A：

头文件：

Class.h

#pragma once

#include<iostream>

#include<string>

#include<vector>

#include<math.h>

using namespace std;

class Poly {

public:

Poly(const vector<float>& b);//构造函数

~Poly() { member--; }//析构函数

Poly(const Poly& a);//拷贝构造函数

void calculate(int x);

void print() const;

static int member;//储存多项式的数量

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

Poly operator -(const Poly& a);

Poly& operator=(const Poly& a);

Poly operator \*(const Poly& a);

private:

vector<float> coef;//储存任意大的多项式

void deletecoef();

void copycoef(const vector<float>& a);

};

源文件：

Class.cpp

#include "Class.h"

#include<iostream>

int Poly::member = 0;

Poly::Poly(const vector<float>& b)

{

coef = b;

member++;

cout << "使用构造函数构造成功" << endl;

}

Poly::Poly(const Poly& a)

{

\*this = a;

member++;

cout << "使用拷贝构造函数复制成功" << endl;

}

void Poly::calculate(int x)

{

vector<float> ca(coef);

float sum = 0;

for (int i = 0; i < ca.size(); i++) {

sum += (ca[i] \* (float)pow(x, i));

}

cout << sum << endl;

return;

}

void Poly::print() const {

vector<float> ca(coef);

int n = (int)ca.size();

for (int i = n - 1; i >= 0; i--) {

if (i != 0) {

if (ca[i] > 0.0) cout << "+" << ca[i] << "x^" << i;

else if (ca[i] == 0.0) continue;

else if (ca[i] < 0.0) cout << ca[i] << "x^" << i;

}

else {

if (ca[i] > 0.0) cout << "+" << ca[i];

else if (ca[i] == 0.0) continue;

else if (ca[i] < 0.0) cout << ca[i];

}

}

cout << endl;

return;

}

Poly operator+(const Poly& a,const Poly& b) {

size\_t p = a.coef.size();

size\_t q = b.coef.size();

vector<float> sum(max(p,q),0);

if (p >= q) {

for (size\_t i = 0; i < q; i++) {

sum[i] = a.coef[i] + b.coef[i];

}

for (size\_t i = q; i < p; i++) {

sum[i] = a.coef[i];

}

}

else {

for (size\_t i = 0; i < p; i++) {

sum[i] = a.coef[i] + b.coef[i];

}

for (size\_t i = p; i < q; i++) {

sum[i] = b.coef[i];

}

}

Poly c(sum);

return c;

}

Poly Poly::operator-(const Poly& a) {

size\_t p = this->coef.size();

size\_t q = a.coef.size();

vector<float> sum(max(p,q), 0);

if (p >= q) {

for (size\_t i = 0; i < q; i++) {

sum[i] = this->coef[i] - a.coef[i];

}

for (size\_t i = q; i < p; i++) {

sum[i] = this->coef[i];

}

}

else {

for (size\_t i = 0; i < p; i++) {

sum[i] = this->coef[i] - a.coef[i];

}

for (size\_t i = p; i < q; i++) {

sum[i] = -a.coef[i];

}

}

Poly c(sum);

return c;

}

Poly& Poly::operator=(const Poly& a)

{

if (this == &a)

return \*this;

this->coef = a.coef;

return \*this;

}

void Poly::deletecoef() {

this->coef.clear();

}

void Poly::copycoef(const vector<float>& a) {

for (int i = 0; i < (int)a.size(); i++) {

coef[i] = a[i];

}

}

Poly Poly::operator\*(const Poly& a) {

size\_t b = this->coef.size() + a.coef.size();

vector<float> pro(b, 0);

for (size\_t i = 0; i < this->coef.size(); i++) {

for (size\_t j = 0; j < a.coef.size(); j++) {

pro[i + j] = this->coef[i] \* a.coef[j];

}

}

Poly p(pro);

return p;

