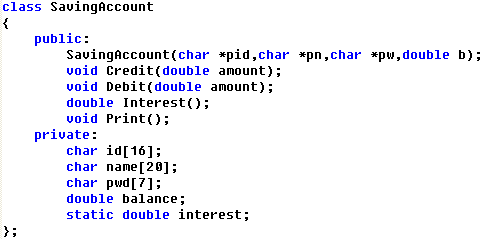
【实验思路】

设计SavingAccount类时，应注意观察哪些数据成员和成员函数是对象所具有的特征，而哪些是储蓄账户这一类别具有的特征。如果是这一类别的特征，那么应该把该成员视为类的静态成员。

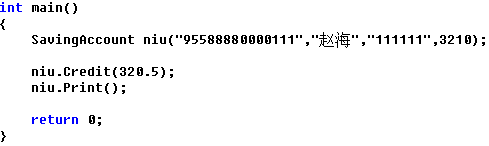
账号、户名、密码、余额等数据成员对每个对象来说，其值是不同的，因此它们是对象的特征。而年利率是账户类所有对象共享的特征，应被定义为静态数据成员。该类的成员函数有构造函数、存钱、取钱、计算年利率。根据上述信息可完成SavingAccount类的定义。

代码模版：

（1）SavingAccount类声明



（2）main()的实现



#include<iostream>

#include<cstring>

using namespace std;

class Savingaccount

{

public:

Savingaccount(char \*pid, char \*pn, char \*pw, double b);

void Credit(double amount);

void Debit(double amount);

double Interest();

void Print();

private:

char id[16];

char name[20];

char pwd[7];

double balance;

static double interest;

};

double Savingaccount::interest = 0.0225;

Savingaccount::Savingaccount(char \*pid, char \*pn, char \*pw, double b)

{

strcpy(id,pid);

strcpy(name,pn);

strcpy(pwd,pw);

balance = b;

}

void Savingaccount::Credit(double amount)

{

balance = balance + amount;

}

void Savingaccount::Debit(double amount)

{

balance = balance - amount;

}

double Savingaccount::Interest()

{

double lx;

lx = balance\*interest;

return lx;

}

void Savingaccount::Print()

{

cout<<"账户："<<id<<endl;

cout<<"户名："<<name<<endl;

cout<<"余额："<<balance<<endl;

cout<<"利息："<<Interest()<<endl;

}

int main()

{

Savingaccount niu("95588880000111","赵海","111111",3210);

niu.Credit(320.5);

niu.Debit(0);

niu.Print();

return 0;

}

题目六：打印信用卡欠费通知单

【问题描述】

已知某银行对每张信用卡收取100元/年的年费，每笔借款享有50天的无利息借贷期。若超过50天还款期限，按照公式

利息 = 日息（万分之五）×借款额×超期天数

来收取罚息。编写程序对欠费的用户打印欠费通知单，要求清单上详细列出卡号、姓名、借款额、欠款天数、罚息和欠款总额等信息。

【实验输出】

实验输出效果如图2-6所示。

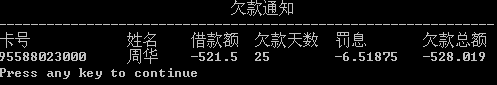
****

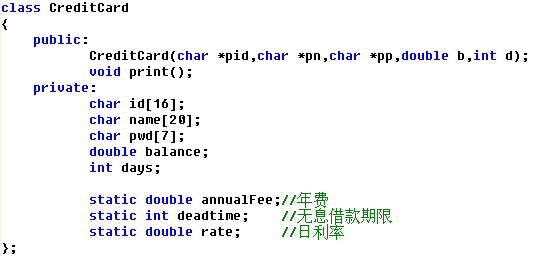
图2-6

【实验思路】

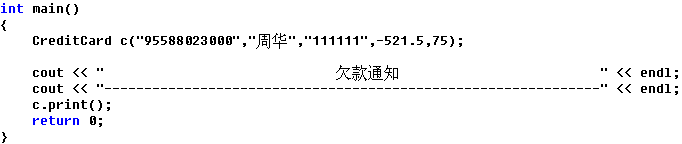
本题的关键是分清哪些属性是属于对象的特征，哪些属性是信用卡类共有的特征（即静态成员）。不难看出，卡号、姓名、密码、余额、借款额、欠款天数等应该是普通数据成员，而年费、无息贷款天数、罚息是信用卡类每个对象共享的数据，是静态数据成员。信用卡类的主要成员函数包括构造函数和打印清单的函数print()。

代码模版：

（1）CreditCard类声明



（2）mian()的实现



#include<iostream>

#include<cstring>

using namespace std;

class CreditCard

{

public:

CreditCard(char \*pid, char \*pn, char \*pp, double b, int d);

void print();

private:

char id[16];

char name[20];

char pwd[7];

double balance;

int days;

static double annualFee; //年费

static int deadtime; //无息借款期限

static double rate; //日利率

};

double CreditCard::annualFee = 100;

int CreditCard::deadtime = 50;

double CreditCard::rate = 0.0005;

CreditCard::CreditCard(char \*pid, char \*pn, char \*pp, double b, int d)

{

strcpy(id,pid);

strcpy(name,pn);

strcpy(pwd,pp);

balance = b;

days = d;

}

void CreditCard::print()

{

cout<<"卡号\t\t姓名\t借款额\t欠款天数\t罚息\t\t欠款总额"<<endl;

cout<<id<<"\t"<<name<<"\t"<<balance<<"\t"<<days-50<<"\t\t"<<balance\*rate\*(days-50)<<"\t"<<balance+balance\*rate\*(days-50)<<endl;

}

int main()

{

cout<<"\t\t\t\t\t欠款通知"<<endl;

cout<<"----------------------------------------------------------------------------"<<endl;

CreditCard c("95588023000","周华","111111",-521.5,75);

c.print();

return 0;

}

题目七：模拟存钱罐

【问题描述】

可以将币值为1角、5角、1元、5元、10元的零钱存入存钱罐中，最多能存放100张。

编程模拟存钱罐功能：用户按一次回车键代表存入一个零钱，币值随机。当用户按除回车键外的任意键后，程序应打印出存钱总额，并分类统计出各类币值的零钱分别是多少。

【实验输出】

实验输出效果如图2-7所示。

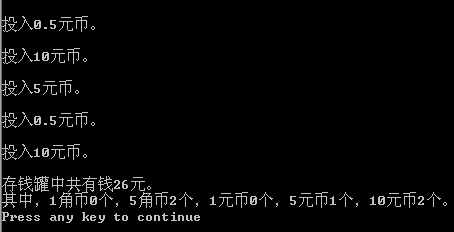
****

图2-7

【实验思路】

本实验难点是整个程序的框架布局。首先应确定问题涉及两个类：SavingBox类和Money类。而确定这两个类具有哪些成员则是难点之最。

（1）SavingBox类

由于存钱罐最多能存放100张零钱，因此其数据成员应该包含一个具有100个元素的指针数组“Money \* items[100]”，数组中的每个指针均可指

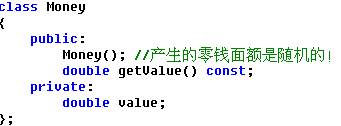
向一个Money类对象，利用该数组就实现了存钱罐的存钱功能。另外，为统计究竟存入多少币，还应定义整型数据成员count，每存入一个零币就计数一次。围绕这两个数据成员，应具有的成员函数有构造函数、析构函数、存钱函数PutIntoMoney()和统计函数Count()。

（2）Money类

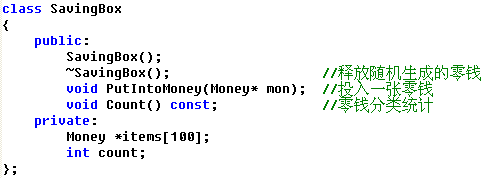
Money类仅有代表币值的浮点型数据成员value，其成员函数有构造函数和读取币值的函数getValue()。其中，构造函数应保证生成的币值是随机数，且该随机数在符合题目要求的范围内。此外，题目要求用户按一次回车键投入一张零钱，故Money类对象可用动态存储分配实现。

代码模块：

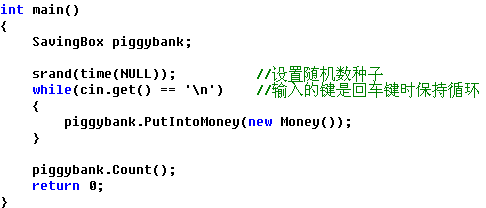
（1）Money类声明



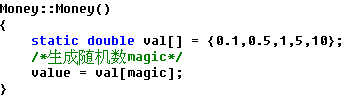
（2）SavingBox类声明

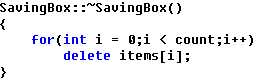


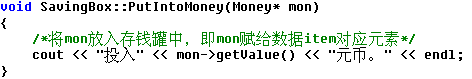
（3）main()的实现



（4）部分重要成员函数提示







#include<iostream>

#include<iomanip>

#include<ctime>

#include<conio.h>

using namespace std;

class Money

{

public:

Money();

double getValue() const;

private:

double value;

};

class SavingBox

{

public:

SavingBox();

~SavingBox();

void PutIntoMoney(Money \*mon);

void Count() const;

private:

Money \*items[100];

int count;

};

Money::Money()

{

static double val[] = {0.1, 0.5, 1, 5, 10};

int magic = rand()%5;

value = val[magic];

}

double Money::getValue() const

{

return value;

}

SavingBox::SavingBox()

{

count = 0;

}

SavingBox::~SavingBox()

{

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

delete items[i];

}

void SavingBox::PutIntoMoney(Money \*mon)

{

items[count] = mon;

count++;

cout<<"投入"<<mon->getValue()<<"元币。"<<endl;

}

void SavingBox::Count() const

{

int i, a=0, b=0, c=0, d=0, e=0;

double t = 0;

