|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **姓名** | **王卓然** | **学号** | **20233005358** | **专业** | **计算机科学与技术** |

提示:交实验报告时，命名格式为：C++实验N-学号-姓名.docx， N为实验的序号。

实验一 类与对象（一）——类与对象的定义、构造函数、析构函数、对象的使用

一、实验目的

1．熟悉类的构成，掌握类的定义、对象的定义及对象成员的访问方法；

2．理解构造函数、析构函数的意义及作用，掌握构造函数、析构函数的定义及调用时间，熟悉构造函数的种类和构造函数的重载；

3．熟悉类与对象简单应用及编程。

二、实验学时

课内实验：2课时 课外练习：2课时

三 本实验涉及的新知识

（一） 类的定义

1．定义格式

**class 类名**

**{ [private :]**

**私有数据成员或成员函数**

**public :**

**公有数据成员或成员函数**

**protected**

**保护数据成员或成员函数**

**} ；**

2．说明：

⑴ 类的定义（或称类的声明）中，private、protected、public可以按任意顺序出现任意次。

⑵ 成员的访问限制

① 公有成员（public）：所定义的成员是开放的，可在任何地方访问（类内部及程序的其他地方）；

② 私有成员（private）：所定义的成员是隐藏的，只能在类内部访问，不能在程序的其他地方访问；

③ 保护成员（protected）：所定义的成员是半开放的，可在类内部访问，也可在其派生类中访问，但不能在程序的其他部分访问。

在类体外可以通过调用类的公有成员使用私有数据成员或保护数据成员，但不能改变其值。

⑶ 如果在类体重既不写关键字private又不写public，就默认为private。数据成员通常定义为私有成员，以实现数据的隐藏；成员函数通常定义为公有成员，以通过消息传递访问数据成员；保护成员主要用于继承。

3．数据成员

⑴ 定义格式：

**类型 数据成员名;**

⑵ 说明：

① 在定义类的数据成员时，由于计算机不给数据成员分配内存空间，不能使用auto、register、extern等进行说明，也不能赋初值；

② 数据成员既可为基本数据类型，也可以是复杂数据类型（数组、指针、引用、结构体变量、类的对象等）。

4．成员函数

在面向对象程序设计中，成员函数也称方法，是面向对象程序设计消息传递机制的实现方式。

⑴ 定义格式

① 在类中定义：

**返回类型 函数名(形参表)**

**{ 函数体}**

② 在类外定义：

在类中只给出成员函数的原型，成员函数体在类的外部定义。

类体内定义格式：

**返回类型 （函数名(形参表)；**

类体外定义格式：

**返回类型 类名::函数名(形参表)**

**{ 函数体}**

⑵ 说明：

① 通常，当成员函数代码量较大时，在类外定义成员函数；

② 在定义成员函数以前一定要加上类名，类名与函数名间一定要用“::”，以表示与普通函数的区别；

③ 成员函数中，应给出形参，当无形参时，通常应加上void；

④ 成员函数的返回值类型，与函数原型中的返回类型必须相同；

⑤ 成员函数与普通函数一样，可以设置参数的默认值；

⑥ 在类内部定义的成员函数为内置函数，如果要将类外部定义的成员函数设置为内联函数，则应在函数定义前加上inline。

（二） 对象的定义及成员的引用

类和对象是面向对象程序设计最重要的两个概念。从程序设计的角度理解，类是将具有相同数据成员、成员函数的操作对象的概括（或抽象），是定义对象的模板；而对象是类的实例化或具体化，即“类”变量。