}

源.cpp

#include<iostream>

#include"Class.h"

using namespace std;

int main() {

system("color f0");

float a[7] = { 1,2,3,4,5,6,7 };

float b[8] = { 1,9,0,0,0,0,0,4 };

vector<float> va(a, a + 7);

vector<float> vb(b, b + 8);

Poly x(va);

Poly y(vb);

cout << "第一个多项式：";

x.print();

cout << "第二个多项式：";

y.print();

cout << "给第一个多项式赋值x为2：";

x.calculate(2);

cout << endl;

Poly z(va);

z = y;

cout << "以第二个多项式赋值创建第三个多项式"<<endl;

z.print();

cout << endl;

Poly sum = x + y;

cout << "加法结果:" << endl;

sum.print();

cout << endl;

Poly dif=x-y;

cout << "减法结果:" << endl;

dif.print();

cout << endl;

Poly pro = x \* y;

cout << "乘法结果:" << endl;

pro.print();

cout << "此时一共建立了的多项式总数：" <<Poly::member<< endl;

return 0;

}

B：

头文件：

String.h

#pragma once

#include<iostream>

#include<string>

#include<cstring>

using namespace std;

class String

{

private:

char \*s\_data;

size\_t s\_size;

public:

void print() const;

String(const char \*str=nullptr);

String(const String& str);

~String();

String& operator=(const String& str);

size\_t legnth();

String& cat(const String& str);

};

源文件：

String.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "String.h"

#include<cstring>

#include<iostream>

String::String(const char\* str) {

if (str == nullptr) {

s\_data = new char[1];

s\_data[0] = '\0';

s\_size = 0;

}

else {

s\_size = strlen(str);

s\_data = new char[s\_size + 1];

strcpy\_s(s\_data, s\_size + 1, str);

}

}

String::String(const String& str) {

s\_size = str.s\_size;

s\_data = new char[s\_size + 1];

strcpy\_s(s\_data, s\_size + 1, str.s\_data);

}

String::~String() {

delete[] s\_data;

}

String& String::operator=(const String& str) {

if (this == &str) return \*this;

delete[] s\_data;

s\_size = strlen(str.s\_data);

s\_data = new char[s\_size + 1];

strcpy\_s(s\_data,s\_size+1, str.s\_data);

return \*this;

}

size\_t String::legnth() {

size\_t x;

x = strlen(s\_data);

return x;

}

String& String::cat(const String& str) {

char\* data;

data = new char[s\_size + 1];

strcpy\_s(data, s\_size + 1, s\_data);

delete[] s\_data;

s\_data = new char[strlen(data) + strlen(str.s\_data) + 1];

for (size\_t i = 0; i < strlen(data); i++) {

s\_data[i] = data[i];

}

for (size\_t i = strlen(data); i < strlen(data) + strlen(str.s\_data); i++) {

s\_data[i] = str.s\_data[i-strlen(data)];

}

s\_data[strlen(data) + strlen(str.s\_data) ] = '\0';

//sprintf(s\_data, "%s%s", data, str.s\_data);

/\*strcpy\_s(s\_data,strlen(data)+1,data);

strcat(s\_data,str.s\_data);\*/

return \*this;

}

void String::print() const {

cout <<s\_data<< endl;

}

源.cpp

#include<iostream>

#include<string>

#include"String.h"

using namespace std;

int main() {

system("color f0");

String s = "吉林大学";

cout<<s.legnth()<<endl;

String t = s;

cout << t.legnth() << endl;

String m;

m = t;

cout << m.legnth()<<endl;

cout << endl<<"测试字符串连接函数：" << endl;

m.cat(t);

cout << m.legnth()<<endl;

m.print();

return 0;