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

{

t = t + items[i]->getValue();

if(items[i]->getValue() == 0.1)

a++;

if(items[i]->getValue() == 0.5)

b++;

if(items[i]->getValue() == 1)

c++;

if(items[i]->getValue() == 5)

d++;

if(items[i]->getValue() == 10)

e++;

}

cout<<"存钱罐中共有钱"<<t<<"元。"<<endl;

cout<<"其中，1角币"<<a<<"个，5角币"<<b<<"个，1元币"<<c<<"个，5元币"<<d<<"个，10元币"<<e<<"个。"<<endl;

}

int main()

{

SavingBox piggybank;

srand(time(NULL));

while(cin.get() == '\n')

{

piggybank.PutIntoMoney(new Money());

}

piggybank.Count();

return 0;

}

题目八：设计实时更新的日期类Date

【问题描述】

设计日期类Date，它具有年、月、日等数据成员，并具有无参构造函数和带参构造函数，其中无参构造函数按照系统当前日期初始化年、月、日。此外，Date类还具有年、月、日对应的get和set的方法以及打印日期的print()方法。

打印出我国举办奥运会的日期和系统当前日期，为简单起见，假设每个月均有31天。

【实验输出】

实验输出效果如图2-8所示。

****

图2-8

【实验思路】

Date类具有哪些类成员题目已明确说明。实现该类的难点主要是如何获取系统当前时间进行初始化工作。此外，请读者观察本题涉及哪些常对象和常成员。

（1）获取系统当前日期

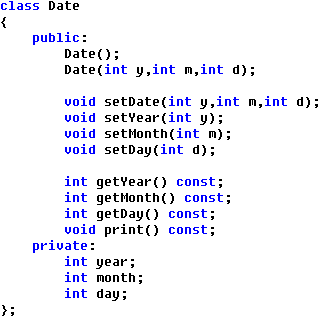
调用标准C头文件<ctime>中的库函数time(NULL)会返回自格林尼治时间1970年1月1日凌晨至现在所经过的秒数，可将其保存在变量now中，再调用函数localtime(&now)可以把time(NULL)返回的时间now转换为当地之间，并保存结构体strct tm类型的变量中，该变量的成员tm\_year、tm\_mon、tm\_mday即代表系统当前的年、月、日。

（2）找出问题中的常对象和常成员

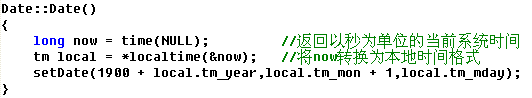
由于我国举办奥运会日期是已知的、确定的，故可以将其视为唱值，也就是常对象，其类型为const Date。而Date类中不会修改数据成员的成员函数是print()和get方法，故可将其声明为常成员函数。

代码模块：

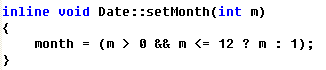
（1）Date类声明



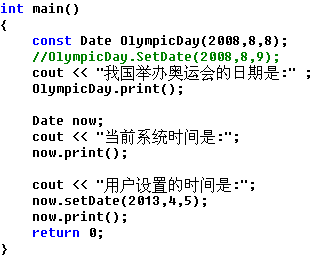
（2）Date类无参构造函数



（3）设置月份的成员函数



（4）main()的实现



#include<iostream>

#include<ctime>

using namespace std;

class Date

{

public:

Date();

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

void setDate(int y,int m,int d);

void setYear(int y);

void setMonth(int m);

void setDay(int d);

int getYear() const;

int getMonth() const;

int getDay() const;

void print() const;

private:

int year;

int month;

int day;

};

Date::Date()

{

long now = time(NULL);

tm local = \*localtime(& now);

setDate(1900 + local.tm\_year,local.tm\_mon+1,local.tm\_mday);

}

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

{

year = y;

month = m;

day = d;

}

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

{

year = y;

month = m;

day = d;

}

inline void Date::setYear(int y)

{

year++;

}

inline void Date::setMonth(int m)

{

month = (m > 0 && m <= 12 ? m : 1);

}

inline void Date ::setDay(int d)

{

day = (d > 0 && d <= 31 ? d : 1);

}

int Date::getYear() const

{

return(year);

}

int Date::getMonth() const

{

return(month);

}

int Date::getDay() const

{

return(day);

}

void Date::print() const

{

cout<<getYear()<<"-"<<getMonth()<<"-"<<getDay()<<endl;

}

int main()

{

const Date OlympicDay(2008,8,8);

cout<<"我国举办奥运会的日期是:";

OlympicDay.print();

Date now;

cout<<"当前系统时间是:";

now.print();

cout<<"用户设置的时间是：";

now.setDate(2013,4,5);

now.print();

return 0;

}

题目九：定义数据成员

【问题描述】

设计一个学生类student，它具有的私有数据成员是：学号，姓名以及数学、英语、计算机成绩；具有的公有成员函数是：求3门课总成绩的函数sum；求3门课平均成绩的函数average；显示学生数据信息的函数print；获取学生学号的函数get\_num；设置学生数据信息的函数set\_inf。

编制主函数，说明一个student类对象的数组并进行全班学生信息的输入与设置，而后求出每一学生的总成绩和平均成绩、全班学生的总成绩最高分、全班学生总平均分，并在输入一个学号后，输出与该学生有关的全部数据信息。

【实验输出】

实验输出效果如图2-9所示。

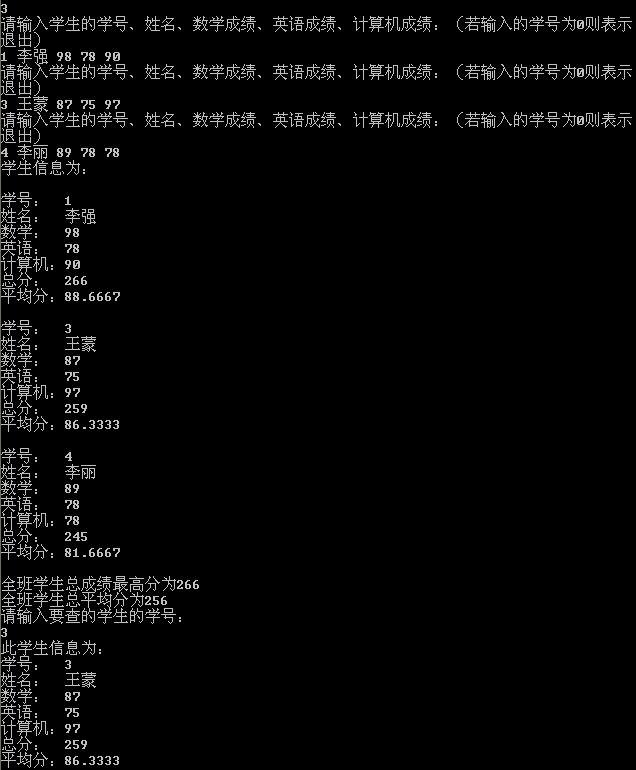
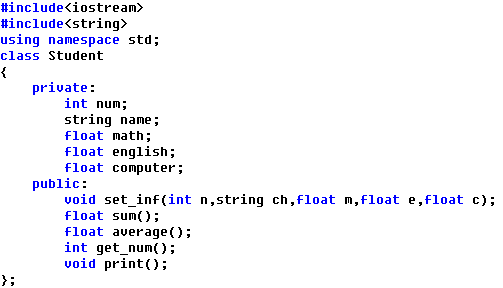
****

图2-9

【实验思路】

主函数中定义对象数组，可以定义长度为50。输入一个n确定有多少个学生信息的数据，根据输入的学生信息计算成绩总和和平均分。输入学号，输出对应的学生信息。

代码模版：



#include<iostream>

#include<string>

using namespace std;

class Student

{

private:

int num;

string name;

float math;

float english;

float computer;

public:

void set\_inf(int n, string ch, float m, float e, float c);

float sum();

float average();

int get\_num();

void print();

};

void Student::set\_inf(int n, string ch, float m, float e, float c)

{

num = n;

name = ch;

math = m;

english = e;

computer = c;

}

float Student::sum()

{

return (math+english+computer);

}

float Student::average()

{

return (math+english+computer)/3;

}

int Student::get\_num()

{

return num;

}

void Student::print()

{

cout<<"学号:"<<num<<endl;

cout<<"姓名:"<<name<<endl;

cout<<"数学:"<<math<<endl;

cout<<"英语:"<<english<<endl;

cout<<"计算机:"<<computer<<endl;

cout<<"总分:"<<sum()<<endl;

cout<<"平均分:"<<average()<<endl;

}

int main()

{

Student s[50];

int i,q,ms,es,xs,max=0,aver=0;

int n;

string mz;

cin>>n;

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

{

cout<<"请输入学生的学号、姓名、数学成绩、英语成绩、计算机成绩：（若输入的学号为0则表示退出）"<<endl;

cin>>q>>mz>>ms>>es>>xs;

if(q == 0)

break;

s[i].set\_inf(q,mz,ms,es,xs);

if(max > ms+es+xs);

else

max = ms+es+xs;

aver+=(ms+es+xs);

}

cout<<"学生信息为："<<endl<<endl;

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

{

s[i].print();

cout<<endl;

}

cout<<"全班学生总成绩最高分为"<<max<<endl;

cout<<"全班学生总平均分为"<<aver/3<<endl<<endl;

cout<<"请输入要查的学生的学号："<<endl;

cin>>q;

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

{

if(q == s[i].get\_num())

{

cout<<"此学生信息为："<<endl;

s[i].print();

break;

}

}

if(i == n)

cout<<"查无此人"<<endl;

return 0;

}

题目十：打印运动会参赛名单

【问题描述】

学校运动会有游泳、跳高、短跑等比赛项目，限每位参赛人员仅能参加一项比赛。请打印出参加游泳比赛的运动人员名单。已知运动员的属性有姓名、院系等，比赛项目的属性有项目名称、比赛时间、参赛人数、参赛人员等。