1．对象的定义

⑴ 格式：

**类名 对象名表;**

⑵ 说明：在定义类时，计算机并不给类分配存储空间，只有定义了对象后，计算机才为对象分配存储空间。

2．对象成员的使用

⑴ 数据成员的使用

格式：

**对象名.数据成员名**

（只能是公有数据成员）。

⑵ 成员函数的使用

格式：

**对象名.成员函数名（实参表）**

**对象名.类名::成员函数名（实参表）**

⑶ 说明：

① 私有成员、公有成员不能在类的外部被使用，如果数据成员定义为私有成员、保护成员，则只能通过公有成员函数使用；

② 可以定义指向对象的指针用操作符“->”访问对象成员，也可以定义对象的引用，其引用名即为对象的别名。

（三） 构造函数与析构函数

在C++中，提供了两个特殊的成员函数，即构造函数和析构函数。

构造函数用于对象的初始化，即在定义一个类对象时，计算机在给对象分配相应的存储单元的同时，为对象的数据成员赋初值。

析构函数执行的是与构造函数相反的操作，用于撤销对象的同时释放对象所占用的内存空间。

1．构造函数

⑴ 构造函数的定义

格式：

**类名（形参表）**

**{ 构造函数体 }**

⑵ 构造函数的调用

构造函数的调用是在定义对象时调用的。

格式：

**类名 对象名（实参表）;**

**类名 对象名=构造函数名（实参表）;**

⑶ 说明

① 构造函数必须与类同名。

② 构造函数没有返回值，但不能在构造函数前加void类型符（其他没有返回值的成员函数必须加类型符void）。

③ 在实际应用中，在定义类时通常应定义一至多个构造函数(重载)，以对各数据成员进行初始化；如果不给出构造函数，系统将自定义一个构造函数。

④ 构造函数可以带参数，也可不带任何参数（称无参构选函数），还可以使用缺省参数；如果既要定义无参构选函数又要定义缺省参数构造函数，应避免初始化对象时出现歧义。

⑤ 构造函数不能象普通成员函数一样单独调用。

2．析构函数

⑴ 析构函数的定义

格式：

**~类名（void）**

**{ 析构函数体 }**

⑵ 析构函数的调用

析构函数是在撤销对象时自动调用的。

⑶ 说明

⑴ 析构函数与构造函数的名字相同，但在其前面加上“~”，如果未定义析构函数，系统将自定义一个析构函数。

⑵ 析构函数没有参数、没有返回值，也不能重载。

⑶ 对于大多数类而言，可以缺省析构函数的定义，但是，当类的数据成员中使用指针变量，在构造函数中用new动态分配内存空间时，应显式定义析构函数，用delete释放已分配的内存空间。

3．复制构造函数（拷贝构造函数）

⑴ 复制构造函数的定义

格式：

**类名（[const] 类名 &对象名）**

**{ 复制构造函数体 }**

⑵ 复制构造函数的调用

复制构造函数是将已知对象作为初值赋值给新定义的对象时自动调用的。

格式：

**类名 目标对象名=源对象名;**

**类名 目标对象名(源对象名);**

⑶ 说明

① 复制构造函数无返回值，也不能有void。

② 如果不定义复制构造函数，系统会自定义一个复制构造函数，默认复制构造函数是一种浅拷贝，以实现对数据成员的拷贝。

③ 复制构造函数在用已有对象复制一个新对象时被调用，在以下3中情况下需要复制对象：

a. 程序中需要新建立一个对象，并用另一个同类对象对它初试化时；

b. 当函数的参数时类的对象时；

c. 函数的返回值是类的对象。

四、实验内容

㈠ 验证及认知实验

按要求调试下列程序，并回答相关问题。

**程序1（exp\_101.cpp）**

#include <iostream>

#include <math.h>

using namespace std;

class complex

{ private:

double real;

double imag;

public:

void set\_complex(double r,double i)

{ real=r;imag=i;}

double get\_real( )

{return real;}

double get\_imag( )

{return imag;}

double get\_abs( )

{ double t;

t=sqrt(real\*real+imag\*imag);

return t;

}

};

int main()

{ complex A;

A.set\_complex(3.0,4.0);

cout <<"real of complex A=";

cout <<A.get\_real()<<endl;

cout <<"imag of complex A=";

cout <<A.get\_imag()<<endl;

cout <<"abs of complex A= ";

cout <<A.get\_abs()<<endl;

return 0;

}

问题：

1. 程序的输出结果为：

real of complex A=3

imag of complex A=4

abs of complex A= 5

1. 成员函数（方法）set\_complex(double r,double i)的作用是 将形参r，l分别赋值给实参real和image，作为一个复数的实部和虚部 ；

get\_real( ) 的作用是 获取复数的实部 ；get\_imag( ) 的作用是 获取复数的虚部 ；

get\_abs( ) 的作用是 求复数实部r和虚部i的平方和开根号（求一个复数的模） 。

1. 将main()函数中的语句行“ cout <<A.get\_real()<<endl ;”改为“cout <<A. real<<endl;”，重新编译程序，将出现 ‘double complex::real is private’报错 其原因是 real是私有的数据成员，不可在类外调用 。

**程序2（exp\_102.cpp）**

#include<iostream.h>

class A

{ private:

int a,b;

public:

A(void)

{ a=0;b=0;}

A(int x1,int x2)

{a=x1;b=x2;}

A(A &ob)

{ a=ob.a;b=ob.b;

cout<<"拷贝构造函数被调用！"<<endl;

}

void print(void)

{ cout<<"a="<<a<<" b="<<b<<endl;}

};

void main()

{ A ob1(20,30),ob;

A ob2(ob1);

ob2.print();

// ob=ob1;

// ob.print();

}

问题：

⑺ 运行该程序的输出结果为：

拷贝构造函数被调用！

a=20 b=30

⑻ 程序中的成员函数A(A &ob)称为 构造函数 ，该函数的执行时间是在执行 建立对象 被调用的。

⑼ 将main()中的“A ob2(ob1);”改为“A ob2=ob1;”，重新运行程序，观察输出结果，说明拷贝构造函数也可在 类外 时调用。

⑽ 将main()函数中加注释的语句去掉前面的“//”，重新运行程序，观测输出结果，说明执行“ob=ob1;”时 没有 调用拷贝构造函数，原因是“ob=ob1;”只是对象的 引用 。