}

C：

继承方式：

头文件：

Class.h

#pragma once

#include<iostream>

using namespace std;

class Camera

{

private:

virtual void so\_co() = 0;

void co\_al() { cout << "使用Camera类功能 Coding algorithm 编码算法" << endl; }// Coding algorithm 编码算法

protected:

virtual void a\_shooting() = 0;

virtual void v\_s\_s() = 0;

void i\_q\_s() { cout << "使用Camera类功能 Image quality setting 图像质量设定" << endl; }// Image quality setting 图像质量设定

public:

virtual void o\_me() = 0;

virtual void c\_f() = 0;

void shooting() { cout << "使用Camera类功能 Shooting 摄像功能" << endl; };// 摄像功能

Camera() { cout << "Constructing Camera 构造摄像机类" << endl; };

~Camera() { cout << "Distructing Camera 析构摄像机类" << endl; };

virtual Camera\* data() { return 0; }

};

class Member {

public:

Camera\* m[4];

Member(){};

Member(Member& s) {

for (int i = 0; i < 4; i++) this->m[i] = s.m[i]->data();

}

Camera\* &f(int k) { return(m[k]); }

};

class Tachograph1:public Camera {

private:

void so\_co() { cout << "使用Tachograph类功能 Source Code 源代码" << endl; };// source code 源代码

protected:

void a\_shooting() { cout << "使用Tachograph类功能 Automatic shooting 自动拍摄" << endl; };//Automatic shooting 自动拍摄

void v\_s\_s() { cout << "使用Tachograph类功能 Vehicle speed sensor 车速传感器" << endl; };//Vehicle speed sensor 车速传感器

public:

void o\_me() { cout << "使用Tachographgraph类功能 Operation menu 操作菜单" << endl; };//Operation menu 操作菜单

Tachograph1() { cout << "Constructing Tachograph1 构造行车记录仪类" << endl; };

~Tachograph1() { cout << "Distructing Tachograph1 析构行车记录仪类" << endl; };

void c\_f();//Chip function 芯片功能测试 包括摄像功能和图像质量设定功能

Camera\* data() { return(new Tachograph1(\*this)); }

};

class Tachograph2 :protected Camera {

private:

void so\_co() { cout << "使用Tachograph类功能 Source Code 源代码" << endl; };// source code 源代码

protected:

void a\_shooting() { cout << "使用Tachograph类功能 Automatic shooting 自动拍摄" << endl; };//Automatic shooting 自动拍摄

void v\_s\_s() { cout << "使用Tachograph类功能 Vehicle speed sensor 车速传感器" << endl; };//Vehicle speed sensor 车速传感器

public:

void o\_me() { cout << "使用Tachographgraph类功能 Operation menu 操作菜单" << endl; };//Operation menu 操作菜单

Tachograph2() { cout << "Constructing Tachograph2 构造行车记录仪类" << endl; };

~Tachograph2() { cout << "Distructing Tachograph2 析构行车记录仪类" << endl; };

void c\_f();//Chip function 芯片功能测试 包括摄像功能和图像质量设定功能

};

class Tachograph3 :private Camera {

private:

void so\_co() { cout << "使用Tachograph类功能 Source Code 源代码" << endl; };// source code 源代码

protected:

void a\_shooting() { cout << "使用Tachograph类功能 Automatic shooting 自动拍摄" << endl; };//Automatic shooting 自动拍摄

void v\_s\_s() { cout << "使用Tachograph类功能 Vehicle speed sensor 车速传感器" << endl; };//Vehicle speed sensor 车速传感器

public:

void o\_me() { cout << "使用Tachographgraph类功能 Operation menu 操作菜单" << endl; };//Operation menu 操作菜单

Tachograph3() { cout << "Constructing Tachograph3 构造行车记录仪类" << endl; };

~Tachograph3() { cout << "Distructing Tachograph3 析构行车记录仪类" << endl; };

void c\_f();//Chip function 芯片功能测试 包括摄像功能和图像质量设定功能

};

源文件：

Class.cpp

#include "Class.h"

void Tachograph1 ::c\_f() {

this->i\_q\_s();

this->shooting();