【实验输出】

实验输出效果如图2-10所示。

****

图2-10

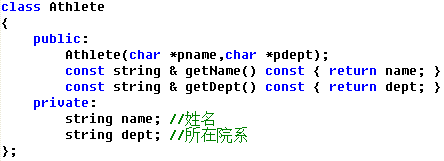
【实验思路】

由题意可知，问题涉及运动员类Athlete和比赛项目类Game。这两个类的关系在UML中称之为“关联（association）”，这里可理解为多个运动员参加游泳比赛。

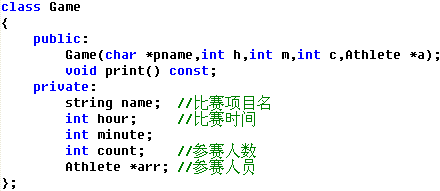
在设计这两个类时，如何体现它们之间的关联是本题的难点。通常情况下，可以采用对象指针来实现关联，即在游泳比赛Game类中定义一个Athlete \*类型的指针arr，arr指向参加游泳比赛的运动员集合（数组），该指针的初始化在Game类的构造函数中进行。

代码模块：

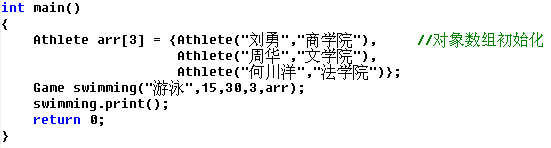
（1）Athlete类声明部分



（2）Game类声明部分



（3）main()的实现



#include<iostream>

#include<iomanip>

#include<string>

using namespace std;

class Athlete

{

public:

Athlete(char \*pname,char \*pdept);

const string & getName() const

{

return name;

}

const string & getDept() const

{

return dept;

}

private:

string name;

string dept;

};

class Game

{

public:

Game(char \*pname,int h,int m,int c,Athlete \*a);

void print() const;

private:

string name;

int hour;

int minute;

int count;

Athlete \*arr;

};

Athlete::Athlete(char \*pname,char \*pdept)

{

name = pname;

dept = pdept;

}

Game::Game(char \*pname,int h,int m,int c,Athlete \*a)

{

name = pname;

hour = h;

minute = m;

count = c;

arr = a;

}

void Game::print() const

{

int i;

cout<<"项目:"<<name<<" "<<"比赛时间:"<<hour<<"时"<<minute<<"分"<<endl;

cout<<"------------------------"<<endl;

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

{

cout<<i+1<<'\t'<<arr[i].getName()<<'\t'<<arr[i].getDept()<<endl;

}

}

int main()

{

Athlete arr[3] = {Athlete("刘勇","商学院"),Athlete("周华","文学院"),Athlete("何川洋","法学院")};

Game swimming("游泳",15,30,3,arr);

swimming.print();

return 0;

}

实验三

题目一：判断字符串是否为空

【问题描述】

定义一个字符串类String，其数据成员有指向字符串的指针elems，成员函数包括构造函数、判断字符串是否为空的operator!()。编程测试类String的功能。

【实验输出】

实验输出效果如图3-1和3-2所示。



图3-1



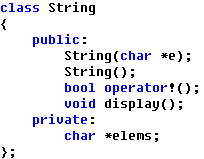
图3-2

【实验思路】

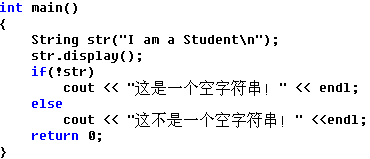
判断字符串是否为空即是对字符串进行逻辑非运算，即重载operator!()。逻辑非运算符“!”是单目运算符，按照运算符重载方针，应该重载为类的成员函数。由于逻辑非运算的结果只有两种：真、假，因此operator!()的函数返回值的类型是bool。

代码模版：

（1）String类声明



（2）main()的实现



#include<iostream>

using namespace std;

class String

{

private:

char \*elems;

public:

String();

String(char \*e);

bool operator ! ();

void display();

};

String::String()

{

elems = NULL;

}

String::String(char \*e)

{

elems = e;

}

bool String::operator ! ()

{

if(\*elems == NULL)

return true;

else

return false;

}

void String::display()

{

cout<<elems;

}

int main()

{

String str("I am a student\n");

str.display();

if(!str)

cout<<"这是一个空字符串！"<<endl;

else

cout<<"这不是一个空字符串！"<<endl;

return 0;

}

题目二：判断字符串是否为空

【问题描述】

定义日期类Date，它能显示日期、设置日期，能重载operator += (int days)实现将日期增加days天，重载前缀运算符operator++()、后缀运算符operator++(int)实现将日期增加1天。编写主函数测试Date类。

【实验输出】

实验输出效果如图3-3所示。

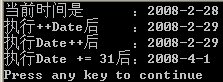


图3-3

【实验思路】

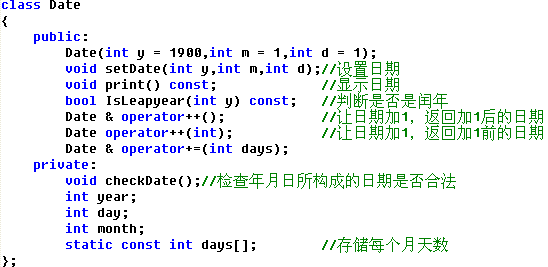
题目已经清楚说明Date类的主要成员，其中重载operator ++()、operator++(int)和operator+=(int days)是本题的难点。对Date对象执行自增运算时，会遇到一些边界条件，如“2008年2月29日”自增一次应该是“2008年3月1日”。为保证Date类能正确处理这些边界条件，应定义一个静态整型数组成员days[]，它存放不考虑闰年情况下，每个月的天数。再定义一个成员函数IsLeapyear(int y)判断参数y是否是闰年。当调用operator++()时，首先根据函数IsLeapyear(int y)和数组days检测自增一次后当前日期是否超出边界，若超出边界，如自增后变为“2008年2月30日”，应将其调整为正确日期“2008年3月1日”。

由于程序中要多次用到检测日期的功能，故再定义私有成员函数checkDate()将当前日期修正为合法日期，专供其他成员函数调用。

对于operator+=(int days)，只需要在函数中循环daysci，每次让数据成员day自增一次，并调用checkDate()修改自增后的日期即可。

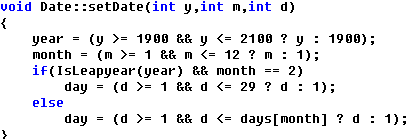
代码模版：

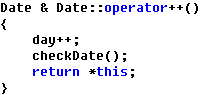
（1）Date类声明



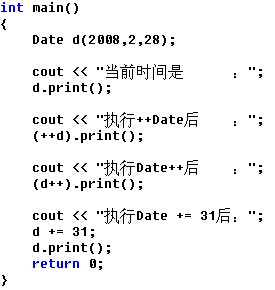
（2）Date类的部分实现







（3）main()的实现



#include<iostream>

using namespace std;

class Date

{

private:

void checkDate();

int year;

int month;

int day;

static const int days[];

public:

Date(int y=1900,int m=1,int d=1):year(y),month(m),day(d){}

void setDate(int y,int m,int d);

void print() const;

bool IsLeapyear(int y) const;

Date & operator++();

Date operator++(int);

Date & operator+=(int days);

};

const int Date::days[]={0,31,28,31,30,31,30,31,31,30,31,30,31};

void Date::checkDate()

{

if(year<1900 && year>2100)

{

year=1900;

if(month<1 && month>12)

{

month=1;

}

}

if(day>days[month])

{

if(IsLeapyear(year) && month == 2)

{

if(day>29)

{

day=1;

month++;

}

}

else

{

day=1;

++month;

}

}

}

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

{

year=(y>=1900 && y<=2100 ? y:1900);

month=(m>=1 && m<=12 ? m:1);

if(IsLeapyear(year) && month==2)

day=(d>=1 && d<=29 ? d:1);

else

day=(d>=1 && d<=days[month] ? d:1);

}

void Date::print() const

{

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

}

bool Date::IsLeapyear(int y) const

{

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

return true;

else

return false;

}

Date Date::operator ++ (int)

{

Date t(\*this);

++day;

checkDate();

return \*this;

}

Date & Date :: operator ++ ()

{

day++;

checkDate();

return \*this;

}

Date & Date::operator +=(int days)

{ int i;

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

{

day++;

}

checkDate();

return \*this;

}

int main()

{

Date d(2008,2,28);

cout<<"当前时间是 ：";

d.print();

cout<<"执行++Date后 ：";

(++d).print();

cout<<"执行Date++后 ：";

(d++).print();

cout<<"执行Date += 31后 ：";

d += 31;

d.print();

return 0;

}

题目三：矩阵运算

【问题描述】

对具有相同行列数的整型矩阵进行加、减、乘、转置、赋值运算。

【实验输出】

实验输出效果如图3-4所示。

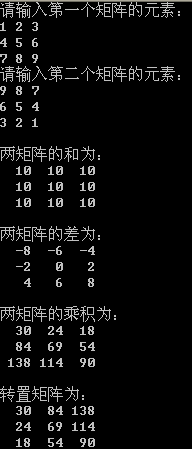


图3-4

【实验思路】

定义矩阵类Matrix，它的数据成员有：行line、列col、指向矩阵元素的指针int elems[100][100]。它的成员函数有：构造函数、复制函数和转置函数。还需要对Matrix类重载如下运算符：

（1）Matrix operator + (const Matrix &a, const Matrix &b);

（2）Matrix operator - (const Matrix &a, const Matrix &b);