㈡ 知识应用实验

1．按要求完善并调试通过下列程序

**程序3**

⑴ 完善“hdate.h”中类的定义：

class Date

{ private:

int year,month,day;

public:

void set\_date(int y=2000,int m=1,int d=1) //对数据成员赋值

{

year = y;

month = m;

day = d;

}

int get\_year() //返回year

{

return year;

}

int get\_month() //返回month

{

return month;

}

int get\_day() //返回day

{

return day;

}

bool isleapyear(void); //是闰年返回1,不是闰年返回0

void print\_date(void)

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

bool Date::isleapyear(void) //是闰年返回1,不是闰年返回0

{

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

}

⑵ 按注释要求完善下列程序**（exp\_103.cpp）**

#include<iostream>

using namespace std;

#include"hdate.h"

int main(void)

{ Date da1,da2;

int y,m,d;

da1.set\_date(2004,5,1);

da1.print\_date();

cout<<"year=";cin>>y;

cout<<"month=";cin>>m;

cout<<"day=";cin>>d;

da2.set\_date(y,m,d);//调用方法set\_date()，用消息y,m,d对da2的数据成员赋值

da2.print\_date(); //调用方法输出将da2用" 年 月 日"格式输出年月日

if(da2.isleapyear()){

cout<<"是闰年"<<endl;

}

else cout<<"不是闰年"<<endl; //调用方法输出da2的年是否为闰年

}

2．分析下列程序，写出程序的输出结果，再上机运行程序验证其正确性，如果不正确，请认真分析出错原因。

**程序4（exp\_104.cpp）**

#include<iostream.h>

class Myclass

{ private:

你分析的程序输出结果是：

构造函数被调用

main: a=10 b=10

调用func1：

拷贝构造函数被调用

func1: a=10 b=10

析构函数被调用

调用func2：

func2: a=10 b=10

调用func3：

func3: a=10 b=10

main: a=10 b=10

析构函数被调用

int a,b;

public:

Myclass(int x1=0,int x2=0)

{a=x1;b=x2;

cout<<"构造函数被调用！"<<endl;

}

~Myclass()

{cout<<"析构函数被调用！"<<endl;}

Myclass(Myclass &ob)

{ a=ob.a;b=ob.b;

cout<<"拷贝构造函数被调用！"<<endl;

}

void print(void)

{ cout<<"a="<<a<<" b="<<b<<endl;}

};

void func1(Myclass ob)

{ cout<<"func1: ";

ob.print();

}

程序的实际输出结果是：

构造函数被调用！

main: a=10 b=10

调用func1：

拷贝构造函数被调用！

func1: a=10 b=10

析构函数被调用！

调用func2：

func2: a=10 b=10

调用func3：

func3: a=10 b=10

main: a=10 b=10

析构函数被调用！

void func2(Myclass \*ob)

{ cout<<"func2: ";

ob->print();

}

void func3(Myclass &ob)

{ cout<<"func3: ";

ob.print();

}

void main()

{ Myclass ob(10,10);

cout<<"main: ";ob.print();

cout<<"调用func1："<<endl;

func1(ob);

cout<<"调用func2："<<endl;

func2(&ob);

cout<<"调用func3："<<endl;

func3(ob);

cout<<"main: ";ob.print();

}

㈢ 程序设计实验

说明：请将所设计的程序以给定的文件名存放在自己的文件夹下（后同）。

模拟一个裁判给比赛选手打分。

1．要求如下：

⑴ 裁判人数为UMPIRE；

⑵ 参赛选手为任意人；

⑶ 裁判给选手打分；

⑷ 去掉一个最高分，一个最低分，取其平均分为选手得分；

⑸ 按参赛选手的序号显示选手得分；

2．提示：

⑴ 定义一个类名为Result的记分类为选手记分，数据成员至少包括选手编号（用整型）、姓名（用字符数组）、裁判为选手的打分及得分（用实型数组）等，成员函数自定（必须有构造函数），将类的定义保存在“result.h”中；

⑵ 测试程序**（保存在exp\_105.cpp中）**采用交互方式：

① 提示输入参赛选手人数；

② 提示输入所有参赛选手的编号及姓名；

③ 显示“比赛开始，请为选手打分：”；

④ 显示：去掉的最高分、最低分，选手得分；

⑤ 比赛结束，按编号显示选手的得分。

//result.h

*//  
// Created by Devil\_Zero using JetBrains CLion on 2024/3/11.  
//*#include <iostream>  
#include <string>  
#define UMPIREMAX 10  
#ifndef EXP1\_RESULT\_H  
#define EXP1\_RESULT\_H  
using namespace std;  
class Result {  
public:  
 Result(int num = 0, string name = "NULL", int umpire = 5);  
 *//这里如果不给默认值,则在main函数中定义对象时必须给出参数，定义类数组必须在定义时给出参数* void setScores(int score[]);  
 int getMax();  
 int getMin();  
 double getAverage();  
  
private:  
 int i;  
 string n;  
 int s[UMPIREMAX];  
 double average;  
 int max;  
 int min;  
 int ump;  
};  
  