}//Chip function 公有继承芯片功能测试 包括摄像功能和图像质量设定功能

void Tachograph2::c\_f() {

this->i\_q\_s();

this->shooting();

}//Chip function 保护继承芯片功能测试 包括摄像功能和图像质量设定功能

void Tachograph3::c\_f() {

this->i\_q\_s();

this->shooting();

}//Chip function 私有继承芯片功能测试 包括摄像功能和图像质量设定功能

源.cpp

#include<iostream>

#include<vector>

#include"Class.h"

using namespace std;

int main() {

system("color f0");

Member s;

s.f(0) = new Tachograph1;

cout << "测验public继承：" << endl;

s.f(0)->c\_f();//测试芯片摄像、图像质量设定功能

s.f(0)->o\_me(); s.f(0)->Camera::shooting();//测试用户操作行车记录仪的操作菜单和摄像机的摄像功能

return 0;

}

组合方式：

头文件：

Class.h

#pragma once

#include<iostream>

using namespace std;

class Camera

{

private:

void co\_al() { cout << "使用Camera类功能 Coding algorithm 编码算法" << endl; }// Coding algorithm 编码算法

protected:

void i\_q\_s() { cout << "使用Camera类功能 Image quality setting 图像质量设定" << endl; }// Image quality setting 图像质量设定

public:

void shooting() { cout << "使用Camera类功能 Shooting 摄像功能" << endl; };// 摄像功能

void i\_q\_s\_cout() { this->i\_q\_s(); };

Camera() { cout << "Constructing Camera 构造摄像机类" << endl; };

~Camera() { cout << "Distructing Camera 析构摄像机类" << endl; };

};

class Tachograph1 {

private:

void so\_co() { cout << "使用Tachograph类功能 Source Code 源代码" << endl; };// source code 源代码

protected:

void shooting() { cout << "使用Tachograph类功能 Automatic shooting 自动拍摄" << endl; };//Automatic shooting 自动拍摄

void v\_s\_s() { cout << "使用Tachograph类功能 Vehicle speed sensor 车速传感器" << endl; };//Vehicle speed sensor 车速传感器

public:

Camera c;

void o\_me() { cout << "使用Tachographgraph类功能 Operation menu 操作菜单" << endl; };//Operation menu 操作菜单

Tachograph1() { cout << "Constructing Tachograph1 构造行车记录仪类" << endl; };

~Tachograph1() { cout << "Distructing Tachograph1 析构行车记录仪类" << endl; };

void c\_f();//Chip function 芯片功能测试 包括摄像功能和图像质量设定功能

};

class Tachograph2 {

private:

void so\_co() { cout << "使用Tachograph类功能 Source Code 源代码" << endl; };// source code 源代码

protected:

Camera c;

void shooting() { cout << "使用Tachograph类功能 Automatic shooting 自动拍摄" << endl; };//Automatic shooting 自动拍摄

void v\_s\_s() { cout << "使用Tachograph类功能 Vehicle speed sensor 车速传感器" << endl; };//Vehicle speed sensor 车速传感器

public:

void o\_me() { cout << "使用Tachographgraph类功能 Operation menu 操作菜单" << endl; };//Operation menu 操作菜单

Tachograph2() { cout << "Constructing Tachograph2 构造行车记录仪类" << endl; };

~Tachograph2() { cout << "Distructing Tachograph2 析构行车记录仪类" << endl; };

void c\_f();//Chip function 芯片功能测试 包括摄像功能和图像质量设定功能

};

class Tachograph3 {

private:

Camera c;

void so\_co() { cout << "使用Tachograph类功能 Source Code 源代码" << endl; };// source code 源代码

protected:

void shooting() { cout << "使用Tachograph类功能 Automatic shooting 自动拍摄" << endl; };//Automatic shooting 自动拍摄

void v\_s\_s() { cout << "使用Tachograph类功能 Vehicle speed sensor 车速传感器" << endl; };//Vehicle speed sensor 车速传感器

public:

void o\_me() { cout << "使用Tachographgraph类功能 Operation menu 操作菜单" << endl; };//Operation menu 操作菜单

