|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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。

实验二 类与对象（二）——常量成员、静态成员、友元、类模板

一、实验目的

1．理解常对象、常数据成员、常成员函数、常指针、常引用等常量对于数据保护的意义和作用，熟悉它们的相互关系及应用；

2. 理解静态数据成员、静态成员函数对于数据共享的意义和作用，熟悉其应用；

3. 掌握友元函数、友元类的定义及应用；

4. 掌握类模板的定义、实例化方法及应用；

5．进一步熟悉类与对象的应用及编程。

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

㈠ 常量成员函数、常量数据成员及常量对象

在C++中可以用const定义常对象（也称为常量对象）、常成员函数（也称为常量成员函数）、常数据成员（也称为常量数据成员）。

1．常成员函数的定义

⑴ 在类中定义：

**返回类型 成员函数名(参数表) const**

**{ 函数体 }**

⑵ 在类外定义

类中声明：

**返回类型 成员函数名(参数表) const ；**

类外定义：

**返回类型 类名::成员函数名(参数表) const**

**{ 函数体 }**

2．常数据成员的定义

定义格式：

**const 数据类型 成员名**

或：

**数据类型 const 成员名**

3．常对象的定义

定义格式：

**类名 const 对象名[(实参表)]；**

或：

**const 类名 对象名[(实参表)]；**

4．说明

⑴ 常对象只能在定义时初始化，常对象的数据成员都是常数据成员，其值不能改变。

⑵ 常对象只能调用常量成员函数，不能调用普通成员函数，常成员函数是常对象唯一的对外接口；而普通对象既可调用普通成员函数，也可以调用常成员函数。

⑶ 普通成员函数可以访问本类的常成员函数，而常成员函数不能访问本类的普通成员函数。

(4) 常数据成员必须在构造函数中通过初始化列表进行初始化，初始化后值不能改变。

(5) 常量数据成员及成员函数隶属于某一对象（与静态数据成员的差异）。

（二） 静态成员

静态成员分为静态数据成员及静态成员函数。

1．静态数据成员

⑴ 静态数据成员的定义（类中定义）

格式：

**static 类型 数据成员名表；**

⑵ 静态数据成员的作用

静态数据成员在内存中只占一份空间（而不是每个对象都分别为它保留一份空间），属于一个类（或者说属于类的第一个对象），所有对象都共享静态成员。

静态数据成员是连接各个对象的桥梁，可用来取代全局变量，主要用于各个对象都共享的公用数据，如：总数的统计、平均数等。

⑶ 说明

① 静态数据成员可以说明为公有成员、私有成员或保护成员，但一般不说明为公有成员；

② 静态数据成员同其它数据成员一样，不能在类中进行初始化（不分配内存空间）；

③ 静态数据成员由于属于类（不依赖任何对象），应在程序开始运行时即生成，必须在任何函数之外进行初始化操作；

格式：

**类型 类名::静态数据成员=初值；**

④ 使用静态数据成员取代全局变量，使数据更为安全；

⑤ 当在类中需要共享某一数据成员时通常定义为静态数据成员；

⑥ 静态数据成员与静态对象是两个完全不同的概念，前者（静态数据成员）是在类中定义的，是类的所有对象共享的数据成员，而后者（静态对象）是指对象存储在静态存储区。

2．静态成员函数

⑴ 静态成员函数的定义

静态成员函数是在定义类的成员函数时，在成员函数名前加上static。

⑵ 静态成员函数的调用方法

① 用类名调用

格式：

**类名::静态成员函数名（实参表）；**

② 用对象调用

格式：

**对象名.静态成员函数名（实参表）；**

⑶ 说明

① 静态成员函数可以在类内部定义（必须加上static），也可在类外定义，即在类体内声明，在类体外定义，定义时与普通成员函数相同；

② 静态成员函数无this指针（this指针属于某一对象，静态成员属于一个类），只能访问静态数据成员，不能访问普通数据成员，因此静态成员函数专门用来访问静态数据成员；

