***实验六 虚函数 参考答案***

**1.阅读并执行代码,然后回答问题**

***class A***

{

public:

A(){ cout<<"Lines="<<++lineno<<" Call A::A()"<<endl;}

virtual ~A() { cout<<"Lines="<<++lineno<<" Call A::~A()"<<endl;} //---------------(1)

virtual int Func1() const=0; //----------------------------------------------(2)

virtual void Func2(**int=500**)=0; //----------------------------------------------(3)

protected: //-------------------------------------------- ----------------(4)

**static** int lineno;

};

**class B:public A**

{

public:

B() {cout<<"Lines="<<++lineno<<" Call B::B()"<<endl;}

virtual ~B() {cout<<"Lines="<<++lineno<<" Call B::~B()"<<endl;} //----------------(5)

**virtual int Func1(int n)** const {return num+n;}

virtual int Func1() const //--------------------------------------------(6)

{

cout<<"Lines="<<++lineno<<" Call B::Func1() const"<<endl;

return num;

}

virtual void Func2(**int n=1000**) //--------------------------------------------(7)

{

cout<<"Lines="<<++lineno<<" Call B::Func2(int) n="<<n<<endl;

num=n;

}

void Func3(int n) {num=n;} //---------------------------------------------(8)

protected:

int num;

};

int A::lineno=0; //----------------------------------------------(9)

int **main**(int argc, char\* argv[])

{

A \* p=new B; //---------------------------------------------(10)

p->Func2(50); //---------------------------------------------(11)

p->Func2(); //---------------------------------------------(12)

p->Func1(); //---------------------------------------------(13)

**//p->Func3(100); //---------------------------------------------(14)**

**//p->Func1(100);**  //---------------------------------------------(15)

delete p;

return 0;

}

***原始程序输出：***

Lines=1 Call A::A()

Lines=2 Call B::B()

Lines=3 Call B::Func2(int) n=50

**Lines=4 Call B::Func2(int) n=500**

Lines=5 Call B::Func1() const

Lines=6 Call B::~B()

Lines=7 Call A::~A()

***1)去掉(1)中的virtual,比较执行结果.***

Lines=1 Call A::A()

Lines=2 Call B::B()

Lines=3 Call B::Func2(int) n=50

Lines=4 Call B::Func2(int) n=500

Lines=5 Call B::Func1() const

***Lines=6 Call A::~A()***

**B对象部分未析构**

***2)去掉(6)中的const,可以吗?***

**不可以，A类virtual int Func1() const=0;纯虚未全实现，仍是抽象类，不能实例化**

***3)(3)和(7)中各自定义了参数的缺省值,(12)执行时,匹配的是哪个缺省值,为什么?由结果能得到什么结论？***

**匹配基类缺省参数值，因为函数参数缺省值在编译时匹配，而不是在运行时。**

**虚函数参数的缺省值在编译时匹配，而非在运行时，派生类虚函数的缺省参数值不起作用，可以省略**

***4)(4)中的protected改为private可以吗?***

**不可以，派生类不可访问基类private类型成员**

***5)去掉(5)中的virtual,对结果有影响吗?***

**没有影响，基类是虚函数，派生类自动是虚的**

***6)(9)的作用是什么?***

**类变量的初始化**

***7)(10)中改为A\* p=new A;可以吗?理解抽象类和具体类的概念.***

不可以，抽象类不能实例化

***8)去掉(3)中的virtual,结果会有什么改变?***

**格式错误，=0必须是虚函数；**

**去掉=0并给出实现后，输出如下：**

Lines=1 Call A::A()

Lines=2 Call B::B()

**Call A::Func2（int=500)**

**Call A::Func2（int=500)**

Lines=3 Call B::Func1() const

Lines=4 Call B::~B()

Lines=5 Call A::~A()

***9)(14)为什么不对?将(8)改为virtual后,(14)可以了吗?***

**静态类型无成员函数Func3**

**（8）改为virtual后，（14）也不可以，静态类型仍无Func3**

***10)(15)为什么不对?理解编译时,使用静态类型;运行时,使用动态类型的含义.***

**静态类型无成员Func1(int n) const.**

**2.一个游戏中有多种怪物(Monster)，怪物之间可能要发生战斗(fight)，每场战斗都是一个怪物与另一怪物之间的一对一战斗。每个怪物都有自己的生命值(hitpoint)、攻击力值(damage)和防御力值(defense)，每种怪物都有各自特有的攻击(attack)方式，产生相应的攻击效果；战斗时，两个怪物依次攻击对方，即怪物a首先攻击怪物b, 然后轮到怪物b攻击怪物a, 之后，怪物a再次攻击怪物b，…, 直到一方生命值为0。**