Tachograph3() { cout << "Constructing Tachograph3 构造行车记录仪类" << endl; };

~Tachograph3() { cout << "Distructing Tachograph3 析构行车记录仪类" << endl; };

void c\_f();//Chip function 芯片功能测试 包括摄像功能和图像质量设定功能

};

源文件：

Class.cpp

#include "Class.h"

void Tachograph1::c\_f() {

this->c.i\_q\_s\_cout();

this->c.shooting();

}//Chip function 公有继承芯片功能测试 包括摄像功能和图像质量设定功能

void Tachograph2::c\_f() {

this->c.i\_q\_s\_cout();

this->c.shooting();

}//Chip function 保护继承芯片功能测试 包括摄像功能和图像质量设定功能

void Tachograph3::c\_f() {

this->c.i\_q\_s\_cout();

this->c.shooting();

}//Chip function 私有继承芯片功能测试 包括摄像功能和图像质量设定功能

源.cpp

#include<iostream>

#include"Class.h"

using namespace std;

int main() {

system("color f0");

Tachograph1 t1;

Tachograph2 t2;

Tachograph3 t3;

cout << "测验public继承：" << endl;

t1.c\_f();//测试芯片摄像、图像质量设定功能

t1.o\_me(); t1.c.shooting();//测试用户操作行车记录仪的操作菜单和摄像机的摄像功能

cout << "测验protected继承：" << endl;

t2.c\_f();//测试芯片摄像、图像质量设定功能

t2.o\_me();//测试用户操作行车记录仪的操作菜单功能

cout << "测验private继承：" << endl;

t3.c\_f();//测试芯片摄像、图像质量设定功能

t3.o\_me();//测试用户操作行车记录仪的操作菜单功能

return 0;

}

D：

头文件：

Class.h

#pragma once

#include<iostream>

#include<string>

#include<time.h>

using namespace std;

class Date {

public:

int month=0;

int day=0;

};

class Employee

{

protected:

string name;

size\_t id=0;

Date birthDate;

public:

virtual void output() = 0;

virtual void change() = 0;

Employee() {};

virtual ~Employee() {};

void date\_cin(int month=1, int day=1) {

birthDate.month = month;

birthDate.day = day;

}

};

class SalariedEmployee :public Employee {

private:

double salary\_by\_week=3000;

double salary();

public:

void output();

SalariedEmployee(string a, size\_t b, int c, int d) {

name = a;

id = b;

birthDate.month = c;

birthDate.day = d;

}

SalariedEmployee() {

cout << "请分别输入SalariedEmployee类姓名、工号、出生月、出生日：" << endl;

cin >> this->name >> this->id >> this->birthDate.month >> this->birthDate.day;

cout << endl;

}

~SalariedEmployee(){};

void change();

};

class HourlyEmployee :public Employee {

private:

double salary\_by\_hour=80;

double salary(int workhour);

public:

void output();

HourlyEmployee(string a, size\_t b, int c, int d) {

name = a;

id = b;

birthDate.month = c;

birthDate.day = d;

}

HourlyEmployee() {

cout << "请分别输入HourlyEmployee类姓名、工号、出生月、出生日：" << endl;

cin >> this->name >> this->id >> this->birthDate.month >> this->birthDate.day;

cout << endl;

}

~HourlyEmployee() {};

void change();

};

class CommissionEmployee :public Employee {

private:

double salary\_by\_commission=50;

double salary(int sales);

public:

void output();

CommissionEmployee(string a, size\_t b, int c, int d) {

name = a;

id = b;

birthDate.month = c;

birthDate.day = d;

}

CommissionEmployee() {

cout << "请分别输入CommissionEmployee类姓名、工号、出生月、出生日：" << endl;

cin >> this->name >> this->id >> this->birthDate.month >> this->birthDate.day;

cout << endl;