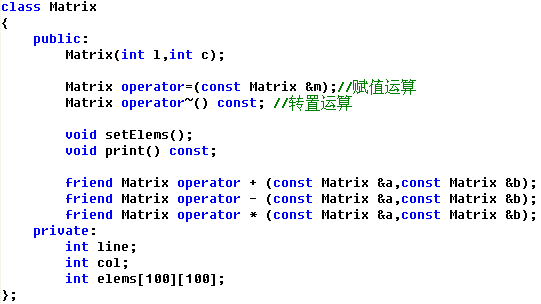
（3）Matrix operator \* (const Matrix &a, const Matrix &b);

（4）Matrix operator = (const Matrix &m);

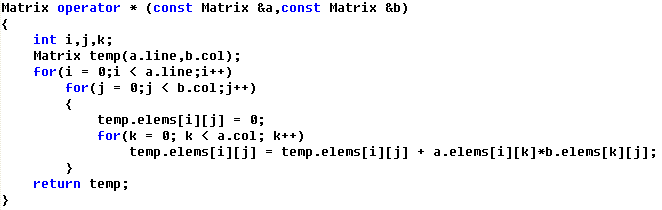
（5）Matrix operator ~() const;

代码模版：

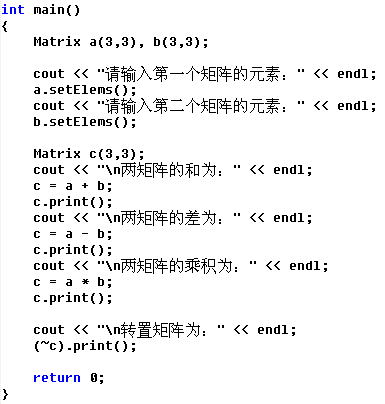
（1）Matrix类声明



（2）成员函数operator\*()的部分实现



（3）main()的实现



#include<iostream.h>

#include<iomanip.h>

//using namespace std;

class Matrix

{

public:

Matrix(int l,int c);

Matrix operator = (const Matrix &m);

Matrix operator ~ () const;

void setElems();

void print() const;

friend Matrix operator + (const Matrix &a,const Matrix &b);

friend Matrix operator - (const Matrix &a,const Matrix &b);

friend Matrix operator \* (const Matrix &a,const Matrix &b);

private:

int line;

int col;

int elems[100][100];

};

Matrix::Matrix(int l,int c)

{

line = l;

col = c;

}

Matrix Matrix::operator = (const Matrix &m)

{

int i, j;

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

for(j = 0; j < col; j++)

elems[i][j] = m.elems[i][j];

return \*this;

}

Matrix Matrix::operator ~ () const

{

Matrix temp(line,col);

int i, j;

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

for(j = 0; j < col; j++)

temp.elems[i][j] = elems[j][i];

return temp;

}

void Matrix::setElems()

{

int i,j;

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

for(j = 0; j < col; j++)

cin>>elems[i][j];

}

void Matrix::print() const

{

int i,j;

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

{

for(j = 0; j < col; j++)

cout<<setw(4)<<elems[i][j];

cout<<endl;

}

}

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

{

int i, j;

Matrix temp(a.line,b.col);

for(i = 0;i < a.line; i++)

for(j = 0; j < b.col; j++)

{

temp.elems[i][j] = 0;

temp.elems[i][j] = a.elems[i][j] + b.elems[i][j];

}

return temp;

}

Matrix operator - (const Matrix &a,const Matrix &b)

{

int i, j;

Matrix temp(a.line,b.col);

for(i = 0;i < a.line; i++)

for(j = 0; j < b.col; j++)

{

temp.elems[i][j] = 0;

temp.elems[i][j] = a.elems[i][j] - b.elems[i][j];

}

return temp;

}

Matrix operator \* (const Matrix &a,const Matrix &b)

{ int i, j, k;

Matrix temp(a.line,b.col);

for(i = 0;i < a.line; i++)

for(j = 0; j < b.col; j++)

{

temp.elems[i][j] = 0;

for(k = 0; k < a.col; k++)

temp.elems[i][j] = temp.elems[i][j] + a.elems[i][k]\*b.elems[k][j];

}

return temp;

}

int main()

{

Matrix a(3,3), b(3,3);

cout<<"请输入第一个矩阵的元素："<<endl;

a.setElems();

cout<<"请输入第一个矩阵的元素："<<endl;

b.setElems();

Matrix c(3,3);

cout<<"\n两矩阵的和为："<<endl;

c = a + b;

c.print();

cout<<"\n两矩阵的差为："<<endl;

c = a - b;

c.print();

cout<<"\n两矩阵的成绩为："<<endl;

c = a \* b;

c.print();

cout<<"\n转置矩阵为："<<endl;

(~c).print();

return 0;

}

题目四：比较日期大小

【问题描述】

在实验3-2的基础上编程实现日期的关系运算，包括比较两日期是否相等、比较两日期的大小。

【实验输出】

实验输出效果如图3-5所示。



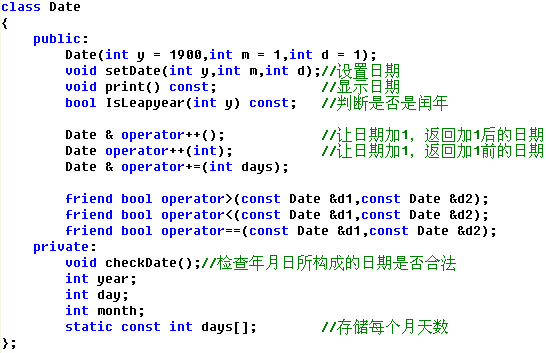
图3-5

【实验思路】

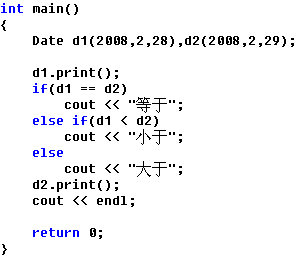
由于关系运算符都是双目运算符，按照运算符重载方针，应被重载为友元函数。这里需要重载的关系运算符有“<”、“>”、“==”,这些运算符的左右操作数都是const Date & 类型。

代码模版：

（1）Date类声明



（2）main()的实现



#include<iostream.h>

//using namespace std;

class Date

{

public:

Date(int y = 1900,int m = 1,int d = 1);

void setDate(int y,int m,int d);

void print() const;

bool IsLeapyear(int y) const;

Date & operator ++();

Date operator ++(int);

Date & operator +=(int days);

friend bool operator > (const Date &d1,const Date &d2);

friend bool operator < (const Date &d1,const Date &d2);

friend bool operator == (const Date &d1,const Date &d2);

private:

void checkDate();

int year;

int month;

int day;

static const int days[];

};

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

{

year = y;

month = m;

day = d;

}

const int Date::days[] = {0,31,28,31,30,31,30,31,31,30,31,30,31};

void Date::checkDate()

{

if(month == 4 || month == 6 || month == 9 || month == 11)

{

if(day == 31)

{

day = 1;

month++;

}

}

else if(month == 2)

{

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

{

if(day == 30)

{

day = 1;

month++;

}

}

else

{

if(day == 29)

{

day = 1;

month++;

}

}

}

else

{

if(day == 32)

day = 1;

month++;

}

if(month == 13)

{

month = 1;

year++;

}

}

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

{

year = (y >= 1900 && y <= 2100 ? y : 1900);

month = (m >= 1 && m <= 12 ? m : 1);

if(IsLeapyear(year) && month == 2)

day = (d >= 1 && d <= 29 ? d : 1);

else

day = (d >= 1 && d <= days[month] ? d : 1);

}

void Date::print() const

{

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

}

bool Date::IsLeapyear(int y) const

{

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

return true;

else

return false;

}

Date & Date::operator ++()

{

day++;

checkDate();

return \*this;

}

Date Date::operator ++(int)

{

Date temp(\*this);

day++;

checkDate();

return temp;

}

Date & Date::operator +=(int days)

{

days++;

checkDate();

return \*this;

}

bool operator > (const Date &d1,const Date &d2)

{

if(d1.day > d2.day)

return true;

else

return false;

}

bool operator < (const Date &d1,const Date &d2)

{

if(d1.day < d2.day)

return true;

else

return false;

}

bool operator == (const Date &d1,const Date &d2)

{

if(d1.day == d2.day)

return true;

else

return false;

}

int main()

{

Date d1(2008,2,28),d2(2008,2,29);

d1.print();

if(d1 == d2)

cout<<"等于";

else if(d1 < d2)

cout<<"小于";

else

cout<<"大于";

d2.print();

return 0;

}

实验四

题目一：模拟智能电表

【问题描述】

电表的主要功能是计量电能消耗。只能预付电表是常见的一种电表，必须现在其中预存一定金额后才合闸供电。用电时，电表一边计量电能消耗一边从剩余值中扣减已用金额，扣完则断电。

设计电表类Ammeter，其数据成员包括电表当前度数reading、电费价位price，成员函数有构造函数、显示信息方法的print()和计电表流量的方法setReading()。从Ammeter类派生出智能电表类Smartmeter，其派生的数据成员有预付款prepaid、剩余款balance，成员函数有构造函数、存入预付款方法setPrepaid()、计算剩余款的方法CalcCharge()、显示信息方法print()。其中，方法CalcCharge()在每耗电1度时被调用一次来计算剩余款。

编写程序测试智能电表的功能，已知电费单价为0.48元。

【实验输出】

实验输出效果如图4-1所示。



图4-1

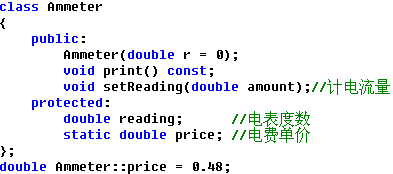
【实验思路】

题意已明确说明应实现哪些类以及这些类之间的关系。实现基类Ammeter时，应注意电费单价是该类所有对象的共享数据，应被定义为静态成员。调用setReading(double amount)计量度数时，应确保度数是非负值。两个类的构造函数应带默认参数值，提供灵活的初始化方式。