#endif *//EXP1\_RESULT\_H*

*//main.cpp(exp\_105.cpp)*

#include <iostream>

#include <string>

#include "Result.h"

#define UMPIREMAX 10

using namespace std;

Result::Result(int num, string name, int umpire) : i(num), n(name), ump(umpire), average(0), max(0), min(0) {

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

s[j] = 0;

}

}

void Result::setScores(int score[]) {

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

s[j] = score[j];

}

getMax();

getMin();

}//提前调用getMax和getMin函数,满足getAverage函数的前提

int Result::getMax() {

max = s[0];

for (int j = 1; j < ump; j++) {

if (s[j] > max) {

max = s[j];

}

}

return max;

}

int Result::getMin() {

min = s[0];

for (int j = 1; j < ump; j++) {

if (s[j] < min) {

min = s[j];

}

}

return min;

}

double Result::getAverage() {

double sum = 0;

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

sum += (double)s[j];

}

average = (sum - (double)max - (double)min) / double(ump - 2);

return average;

}

int main() {

int n;

cout << "Please input the number of students: ";

cin >> n;

Result student[10];

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

int num;

string name;

int score[UMPIREMAX];

cout << "Please input the number and name of student NO." << i + 1 << ": ";

cin >> num >> name;

for (int j = 0; j < 5; j++) {

cout << "Umpire NO." << j + 1 << "Please score the player:";

cin >> score[j];

}

student[i] = Result(num, name, 5);

student[i].setScores(score);

}

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

cout << "Average: " << student[i].getAverage() << endl;

}

return 0;

}

五、实验收获与创新

1．通过完成本实验，谈谈你对面向对象程序设计的初步认识。

面向对象程序设计（OOP）将软件分解为一系列通过消息进行交互的对象。每个对象代表现实世界中的某个实体或概念，能够提高代码的重用性、可扩展性和可维护性。通过模拟现实世界的结构和行为，OOP 使得开发复杂的系统变得更加直观和管理。通过学习C++语言中对OOP的实现，我感知这个理念对大型项目的开发和维护显著降低了难度，学到了核心思想

2．自已拟定一个解决实际题目，分析并抽象为一个类，然后编写类的测试程序（类的定义放在头文件中，测试程序用**exp\_106.cpp**保存），要求：

数据成员不少于三个，至少有一个数组成员或指针成员；

成员函数（方法）自定，但必须显式定义构造函数、析构函数、复制构造函数。

#include <iostream>  
#include <string>  
using namespace std;  
*//解决商品标价，数量，名称，总价的问题*class Products{  
public:  
 Products(int num = 0, string name = "NULL", double price = 0.0);  
 ~Products();  
 Products(const Products &p);  
 void setPrice(double price);  
 void setNum(int num);  
 void setName(string name);  
 double getPrice();  
 int getNum();  
 string getName();  
 double getTotalPrice();  
private:  
 int n;  
 string na;  
 double p;  
 double total;  
};  
Products::Products(int num, string name, double price) {  
 n = num;  
 na = name;  
 p = price;  
 total = 0;  
 cout<<"Constructor called"<<endl;  
}  
Products::~Products() {  
 cout << "Destructor called" << endl;  
}  
Products::Products(const Products &p) {  
 n = p.n;  
 na = p.na;  
 this->p = p.p;  
 total = p.total;  
 cout << "Copy constructor called" << endl;  
}  
void Products::setPrice(double price) {  
 p = price;  
}  
void Products::setNum(int num) {  
 n = num;  
}  
void Products::setName(string name) {  
 na = name;  
}  
double Products::getPrice() {  
 return p;  
}  
int Products::getNum() {  
 return n;  
}  
string Products::getName() {  
 return na;  
}  
double Products::getTotalPrice() {  
 return n \* p;  
}  
int main() {  
 Products PD[3];  
 for (int i = 0; i < 3; i++) {  
 int num;  
 string name;  
 double price;  
 cin >> num >> name >> price;  
 PD[i].setNum(num);  
 PD[i].setName(name);  
 PD[i].setPrice(price);  
 }  
 for (int i = 0; i < 3; i++) {  
 cout << PD[i].getName() << " " << PD[i].getNum() << " " << PD[i].getPrice() << " " << PD[i].getTotalPrice() << endl;  
 }  
 return 0;  
}