**请根据你对上述描述的理解，定义并实现怪物类Monster，成员的设计可以任意，但要求该类至少有一个不带virtual修饰的成员函数fight，用来描述与另外一个怪物进行战斗的过程，该函数的实现可为Monster类的任意派生类所复用（派生类不需重新定义及实现）。不必考虑怪物的生命值减少至0后如何处理。**

***class Monster***

{

public:

Monster(int hp,int dam,int def):hitpoint(hp),damage(dam),defense(def)

{

}

virtual ~Monster( )

{

}

bool **Fight**(Monster & other)

{

while(1)

{ **//注意如果自己未胜，而且对方也未取胜，则会继续战斗**

if (**Attack**(other)) return true;

if (other.**Attack**(\*this)) return false;

}

}

int GetHitpoint() const { return hitpoint; }

int GetDamage() const { return damage; }

int GetDefense() const { return defense; }

void  ***ReduceHP***(int harm)

{

if (harm > 0)

{

hitpoint = hitpoint>harm ? hitpoint-harm : 0;

}

}

private:

**virtual** bool **Attack**(Monster& other )**=0**;

private:

int hitpoint;

int damage;

int defense;

};

**3．作为怪物的特例，猫和狗的攻击效果如下表所示。在Monster的基础上，以继承手段定义并实现这两个类。**

**猫进攻导致对方的生命值减少量：**

**（猫的攻击力值 \* 2 — 对方的防御力值） 若上式小于1，则取1**

**狗进攻导致对方的生命值减少量：**

**(狗的攻击力值 — 对方的防御力值 ＋5 )\*2 若上式小于2，则取2**

***class Cat***:**public Monster**

{

public:

Cat(int hp,int dam,int def):Monster(hp,dam,def)

{

}

private:

**virtual** bool **Attack**(Monster& other)

{

///猫的攻击力值 \* 2—对方的防御力值）

int harm = GetDamage()\*2 - other.GetDefense();

if (harm<1) harm = 1;

other.ReduceHP(harm);

return (other.GetHitpoint()<=0);**//对方小于等于0，则自己胜true，否则不胜false**

}

};

***class Dog:*public Monster**

{

public:

Dog(int hp,int dam,int def):Monster(hp,dam,def)

{

}

private:

**virtual** bool **Attack**(Monster& other)

{

///(狗的攻击力值—对方的防御力值 ＋5 )\*2 若上式小于2，则取2

int harm = (GetDamage() - other.GetDefense()+5)\*2;

if (harm<2) harm = 2;

other.ReduceHP(harm);

return (other.GetHitpoint()<=0);**//同上**

}

};

int main( )

{

Cat cat(36,10,10);

Dog dog(36,12,8);

**//Cat cat(37,10,10);**

**//Dog dog(37,12,8);**

if (cat.Fight(dog))

cout<<"Cat is WINNER."<<endl;

else

cout<<"Dog is WINNER."<<endl;

cout<<"Current Each Hitpoints are: "<<endl;

cout<<"Cat's Hitpoint ="<<cat.GetHitpoint()<<endl;

cout<<"Dog's Hitpoint ="<<dog.GetHitpoint()<<endl;

return 0;

}

**4.给出适当的类设计和相应的代码.**

**有一个只能放进不能取出的盒子,最多可放8个水果,不一定同一天放入.水果只是苹果和桔子两种,它们放入盒子前的原始重量分别为50和30,放入盒子后,由于丢失水分,它们的重量减轻,苹果和桔子每天分别减轻4和3,直到达到各自原始重量的3/5后,不再减轻重量.盒子的功能有:输出盒子中苹果的数量;输出盒子中桔子的数量;输出*一天来*盒子中水果减轻的总重量;输出当前水果的总重量;（使用虚函数和向下类型转换）。**

#include <iostream>

using namespace std;

***class Fruit***

{

public:

Fruit(int w): origin(w), weight(w) {}

**virtual ~**Fruit() {}

**virtual** int **reduceWater()** **= 0**;

int getOrigin() const { return origin;}

int getWeight() const { return weight;}

void setWeight(int w) { weight = w; }

private:

**const** int origin; //original weight

int weight; //current weight

};

***class Apple:*** public **Fruit**

{

public:

Apple(int w): Fruit(w) {}

**virtual int reduceWater()//返回值为本次（每天）实际减少的重量**

{

// an apple reduce 4 each day

int minWeight = getOrigin() \* 3 / 5;

int oldWeight = getWeight();

int newWeight = (oldWeight - 4 < minWeight ? minWeight : oldWeight - 4);

setWeight(newWeight);

return oldWeight - newWeight;