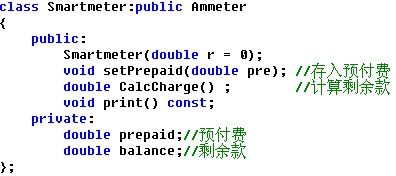
实现派生类Smartmeter时，应注意预付款最好不要在构造函数中通过参数初始化，而应直接初始化为0，若要存入预付款，必须调用专门的方法setPrepaid(double pre)完成，并且应注意存入的预付款不能为负值。

代码模版：

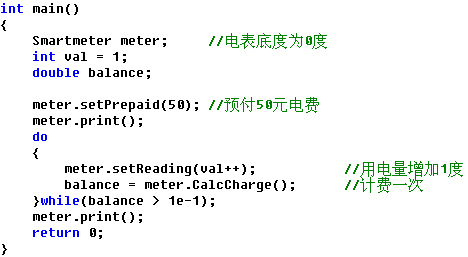
（1）Ammeter类声明



（2）Smartmeter类声明



（3）main()的实现



#include<iostream>

using namespace std;

class Ammeter

{

public:

Ammeter(double r = 0);

void print() const;

void setReading(double amount);

double get\_price();

protected:

double reading;

static double price;

};

double Ammeter::price = 0.48;

Ammeter::Ammeter(double r)

{

reading = r;

}

void Ammeter::print() const

{

cout<<"度数："<<reading<<" "<<"单价："<<price<<" ";

}

void Ammeter::setReading(double amount)

{

reading = amount;

}

double Ammeter::get\_price()

{

return price;

}

class Smartmeter:public Ammeter

{

public:

Smartmeter(double p = 0);

void setPrepaid(double pre);

double CalcCharge();

void print() const;

private:

double prepaid;

double balance;

};

Smartmeter::Smartmeter(double r):Ammeter(r)

{

prepaid = 0;

balance = prepaid;

}

void Smartmeter::setPrepaid(double pre)

{

prepaid += pre;

balance += pre;

}

double Smartmeter::CalcCharge()

{

balance = balance - price;

return balance;

}

void Smartmeter::print() const

{

Ammeter::print();

cout<<"预付款："<<prepaid<<" "<<"剩余款："<<balance<<endl;

}

int main()

{

Smartmeter meter;

int val = 1;

double balance;

double price = meter.get\_price();

meter.setPrepaid(50);

meter.print();

do

{

meter.setReading(val++);

balance = meter.CalcCharge();

}while(balance > meter.get\_price());

meter.print();

return 0;

}

题目二：管理快递资费

【问题描述】

某快递公司提供两类寄件方式：快件和当日达件。快件的资费标准是：起重1kg及以内收费15元，续重8元/kg。当日达件的资费标准是：起重2kg及以内收费40元，续重25元/kg。当日达件若未按时（24小时内）到达，应全额赔付客户所付运费并将寄件送达。

设计包裹类Package，它具有发件人、收件人的姓名、地址、电话号码、邮编以及包裹重量（以千克为单位）等数据成员，并具有构造函数、显示信息的方法print()。从Package类派生出快递类Express和限时快递类SpecialExpress。两个派生类除构造函数外，均具有CalcCost()方法来计算邮资、print()方法来显示信息，还具有相应的数据成员存储起重资费和续重资费。此外，SpecialExpress类还具有计算全额赔付的成员函数payment()和存储送件时长的数据成员deliveryHour。编写一个测试程序，完成上述功能测试。

【实验输出】

实验输出效果如图4-2所示。



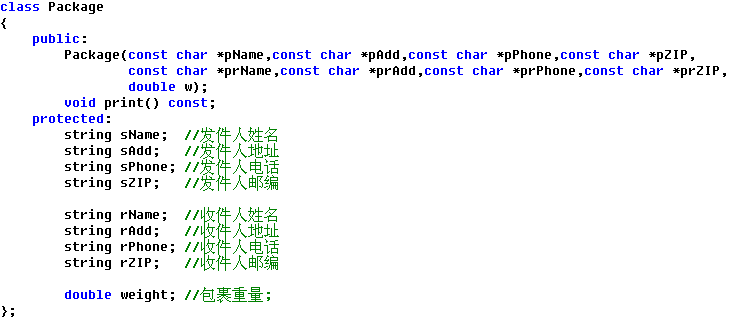
图4-2

【实验思路】

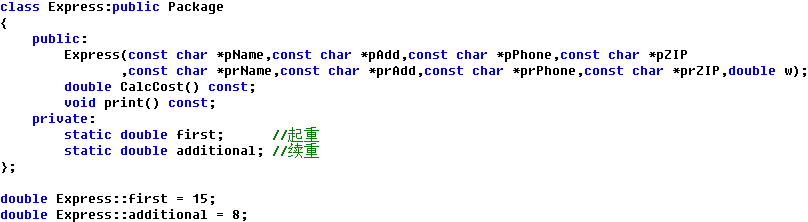
本题涉及三个类：基类Package、派生类Express、派生类SpecialExpress。设计时，先实现基类，再实现派生类。由于基类有8个对象成员，从效率考虑，应将对象成员的初始化放在构造函数的初始化表中执行，并注意检查货物重量是否为负数。print()方法打印货物信息的格式可模拟快递签收单的格式。派生类的邮费计算应注意，无论是起重资费还是续重资费，货重的小数部分（不足1kg部分）应按1kg收费。对于SpecialExpress类，若寄件时长超过24小时，应全额赔付顾客邮费。

代码模版：

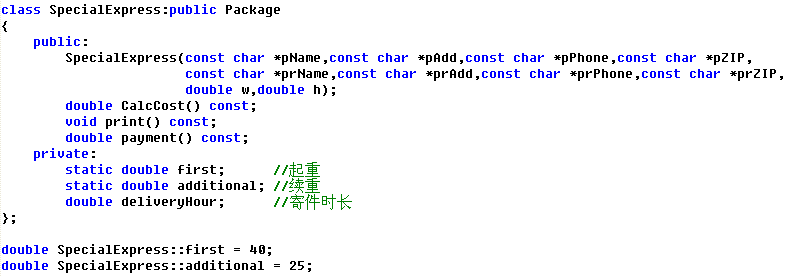
（1）Package类声明



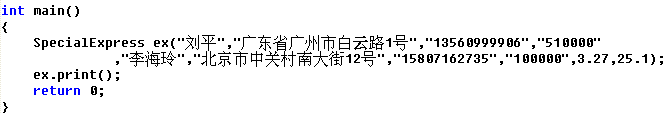
（2）Express类声明



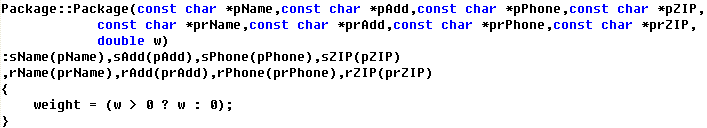
（3）SpecialExpress类声明

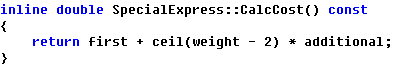


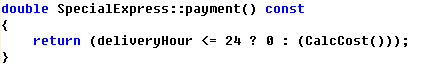
（4）main()的实现



（5）部分成员函数实现







#include<iostream>

#include<string>

#include<cmath>

using namespace std;

class Package

{

public:

Package(const char \*pName,const char \*pAdd,const char \*pPhone,const char \*pZIP,

const char \*prName,const char \*prAdd,const char \*prPhone,const char \*prZIP,double w);

void print() const;

protected:

string sName;

string sAdd;

string sPhone;

string sZIP;

string rName;

string rAdd;

string rPhone;

string rZIP;

double weight;

};

Package::Package(const char \*pName,const char \*pAdd,const char \*pPhone,const char \*pZIP,

const char \*prName,const char \*prAdd,const char \*prPhone,const char \*prZIP,double w)

:sName(pName),sAdd(pAdd),sPhone(pPhone),sZIP(pZIP),rName(prName),rAdd(prAdd),rPhone(prPhone),rZIP(prZIP)

{

weight = (w > 0 ? w : 0);

}

void Package::print() const

{

cout<<"收件人信息"<<endl<<"-------------------------------"<<endl;

cout<<"姓名："<<rName<<" "<<"地址："<<rAdd<<endl;

cout<<"电话："<<rPhone<<" "<<"邮编："<<rZIP<<endl<<endl;

cout<<"寄件人信息"<<endl<<"-------------------------------"<<endl;

cout<<"姓名："<<sName<<" "<<"地址："<<sAdd<<endl;

cout<<"电话："<<sPhone<<" "<<"邮编："<<sZIP<<endl<<endl;

}

class Express:public Package

{

public:

Express(const char \*pName,const char \*pAdd,const char \*pPhone,const char \*pZIP,

const char \*prName,const char \*prAdd,const char \*prPhone,const char \*prZIP,double w);

double CalcCost() const;

void print() const;

private:

static double first;

static double additional;

};

Express::Express(const char \*pName,const char \*pAdd,const char \*pPhone,const char \*pZIP,

const char \*prName,const char \*prAdd,const char \*prPhone,const char \*prZIP,double w)

:Package(pName,pAdd,pPhone,pZIP,prName,prAdd,prPhone,prZIP,w){}

double Express::CalcCost() const

{

return first + ceil(weight -1)\*additional;

}

void Express::print() const

{

Package::print();

cout<<"货重："<<weight<<" "<<"邮费："<<CalcCost()<<endl;

}

double Express::first = 15;

double Express::additional = 8;

class SpecialExpress:public Package

{

public:

SpecialExpress(const char \*pName,const char \*pAdd,const char \*pPhone,const char \*pZIP,

const char \*prName,const char \*prAdd,const char \*prPhone,const char \*prZIP,double w,double h);

double CalcCost() const;

void print() const;

double payment() const;

private:

static double first;

static double additional;

double deliveryHour;

};