③ 当在类中定义有静态数据成员时，通常应定义静态成员函数来访问静态数据成员。

(三) 友元

1．友元的概念

在C++中提供了一种类以外的函数访问类的私有成员、保护成员的方法，即将这些函数定义为类的友元。

2．友元的种类及应用

⑴ 友元函数

定义方法：在类中进行声明时，在声明的前面加上friend。

说明：

① 友元函数不是类的成员函数，除非确因成员函数使用频繁，需要提高程序的执行效率而定义为友元函数外，一般不要定义。

② 友元函数可以访问对象的私有成员和保护成员，只是在类中声明时加上friend，但它不是成员函数，定义时不能在函数名前加上“类名::”。

③ 友元函数无this指针，一般至少应带有一个入口参数，通过入口参数传递的对象名来引用该对象的成员。

④ 友元函数主要用于运算符的重载。

⑵ 友元成员

友元成员是指一个类的成员函数可以是另一个类的友元函数，这样就可以通过一个类的成员函数访问另一个类的私有成员或保护成员。

说明：

① 一个类的成员函数要作为另一个类的友元函数时，必须先定义这个类。

② 由于友元函数所在的类名先于类的定义出现，应提前声明。

⑶ 友元类

友元类是指一个类可以作为另一个类的友元，这样友元类的所有成员函数均可以访问另一个类中的成员。

1. 类模板

1. 类模板的概念

如果有多个类，其数据成员、成员函数相同，而只是某些数据成员的类型不同，可用类模板的方法处理。

1. 类模板的定义

(1)类模板的定义格式

**template<class T1,class T2,… >**

**class 类模板名**

**{ 类体 }**

(2)类模板中的成员函数的定义

① 在类体中定义（即定义为内置函数）

与普通类的成员函数定义相同。

② 在类体外定义格式

**template<class T1,class T2,… >**

**返回类型 类模板名<模板形参表>::成员函数名（形参表）**

**{ 函数体 }**

1. 类模板的实例化

类模板是在定义对象时实例化的。格式为：

**类模板名 <类模板实参表> 对象名表[(实参表)]；**

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\_201.cpp）

#include<iostream.h>

class counter

{ private:

static int count; //count为静态数据成员

char ch; //普通数据成员

public:

counter(char c) //构造函数

{ this->count ++;ch=c;}

void print\_counter(void)

{cout<<ch<<": "<<count<<endl;}

~counter(void)

{count--;}

};

int counter::count=100; //静态数据成员赋初值

void main(void)

{ counter c1('A');

c1.print\_counter();

cout<<endl;

counter c2('B');

c1.print\_counter();

c2.print\_counter();

cout<<endl;

counter c3('C');

c1.print\_counter();

c2.print\_counter();

c3.print\_counter();

}

问题：

⑴ 运行该程序的输出结果为

A: 101

A: 102

B: 102

A: 103

B: 103

C: 103

⑵ 由输出结果可知，静态成员属于 整个类，或者所有该类对象 ，在类中只有 一份 拷贝。

⑶ 程序中的“int counter::count=100;”的作用是 给静态成员count赋初值100 ，如果将其放在main()函数中，重新编译程序，会出现 编译错误 ，其原因是静态成员属于全局变量，不可在任何函数和局部作用域中初始化 。

程序2（exp\_202.cpp）

#include<iostream.h>

#include<string.h>

class girl

{ private:

char \*name;

int age;

public:

girl(char \*na,int n)

{ name=new char[strlen(na)+1];

strcpy(name,na);

age=n;

}

~girl(void)

{delete name;} //释放new分配的内存

char \*get\_name(void)

{return name;}

int get\_age(void)

{return age;}

void print\_girl(void);

};

void girl::print\_girl(void)

{ cout<<"姓名:"<<name;

cout<<" 年龄:"<<age<<endl;

}

void disp(girl &g)

{ cout<<"姓名:"<<g.name;

cout<<" 年龄:"<<g.age<<endl;

}

void main(void)