}

~CommissionEmployee() {};

void change();

};

class BasePlusCommissionEmployee :public Employee {

private:

double salary\_by\_commission = 50;

double salary\_base = 2000;

double salary(int sales);

public:

void output();

BasePlusCommissionEmployee(string a, size\_t b, int c, int d) {

name = a;

id = b;

birthDate.month = c;

birthDate.day = d;

}

BasePlusCommissionEmployee() {

cout << "请分别输入BasePlusCommissionEmployee类姓名、工号、出生月、出生日：" << endl;

cin >> this->name >> this->id >> this->birthDate.month >> this->birthDate.day;

cout << endl;

}

~BasePlusCommissionEmployee() {};

void change();

};

源文件：

Class.cpp

#include "Class.h"

void SalariedEmployee::output() {

cout << "SalariedEmployee:" ;

cout << name << endl;

cout <<"工号为：" << id << endl;

cout <<"薪水为：" << salary() << endl;

}

double SalariedEmployee::salary()

{

double bonus=0;

time\_t nowtime = time(0);

struct tm local;

localtime\_s(&local, &nowtime);

if (birthDate.month == (local.tm\_mon + 1)) bonus = 100;

double s = salary\_by\_week+bonus;

return s; }

double HourlyEmployee::salary(int workhour)

{

double bonus = 0;

time\_t nowtime = time(0);

struct tm local;

localtime\_s(&local, &nowtime);

if (birthDate.month == (local.tm\_mon + 1)) bonus = 100;

double s=0;

if (workhour <= 40 && workhour > 0) s = workhour \* salary\_by\_hour + bonus;

else if (workhour > 40) s = 1.5 \* workhour \* salary\_by\_hour - 40 \* salary\_by\_hour;

return s;

}

void HourlyEmployee::output() {

cout << "请输入员工" << name << "本周工时：" << endl;

int workhour;

cin >> workhour;

cout << endl;

cout << "SalariedEmployee:";

cout << name << endl;

cout << "工号为：" << id << endl;

cout << "薪水为："<<salary(workhour)<<endl;

}

double CommissionEmployee::salary(int sales) {

double bonus = 0;

time\_t nowtime = time(0);

struct tm local;

localtime\_s(&local, &nowtime);

if (birthDate.month == (local.tm\_mon + 1)) bonus = 100;

double s=0;

s = sales\*(double)sales + bonus;

return s;

}

void CommissionEmployee::output() {

cout << "请输入员工"<<name<<"本周销售量：" << endl;

int sales;

cin >> sales;

cout << endl;

cout << "SalariedEmployee:";

cout << name << endl;

cout << "工号为：" << id << endl;

cout << "薪水为：" << salary(sales) << endl;

}

double BasePlusCommissionEmployee::salary(int sales) {

double bonus = 0;

time\_t nowtime = time(0);

struct tm local;

localtime\_s(&local, &nowtime);

if (birthDate.month == (local.tm\_mon + 1)) bonus = 100;

double s;

s = sales \* (double)sales + bonus+salary\_base;

s = s \* 1.1;

return s;

}

void BasePlusCommissionEmployee::output() {

cout << "请输入员工" << name << "本周销售量：" << endl;

int sales;

cin >> sales;

cout << endl;

cout << "SalariedEmployee:";

cout << name << endl;

cout << "工号为：" << id << endl;

cout << "薪水为：" << salary(sales) << endl;

}

void SalariedEmployee::change() {

cout << "请输入修改后的周薪：" << endl;

cin >> this->salary\_by\_week;

}

void HourlyEmployee::change() {

cout << "请输入修改后的时薪：" << endl;

cin >> this->salary\_by\_hour;

}

void CommissionEmployee::change() {

cout << "请输入修改后的销售佣金：" << endl;

cin >> this->salary\_by\_commission;