SpecialExpress::SpecialExpress(const char \*pName,const char \*pAdd,const char \*pPhone,const char \*pZIP,

const char \*prName,const char \*prAdd,const char \*prPhone,const char \*prZIP,double w,double h)

:Package(pName,pAdd,pPhone,pZIP,prName,prAdd,prPhone,prZIP,w)

{

deliveryHour = h;

}

inline double SpecialExpress::CalcCost() const

{

return first + ceil(weight - 2) \* additional;

}

void SpecialExpress::print() const

{

Package::print();

cout<<"货重："<<weight<<" "<<"邮费："<<CalcCost()<<" "<<"赔付："<<payment()<<endl;

}

double SpecialExpress::payment() const

{

return (deliveryHour <= 24 ? 0 : (CalcCost()));

}

double SpecialExpress::first = 40;

double SpecialExpress::additional = 25;

int main()

{

SpecialExpress ex("刘平","广东省广州市白云路1号","13560999906","510000",

"李海玲","北京市中关村南大街12号","15807162735","100000",3.27,25);

ex.print();

return 0;

}

题目三：设计实用整型数组

【问题描述】

实现具有如下功能的整型数组：（1）带有小标越界检查功能的数组；（2）可排序的数组；（3）既可进行边界检查又可排序的数组。

【实验输出】

实验输出效果如图4-3所示。

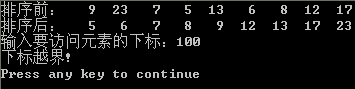


图4-3

【实验思路】

本实验涉及4个类：整型数组类Array、带越界检查的数组类ChkArray、排序数组类SortArray、既越界检查又排序的数组类ChkSrtArray。其中，Array类是共同基类，由它派生出直接基类ChkArray和SortArray，再由这两个派生出最终派生类ChkSrtArray类。为避免造成多继承情况下成员访问的二义性，应将Array类定义为虚基类。

首先创建基类Array，它包含的数据成员有数组长度len和指向数组元素的指针elems，它能在构造函数中根据数组长度动态创建数组元素，其他成员函数有：析构函数、求数组长度的方法length()、返回下标i对应元素的方法at(int i)。注意，数组长度不能为负。

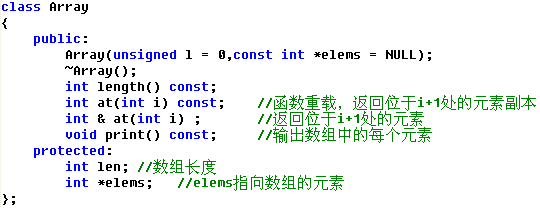
直接基类ChkArray唯一的特殊行为是在执行下标问操作at()时，会用该类特有的方法BorderCheck()进行越界检查。

直接基类SortArray能根据用户的要求进行升序、降序排序，该功能由公有成员函数sort(bool asc = true)实现，其中参数asc值为true表示按照升序排列，false表示降序排列。

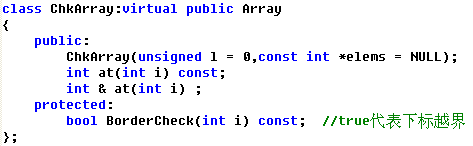
最终派生类ChkSrtArray很简单，只需要有构造函数即可。

代码模版：

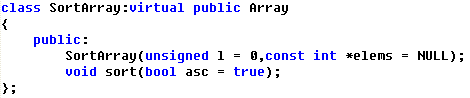
（1）Array类声明



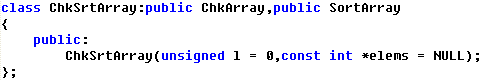
（2）ChkArray类声明



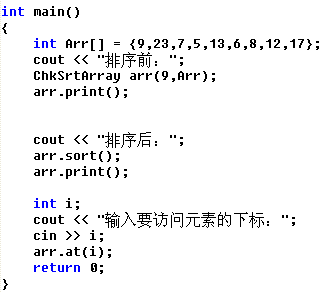
（3）SortArray类声明



（4）SortArray类声明



（5）main()的实现



#include<iostream>

#include<iomanip>

#include<cstdlib>

using namespace std;

class Array

{

public:

Array(unsigned l=0,const int \*e=NULL);

~Array();

int length() const;

int at(int i) const;

int & at(int i);

void print() const;

protected:

int len;

int \*elems;

};

Array::Array(unsigned l,const int \*e)

{

len = l;

elems = NULL;

if(len > 0)

{

elems = new int[len];

if(e == NULL)

{

cout<<"申请空间失败！"<<endl;

exit(0);

}

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

elems[i] = e[i];

}

}

Array::~Array()

{

delete[] elems;

}

int Array::length() const

{

return len;

}

int Array::at(int i) const

{

return elems[i-1];

}

int & Array::at(int i)

{

return elems[i-1];

}

void Array::print() const

{

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

cout<<setw(5)<<elems[i];

cout<<endl;

}

class ChkArray:virtual public Array

{

public:

ChkArray(unsigned l=0,const int \*e=NULL);

int at(int i) const;

int & at(int i);

protected:

bool BorderCheck(int i) const;

};

ChkArray::ChkArray(unsigned l,const int \*e):Array(l,e){}

int ChkArray::at(int i) const

{

if(BorderCheck(i) == true)

{

cout<<"下标越界！"<<endl;

return elems[0];

}

else

return elems[i-1];

}

int & ChkArray::at(int i)

{

if(BorderCheck(i) == true)

{

cout<<"下标越界！"<<endl;

return elems[0];

}

else

return elems[i-1];

}

bool ChkArray::BorderCheck(int i) const

{

if(i > len || i <= 0)

return true;

else

return false;

}

class SortArray:virtual public Array

{

public:

SortArray(unsigned l=0,const int \*e=NULL);

void sort(bool asc=true);

};

SortArray::SortArray(unsigned l,const int \*e):Array(l,e){}

void SortArray::sort(bool asc)

{

int i,j,t;

if(asc == true)

{

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

for(j = i+1; j < len; j++)

if(elems[i]>elems[j])

{

t = elems[i];

elems[i] = elems[j];

elems[j] = t;

}

}

else

{

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

for(j = i+1; j < len; j++)

if(elems[i]<elems[j])

{

t = elems[i];

elems[i] = elems[j];

elems[j] = t;

}

}

}

class ChkSrtArray:public ChkArray,public SortArray

{

public:

ChkSrtArray(unsigned l=0,const int \*e=NULL);

};

ChkSrtArray::ChkSrtArray(unsigned l,const int \*e):Array(l,e),ChkArray(l,e),SortArray(l,e){}

int main()

{

int Arr[]={9,23,7,5,13,6,8,12,17};

cout<<"排序前：";

ChkSrtArray arr(9,Arr);

arr.print();

cout<<"排序后：";

arr.sort();

arr.print();

int i;

cout<<"输入要访问元素的下标：";

cin>>i;

arr.at(i);

return 0;

}

题目四：图形求面积

【问题描述】

设计一个能细分为矩形、三角形、圆形和椭圆形的“图形”类。使用继承将这些图形分类，找出能作为基类部分的共同特征（如宽、高、中心点等）和方法（如初始化、求面积等）。

【实验输出】

实验输出效果如图4-4所示。

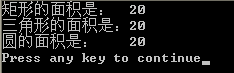


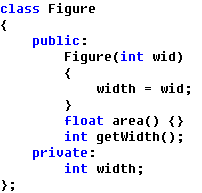
图4-4

【实验思路】

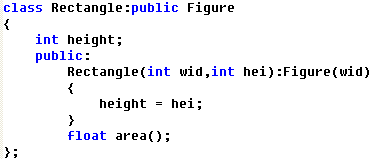
定义一个基类，名为Figure（图形）类，类的数据成员有定义宽度或半径的整型数据width。从他派生出Rectangle（矩形）类，Triangle（三角形）类，Circle（圆）类。

代码模版：

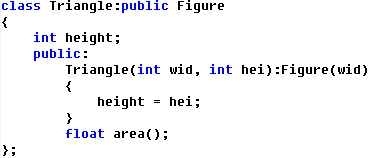
（1）Figure类声明



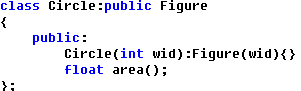
（2）Rectangle类声明



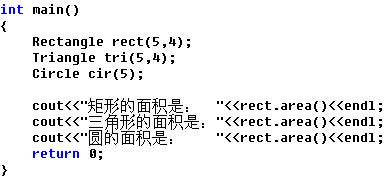
（3）Triangle类声明



（4）Circle类声明



（5）main函数



#include<iostream>

using namespace std;

class Figure

{

public:

Figure(int wid)

{

width=wid;

}

float area(){}

int getWidth();

private:

int width;

};

int Figure::getWidth()

{

return width;

}

class Rectangle:public Figure

{

int height;

public:

Rectangle(int wid,int hei):Figure(wid)

{

height=hei;

}

float area();

};

float Rectangle::area()

{

return getWidth()\*height;

}

class Triangle:public Figure

{

int height;

public:

Triangle(int wid,int hei):Figure(wid)

{

height=hei;

}

float area();

};

float Triangle::area()

{

return getWidth()\*height/2;

}

class Circle:public Figure

{

public:

Circle (int wid):Figure(wid){}

float area();

};

float Circle::area()

{

return 3.14\*getWidth()\*getWidth();

}

int main()

{

Rectangle rect(5,4);

Triangle tri(5,4);

Circle cir(5);

cout<<"矩形的面积： "<<rect.area()<<endl;

cout<<"三角形的面积："<<tri.area()<<endl;

cout<<"圆的面积： "<<cir.area()<<endl;

return 0;

}

实验五

题目一：计算教学工作量

【问题描述】

大学的专业技术职称主要分初级、中级、高级。具有初级职称的教师从事实验教学工作（含实验室设备维护工作），具有中级职称的教师从事本科理论教学工作，具有高级职称的教师既从事本科理论教学工作，也从事研究生教学工作。各级教学工作量计算公式及系数如下：