{ girl g1("李小丫",12),g2("王永兰",15),g3("赵梦美",13);

cout<<"姓名:"<<g1.get\_name();

cout<<" 年龄:"<<g1.get\_age()<<endl;

g2.print\_girl();

disp(g3);

}

问题：

(4)编译程序出错的原因是 name是私有的，不可直接引用g对象中的私有变量g 。

(5)在girl类中加上“friend void disp(girl &g);”重新编译、运行程序，其输出结果为：

姓名:李小丫 年龄:12

姓名:王永兰 年龄:15

姓名:赵梦美 年龄:13

其中，“friend void disp(girl &g);”表示函数disp( )为girl类的 友元函数 。

程序3（exp\_203.cpp）

#include<iostream.h>

template <class A>

class point

{ private:

A x,y;

public:

point(void)

{x=0;y=0;}

point(A a,A b)

{x=a ; y=b; }

point(const point &ob)

{x=ob.x;y=ob.y;}

void set\_point(A a,A b);

A read\_x(void);

A read\_y(void);

void move(A a,A b);

void print(void);

};

template <class A>

void point <A>::set\_point(A a,A b)

{x=a ; y=b; }

template <class A>

A point <A>::read\_x(void)

{ return x;}

template <class A>

A point<A>::read\_y(void)

{ return y;}

template <class A>

void point<A>::move (A a,A b)

{ x+=a;y+=b;}

template <class A>

void point<A>::print(void)

{ cout<<"x="<<x<<" y="<<y<<endl;}

void main()

{ }

问题：

(6) 在main()中加入下列语句：

point <int> p1(20,30);

p1.print();

p1.move(5,5); p1.print();

运行程序的输出结果为：

x=20 y=30

x=25 y=35

(7)程序中的“point <int> p1(20,30);”的作用是 用int替换函数模板A，创建对象p2并赋值a=20,b=30 ；

(8) 在main()中加入下列语句：

point <double> p2(20.55,30.65);

p2.print();

p2.move(5.5,10.5); p2.print();

运行程序的输出结果为：

x=20.55 y=30.65

x=26.05 y=41.15

(9) 程序中的“point <double> p2(20.55,30.65);”的作用是 用double替换函数模板A，创建对象p2并赋值a=20.55,b=30.65 。

㈡ 知识应用实验

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\_204.cpp）

#include<iostream.h>

#include<string.h>

你分析的程序输出结果是：

class boy;//类boy的声明;

class girl

{ private:

char \*name;

int age;

程序的实际输出结果是：

女孩姓名:李小丫 年龄:12

男孩姓名:张海兵 年龄:15

public:

girl(char \*na,int n)

{ name=new char[strlen(na)+1];

strcpy(name,na);

age=n;

}

~girl(void)

{delete name;} //释放new分配的内存

friend void disp(girl &g,boy &b);//声明disp为girl类友元函数

};

class boy

{ private:

char \*name;

int age;

public:

boy(char \*na,int n)

{ name=new char[strlen(na)+1];

strcpy(name,na);

age=n;

}

~boy(void)

{delete name;} //释放new分配的内存

friend void disp(girl &g,boy &b);//声明disp为boy类友元函数

};

void disp(girl &g,boy &b)

//定义友元函数：不是类的成员不需要作用域运算符

{ cout<<"女孩姓名:"<<g.name;

cout<<" 年龄:"<<g.age<<endl;

cout<<"男孩姓名:"<<b.name;

cout<<" 年龄:"<<b.age<<endl;

}

void main(void)

{ girl g1("李小丫",12);boy b1("张海兵",15);

disp(g1,b1);//调用友元函数：与普通函数调用相同

}

**程序5**（exp\_205.cpp）

#include <iostream.h>

const int size=50;

template <class T>

class array

运行程序，输入n的值为5，再分别输入：21、25、18、32、59时，你分析的程序输出结果为：

18 21 25 32 39

{ T x[size];

public:

array()

{int i;

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

x[i]=0;

}

void input(int n)

程序的实际输出结果是：

18 21 25 32 59

{int i;

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

{cout<<"x["<<i<<"]=";

cin>>x[i];