}

void BasePlusCommissionEmployee::change() {

cout << "请分别输入修改后的底薪和销售佣金：" << endl;

cin >> this->salary\_base >> this->salary\_by\_commission;

}

源.cpp

#include"Class.h"

#include<vector>

int main() {

system("color f0");

vector<Employee\*> m(10);

SalariedEmployee a;

HourlyEmployee b;

CommissionEmployee c;

BasePlusCommissionEmployee d;

m[0] = &a;

m[0]->output();

m[1] = &b;

m[1]->output();

m[2] = &c;

m[2]->output();

m[3] = &d;

m[3]->output();

return 0;

}

E：

头文件：

Class.h

#pragma once

#include<vector>

#include<iostream>

#include<math.h>

using namespace std;

class Shape {

public:

virtual double area() = 0;

virtual Shape\* data() = 0;

};

class Circle :public Shape {

public:

double area() { return 3.141 \* radius \* radius; }

Circle(double r = 1.0) :radius(r) {}

Circle(Circle const& c) { radius = c.Radius(); }

double Radius() const { return this->radius; }

void change() {

cout << "请输入新的半径：" << endl;

cin >> radius;

}

Shape\* data() { return new Circle(\*this); }

private:

double radius;

};

class Triangle :public Shape {

public:

double area() {

double s = 0.5\*(length[0] + length[1] + length[2]);

return sqrt(s \* (s - length[0]) \* (s - length[1]) \* (s - length[2]));

}

Triangle(vector<double> l) {

if (l[0] + l[1] <= l[2] || l[2] + l[1] <= l[0] || l[0] + l[2] <= l[1]) {

length = vector<double>(3, 1.0); cout << "构造失败，以默认值构造边长为1的等边三角形" << endl;

}

length = l;

}

Triangle(): length(vector<double>(3,1.0)){}

Triangle(Triangle const& t) {

length = t.length;

}

void change() {

cout << "请输入新的三角形三边长：" << endl;

cin >> length[0] >> length[1] >> length[2];

while (length[0]+length[1]<=length[2]|| length[2] + length[1] <= length[0] || length[0] + length[2] <= length[1] ) {

cout << "不符合三角形构图，请输入新的三角形三边长：" << endl;

cin >> length[0] >> length[1] >> length[2];

}

}

Shape\* data() { return new Triangle(\*this); }

private:

vector<double> length;

};

class Manage {

private:

Shape\* a[100];

public:

Manage() { for (int i = 0; i < 100; i++) this->a[i] = 0; };

Manage(Manage const& s) {

for (int i = 0; i < 100; i++) {

if (s.a[i] != 0)this->a[i] = s.a[i]->data();

else this->a[i] = 0;

}

};

Shape\* &f(int k) { return(a[k]); }

};

源文件：

Class.cpp

#include "Class.h"

源.cpp

#include"Class.h"

int main() {

system("color f0");

vector<double> f(3, 2.0);

Triangle a(f);

cout << a.area() << endl;

Triangle b(a);

cout << b.area() << endl;

Circle c(5);

cout << c.area() << endl;

Circle d(c);

cout << d.area() << endl;

cout << endl << "派生类深度拷贝测试完毕，下测试修改Circle类和Triangle类：" << endl;

c.change();

cout << c.area()<<endl;

a.change();

cout << a.area();

cout << endl << "改动所有的类功能测试完毕，下测试Triangle类的三边合理性检验：" << endl;

double m[3] = { 1,2,3 };

vector<double> g(m,m + 3);

Triangle n(g);

cout << endl << "下将上述所构造的Circle类对象和Triangle类对象归总到Manage类中，并测试Manage类深度拷贝：" << endl;

Manage p;

p.f(0) = &a;

p.f(1) = &b;

p.f(2) = &c;

p.f(3) = &d;

p.f(4) = &n;

cout << p.f(1)->area() << endl;

Manage q(p);

cout << q.f(1)->area() << endl;

return 0;

}