实验教学工作量 = 教日数 × （实验教学系数 + 设备维护系数）

本科理论教学工作量 = 教日数 × 本科理论教学系数

研究生教学工作量 = 教日数 × 研究生教学系数

各级岗位教学系数如表所示：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **高 级** | | **中 级** | **初 级** | |
| 研究生 | 本科生 | 60 | 实验 | 维护 |
| 95 | 80 | 40 | 10 |

编写程序计算各级职称的工作量。

【实验输出】

实验输出效果如图5-1所示。



图5-1

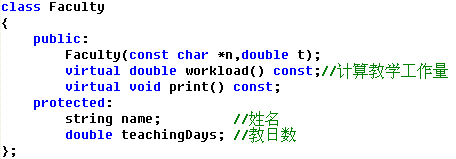
【实验思路】

由题意可知，应设计基类Faculty代表全体教员，其数据成员包括姓名和教日数，成员函数包括构造函数、计算工作量的方法workload()和显示信息的方法print()，其中后两个方法是虚函数。

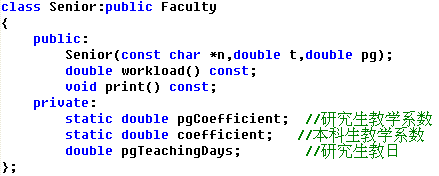
从Faculty类派生出高级职称类Senior、中级职称类Middle、初级职称类Primary。由于每类职称的系数均是固定不变的数据，故所有系数数据成员均应定义为类的静态成员。此外，还应为Senior类增加代表研究生教日数的数据成员。所有派生类的成员函数主要是构造函数和重新改写的虚函数workload()、print()。

【代码模版】

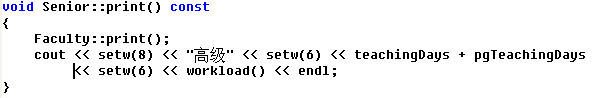
（1）Faculty类声明



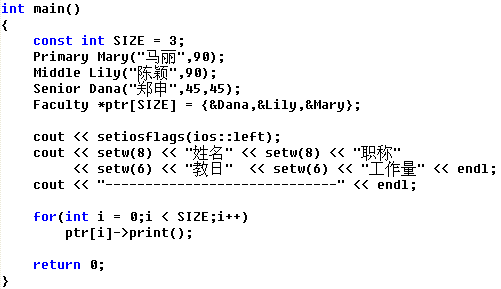
（2）Senior类声明



（3）Senior::print()



（4）main()的实现



#include<iostream>

#include<string>

#include<iomanip>

using namespace std;

class Faculty

{

public:

Faculty(const char \*n,double t);

virtual double workload() const;

virtual void print() const;

protected:

string name;

double teachingDays;

};

Faculty::Faculty(const char \*n,double t)

{

name = n;

teachingDays = t;

}

double Faculty::workload() const

{

return 0;

}

void Faculty::print() const

{

cout<<setiosflags(ios::left);

cout<<setw(8)<<name;

}

class Senior:public Faculty

{

public:

Senior(const char \*n,double t,double pg);

double workload() const;

void print() const;

private:

static double pgCoefficient;

static double coefficient;

double pgTeachingDays;

};

Senior::Senior(const char \*n,double t,double pg):Faculty(n,t)

{

pgTeachingDays = pg;

}

double Senior::workload() const

{

return teachingDays\*coefficient+pgTeachingDays\*pgCoefficient;

}

void Senior::print() const

{

Faculty::print();

cout<<setw(8)<<"高级"<<setw(6)<<teachingDays+pgTeachingDays

<<setw(6)<<workload()<<endl;

}

double Senior::pgCoefficient = 95;

double Senior::coefficient = 80;

class Middle:public Faculty

{

public:

Middle(const char \*n,double t);

double workload() const;

void print() const;

private:

static double coefficient;

};

Middle::Middle(const char \*n,double t):Faculty(n,t){}

double Middle::workload() const

{

return teachingDays\*coefficient;

}

void Middle::print() const

{

Faculty::print();

cout<<setw(8)<<"中级"<<setw(6)<<teachingDays

<<setw(6)<<workload()<<endl;

}

double Middle::coefficient = 60;

class Primary:public Faculty

{

public:

Primary(const char \*n,double t);

double workload() const;

void print() const;

private:

static double coefficient;

static double coefficient1;

};

Primary::Primary(const char \*n,double t):Faculty(n,t){}

double Primary::workload() const

{

return teachingDays\*(coefficient+coefficient1);

}

void Primary::print() const

{

Faculty::print();

cout<<setw(8)<<"初级"<<setw(6)<<teachingDays

<<setw(6)<<workload()<<endl;

}

double Primary::coefficient = 40;

double Primary::coefficient1 = 10;

int main()

{

const int SIZE = 3;

Primary Mary("马丽",90);

Middle Lily("陈颖",90);

Senior Dana("郑申",45,45);

Faculty \*ptr[SIZE] = {&Dana,&Lily,&Mary};

cout<<setiosflags(ios::left);

cout<<setw(8)<<"姓名"<<setw(8)<<"职称"

<<setw(6)<<"教日"<<setw(6)<<"工作量"<<endl;

cout<<"--------------------------"<<endl;

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

ptr[i]->print();

return 0;

}

题目二：计算快递公司日营业额

【问题描述】

适当修改实验四的题目二，使该程序能计算快递公司当天所收包裹的邮资总额。假设快递公司当天总共发出3个包裹，其中2个是快件，1个是当日达件。

【实验输出】

实验输出效果如图5-2所示。

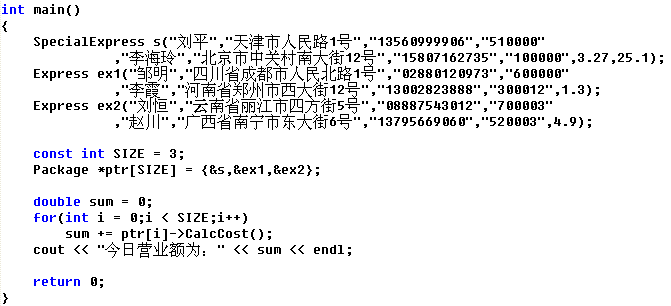


图5-2

【实验思路】

计算日营业额所需要调用派生类的CalcCost()方法来计算每个包裹的邮资，该功能可通过多态来实现，只需要在主函数的循环中反复调用“ptr[i] -> CalcCost()”即可。为此，需要将基类Package的CalcCost()声明为虚函数，实验四第二题中其他派生类的代码可维持不变。

【代码模版】



#include<iostream>

#include<string>

#include<cmath>

using namespace std;

class Package

{

public:

Package(const char \*pName,const char \*pAdd,const char \*pPhone,const char \*pZIP,

const char \*prName,const char \*prAdd,const char \*prPhone,const char \*prZIP,double w);

void print() const;

virtual double CaclCost() const;

protected:

string sName;

string sAdd;

string sPhone;

string sZIP;

string rName;

string rAdd;

string rPhone;

string rZIP;

double weight;

};

Package::Package(const char \*pName,const char \*pAdd,const char \*pPhone,const char \*pZIP,

const char \*prName,const char \*prAdd,const char \*prPhone,const char \*prZIP,double w)

:sName(pName),sAdd(pAdd),sPhone(pPhone),sZIP(pZIP),rName(prName),rAdd(prAdd),rPhone(prPhone),rZIP(prZIP)

{

weight = (w > 0 ? w : 0);

}

void Package::print() const

{

cout<<"收件人信息"<<endl<<"-------------------------------"<<endl;

cout<<"姓名："<<rName<<" "<<"地址："<<rAdd<<endl;

cout<<"电话："<<rPhone<<" "<<"邮编："<<rZIP<<endl<<endl;

cout<<"寄件人信息"<<endl<<"-------------------------------"<<endl;

cout<<"姓名："<<sName<<" "<<"地址："<<sAdd<<endl;

cout<<"电话："<<sPhone<<" "<<"邮编："<<sZIP<<endl<<endl;

}

double Package::CaclCost() const

{

return 0;

}

class Express:public Package

{

public:

Express(const char \*pName,const char \*pAdd,const char \*pPhone,const char \*pZIP,

const char \*prName,const char \*prAdd,const char \*prPhone,const char \*prZIP,double w);

double CaclCost() const;

void print() const;

private:

static double first;

static double additional;

};

Express::Express(const char \*pName,const char \*pAdd,const char \*pPhone,const char \*pZIP,

const char \*prName,const char \*prAdd,const char \*prPhone,const char \*prZIP,double w)

:Package(pName,pAdd,pPhone,pZIP,prName,prAdd,prPhone,prZIP,w){}

double Express::CaclCost() const

{

return first + ceil(weight -1)\*additional;

}

void Express::print() const

{

Package::print();

cout<<"货重："<<weight<<" "<<"邮费："<<CaclCost()<<endl;

}

double Express::first = 15;

double Express::additional = 8;

class SpecialExpress:public Package

{

public:

SpecialExpress(const char \*pName,const char \*pAdd,const char \*pPhone,const char \*pZIP,

const char \*prName,const char \*prAdd,const char \*prPhone,const char \*prZIP,double w,double h);

double CaclCost() const;

void print() const;

double payment() const;

private:

static double first;

static double additional;

double deliveryHour;

};

SpecialExpress::SpecialExpress(const char \*pName,const char \*pAdd,const char \*pPhone,const char \*pZIP,

const char \*prName,const char \*prAdd,const char \*prPhone,const char \*prZIP,double w,double h)

:Package(pName,pAdd,pPhone,pZIP,prName,prAdd,prPhone,prZIP,w)

{

deliveryHour = h;