}

}

void print(int n)

{int i;

程序的实际输出结果是：

12.8 17.8 18.2 19.5 23.2

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

cout <<x[i]<<" ";

当输入：18.2、12.8、19.5、23.2、17.8时，你分析的程序输出结果为：

12.8 17.8 18.2 19.5 23.2

cout<<endl;

}

T max(int n);

void sort(int n);

};

template <class T>

T array<T>:: max(int n)

{ T m;

m=x[0];

int i;

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

if (m<x[i]) m=x[i];

return m;

}

template <class T>

void array<T>:: sort(int n)

{ int i,j;

T t;

for(i=0;i<n-1;i++)

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

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

{t=x[i];x[i]=x[j];x[j]=t;}

}

void main()

{ int n;

array <int> ob1;

cout<<"n=";cin>>n;

cout<<"integer:\n";

ob1.input(n);ob1.print(n);

cout<<"max="<<ob1.max(n)<<endl;

ob1.sort(n);

cout<<"sort:";ob1.print(n);

cout<<"\ndouble:\n";

array <double> ob2;

ob2.input(n);ob2.print(n);

cout<<"max="<<ob2.max(n)<<endl;

ob2.sort(n);

cout<<"sort:";ob2.print(n);

}

㈢ 程序设计实验

编写程序，定义一个平面点“Point”类及计算两点之间距离“Distance”类，将“Distance”类定义为“Point”类的友元类；并在main()中计算任意两点的距离（保存在exp\_206.cpp中）。

exp\_206.cpp

#include <iostream>  
#include <cmath>  
using namespace std;  
  
class Point {  
 friend class Date; *// Date类被声明为Point类的友元*private:  
 int ax, ay, bx, by;  
  
public:  
 Point(int x1 = 0, int y1 = 0, int x2 = 0, int y2 = 0) : ax(x1), ay(y1), bx(x2), by(y2){}  
};  
class Date {  
public:  
 double getDistance(Point &p) { *//对形参对象p的引用* return sqrt((p.bx - p.ax) \* (p.bx - p.ax) + (p.by - p.ay) \* (p.by - p.ay));  
 }  
};  
  
int main() {  
 int x1, y1, x2, y2;  
 cin >> x1 >> y1 >> x2 >> y2;  
 Point point(x1, y1, x2, y2); *//构造对象point* Date date;  
 cout << date.getDistance(point) << endl; *// 使用Date类的方法来获取距离* return 0;  
}

五、实验收获与创新

1．通过本实验，谈谈你对“数据的共享与保护”的认识。

通过面向对象的程序设计，利用其通过封装将数据和相关行为组合在一起，形成对象。这种封装性质使得数据在对象内部受到保护，外部只能通过对象提供的接口来访问和修改数据，从而实现了数据的保护。通过访问修饰符（如private、protected、public）控制数据的访问权限。将数据声明为私有（private）可以限制对数据的直接访问，只允许通过对象的公共接口来访问，这样可以提高数据的安全性和保护隐私。虽然面向对象程序设计通过封装保护了数据，但仍然可以通过对象的公共接口来共享数据。对象可以暴露一些公共方法或属性，允许其他对象或模块访问和使用数据，从而实现数据的分享。

2．自已拟定一个解决实际问题的题目，综合应用“实验一”至“实验二”的知识，编写程序并调试通过。

#include <iostream>  
using namespace std;  
class Values{  
 friend class Bulk;  
private:  
 int length;  
 int width;  
 static int height;  
public:  
 Values(int len = 0, int wid = 0): length(len),width(wid){} *//构造函数*};  
int Values::height = 10;  
class Bulk{  
public:  
 int getBulk(Values &p){  
 return p.length \* p.width \* p.height; *//求长方体的体积* }  
};  
int main(){  
 int a,b;  
 cin>>a>>b;  
 Values v(a,b); *//创建Values类对象v并初始化* Bulk bulk; *//创建Bulk类对象bulk* cout << bulk.getBulk(v) << endl;  
 return 0;  
}