}

inline double SpecialExpress::CaclCost() const

{

return first + ceil(weight - 2) \* additional;

}

void SpecialExpress::print() const

{

Package::print();

cout<<"货重："<<weight<<" "<<"邮费："<<CaclCost()<<" "<<"赔付："<<payment()<<endl;

}

double SpecialExpress::payment() const

{

return (deliveryHour <= 24 ? 0 : (CaclCost()));

}

double SpecialExpress::first = 40;

double SpecialExpress::additional = 25;

int main()

{

SpecialExpress s("刘平","广东省广州市白云路1号","13560999906","510000",

"李海玲","北京市中关村南大街12号","15807162735","100000",3.27,25.1);

Express ex1("邹明","四川省成都市人民北路1号","02880120973","600000"

,"李霞","河南省郑州市西大街12号","13002823888","300012",1.3);

Express ex2("刘恒","云南省丽江市四方街5号","08887543012","700003"

,"赵川","广西省南宁市东大街6号","13795669060","520003",4.9);

const int SIZE = 3;

Package \*ptr[SIZE] = {&s,&ex1,&ex2};

double sum = 0;

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

sum += ptr[i]->CaclCost();

cout<<"今日营业额为："<<sum<<endl;

return 0;

}

题目三：计算图形面积

【问题描述】

此处所指图形可分为两大类：二维图形和三维图形。二维图形包括圆、矩形等。三维图形包括球体、立方体等。编程计算二维图形的面积、三维图形的表面积和体积。

【实验输出】

实验输出效果如图5-3所示。

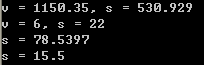
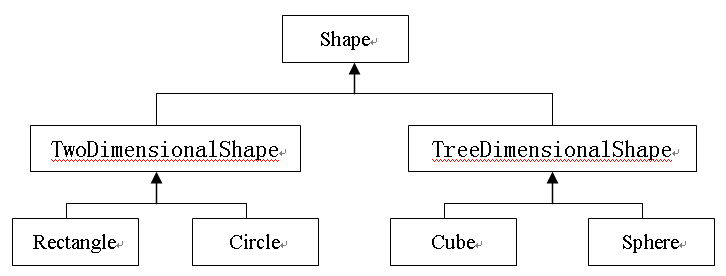


图5-3

【实验思路】

问题的类层次结构如图所示。



基类Shape代表抽象概念“图形”，因此把Shape类定义为抽象类，它为整个类簇提供计算面积（表面积）的接口getArea()。同理，二维图形类TwoDimensionalShape和三维图形类TreeDimensionalShape也应被定义为抽象类，其中三维图形类还应为后续派生类提供计算体积的接口getVolume()。所有具体的图形类，如矩形、球体等，均应从抽象类TwoDimensionalShape或TreeDimensionalShape派生而来。

对应面积或表面积的计算，无论哪类图形，只需要调用getArea()即可实现。但三维图形还需要调用getVolume()计算体积。这里介绍两种求解本题的方法。

在主函数中调用统一接口“ptr[i] -> print()”。在Shape类中再定义一个纯虚函数print()。二维图形的派生类在实现该虚函数时，只需要在print()方法中调用getArea()方法打印二维图形的面积，即:

void Cricle::print() const

{

cout << "s = " << getArea() << endl;

}

而三围图形的派生类在print()方法中 调用getArea()和getVolume()打印三维图形的表面积和体积，即：

void Cube::print() const

{

cout << "s = " << getArea() << ", v = " << getVolume() << endl;

}

这样就可以在主函数中保持统一的接口“ptr[i] -> print()”：

int main()

{

Shape \*ptr[3] = {&Shpere(6.5), &Cube(3,2,1), &Circle(5)};

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

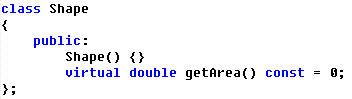
ptr[i] -> print();

return 0;

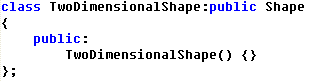
}

【代码模版】

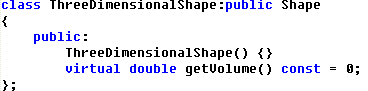
（1）Shape类声明



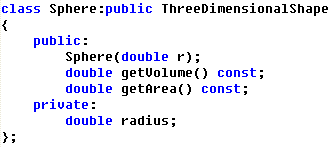
（2）TwoDimensionalShape类声明



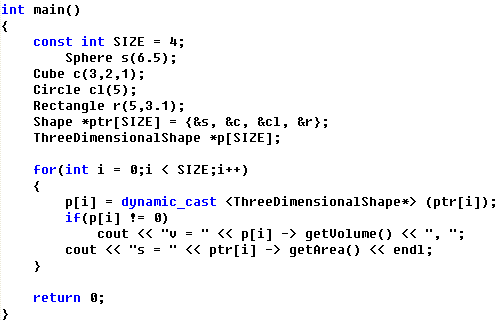
（3）TreeDimensionalShape类声明



（4）Sphere类声明



（5）main()的实现



#include<iostream>

#define PI 3.14159

using namespace std;

class Shape

{

public:

Shape() {}

virtual double getArea() const = 0;

virtual void print() const = 0;

};

class TwoDimensionalShape:public Shape

{

public:

TwoDimensionalShape() {}

};

class Circle:public TwoDimensionalShape

{

public:

Circle(double);

double getArea() const;

void print() const;

private:

double radius;

};

Circle::Circle(double r)

{

radius = r;

}

double Circle::getArea() const

{

double area;

area = PI\*radius\*radius;

return area;

}

void Circle::print() const

{

cout<<"s = "<<getArea()<<endl;

}

class Rectangle:public TwoDimensionalShape

{

public:

Rectangle(double, double);

double getArea() const;

void print() const;

private:

double length;

double width;

};

Rectangle::Rectangle(double l,double w)

{

length = l;

width = w;

}

double Rectangle::getArea() const

{

return length\*width;

}

void Rectangle::print() const

{

cout<<"s = "<<getArea()<<endl;

}

class ThreeDimensionalShape:public Shape

{

public:

ThreeDimensionalShape(){}

virtual double getVolume() const = 0;

};

class Sphere:public ThreeDimensionalShape

{

public:

Sphere(double);

double getVolume() const;

double getArea() const;

void print() const;

private:

double radius;

};

Sphere::Sphere(double r)

{

radius = r;

}

double Sphere::getVolume() const

{

double vol;

vol = PI\*radius\*radius\*radius\*4/3;

return vol;

}

double Sphere::getArea() const

{

double area;

area = PI\*radius\*radius\*4;

return area;

}

void Sphere::print() const

{

cout<<"v = "<<getVolume()<<", ";

cout<<"s = "<<getArea()<<endl;

}

class Cube:public ThreeDimensionalShape

{

public:

Cube(double,double,double);

double getVolume() const;

double getArea() const;

void print() const;

private:

double length;

double width;

double height;

};

Cube::Cube(double l,double w,double h)

{

length = l;

width = w;

height = h;

}

double Cube::getVolume() const

{

return length\*width\*height;

}

double Cube::getArea() const

{

return 2\*(length\*width+width\*height+height\*length);

}

void Cube::print() const

{

cout<<"v = "<<getVolume()<<", ";

cout<<"s = "<<getArea()<<endl;

}

int main()

{

const int SIZE = 4;

Sphere s(6.5);

Cube c(3,2,1);

Circle cl(5);

Rectangle r(5,3.1);

Shape \*ptr[SIZE]={&s, &c, &cl, &r};

/\*ThreeDimensionalShape \*p[SIZE];

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

{

p[i] = dynamic\_cast <ThreeDimensionalShape\*>(ptr[i]);

if(p[i]!=0)

cout<<"v = "<<p[i]->getVolume()<<",";

cout<<"s = "<<ptr[i]->getArea()<<endl;

}\*/

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

{

ptr[i] ->print();

}

return 0;

}

题目四：动物

【问题描述】

设计一个animal基类和它的派生类tiger（老虎）、sheep（羊），实现虚函数。

【实验输出】

实验输出效果如图5-4所示。

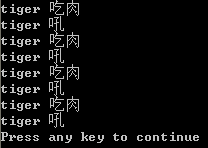


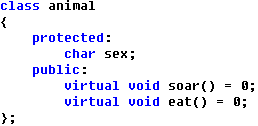
图5-4

【实验思路】

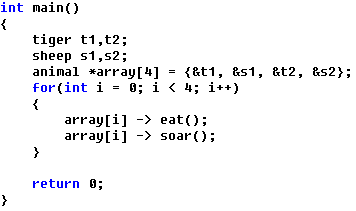
可自行定义这些基类的成员变量，并且animal基类中应有动物性别的成员变量，但要设定每种动物的叫soar()及吃eat()的成员函数，可用cout输出来表示。要求每个派生类生成两个对象，打乱次序存于一个数组中，然后用循环程序访问其叫与吃的成员函数，必须用到虚函数。

【代码模版】

（1）animal类的声明



（2）main函数



#include<iostream>

using namespace std;

class Animal

{ protected:

char sex;

public:

virtual void soar() = 0;

virtual void eat() = 0;

};

class Tiger:public Animal

{ public:

void soar();

void eat();

};

void Tiger::soar()

{ cout<<"tiger"<<" 咩"<<endl;

}

void Tiger::eat()

{ cout<<"tiger"<<" 吃肉"<<endl;

}

class Sheep:public Animal

{ public:

void soar();

void eat();

};

void Sheep::soar()

{ cout<<"sheep"<<" 吼"<<endl;

}

void Sheep::eat()

{ cout<<"sheep"<<" 吃草"<<endl;

}

int main()

{ Tiger t1,t2;

Sheep s1,s2;

Animal \*array[4]={&t1, &s1, &t2, &s2};

for(int i = 0; i < 4; i++)

{

array[i]->eat();

array[i]->soar();

}

return 0;

}