}

};

***class Orange***: **public Fruit**

{

public:

Orange(int w): Fruit(w) {}

**virtual int reduceWater()//同上**

{

// an orange reduce 3 each day

int minWeight = getOrigin() \* 3 / 5;

int oldWeight = getWeight();

int newWeight = (oldWeight - 3 < minWeight ? minWeight : oldWeight - 3);

setWeight(newWeight);

return oldWeight - newWeight;

}

};

***class Box***

{

public:

Box();

void **addFruit(Fruit & one);**

int passOneDay(); **// return the total reduced weight**

int appleCount() const;

int orangeCount() const;

int totalWeight() const;

public:

void show( int day);

private:

**Fruit \* fruits[8];**

};

Box::Box()

{

for(int i = 0; i < 8; ++i)

fruits[i] = nullptr;

}

void Box::addFruit(Fruit & one)

{

for(int i = 0; i < 8; ++i)

{

if(fruits[i] != nullptr) { continue; }

fruits[i] = &one;

return;

}

}

int Box::appleCount() const

{

int result = 0;

for(int i = 0; i < 8; ++i)

{

***Apple \*p = dynamic\_cast<Apple \*> (fruits[i]);***

if( p )

++result;

}

return result;

}

int Box::orangeCount() const

{

int result = 0;

for(int i = 0; i < 8; ++i)

{

Orange \*p = dynamic\_cast<Orange \*> (fruits[i]);

if( p ) ++result;

}

return result;

}

int Box::totalWeight() const

{

int total = 0;

for(int i = 0; i < 8; ++i) {

if (fruits[i])

total += fruits[i]->getWeight();

}

return total;

}

**int Box::passOneDay()**

{

int total = 0;

for(int i = 0; i < 8; ++i)

{

if (fruits[i])

total += **fruits[i]->reduceWater();**

}

return total;

}

void Box::show( int day)**//这里*也可改为*按day天的值减少而不是只减少一天的值**

{

cout<<"Day = "<<day<<endl;

cout<<"Apple Count = "<<appleCount()<<endl;

cout<<"Orange Count = "<<orangeCount()<<endl;

cout<<"Weigth Before Reduce="<<totalWeight()<<endl;

**cout<<"Total Reduce = "<<passOneDay()<<endl;//或可循环day次**

cout<<"Weigth After Reduce="<<totalWeight()<<endl<<endl;

}

int main()

{

//create 4 apples

Apple apples[] = { Apple(50), Apple(50), Apple(50), Apple(50) };

//create 3 oranges

Orange oranges[] = { Orange(30), Orange(30), Orange(30) };

//Create a Box

Box box;

// the 1st day

box.addFruit(apples[0]);

box.addFruit(oranges[0]);

box.show(1);

// the 2nd day

box.addFruit(apples[1]);

box.addFruit(oranges[1]);

box.show(2);

// the 3rd day

box.addFruit(apples[2]);

box.addFruit(oranges[2]);

box.show(3);

// the 4th day

box.show(4);

// the 5th day

box.show(5);

// the 6th day

box.show(6);

// the 7th day

box.show(7);

return 0;

}

***5.***

#include <iostream.h>

**class CallBackObject;**

***class Server***

{

public:

Server(int size) :len(size)

{

**-----(1)-----;*data=new int[len];***

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

data[i]= i+1;

}

~Server( ) { **----(2)-----;** }***delete []data;***

int Total(CallBackObject& obj);

private:

int len;

int \* data;

};

***class CallBackObject***

{

public:

virtual ~CallBackObject( ) {}

**----(4)-----;*virtual int CallBackFunc(int val)=0;***

};

***class ClientA***:public **CallBackObject**

{

public:

virtual ~ClientA( ) {}

virtual int CallBackFunc(int val)

{

return val;

}

void RequestA(Server& srv)

{

cout<< **-----(5)-----** <<endl;***srv.Total(\*this)***

}

};

int Server::Total(CallBackObject& obj)

{

int sum = 0;

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

{

**-----(3)-----;sum+=** **obj.CallBackFunc(data[i]);**

}

return sum;

}

***class ClientB:***public **CallBackObject**

{

public:

virtual ~ClientB( ) {}

virtual int CallBackFunc(int val)

{

**-----(6)-----;return val\*val;**

}

void RequestB(Server& srv)

{

**-----(7)-----; cout<<“平方和=”<<srv.Total(\*this)<<endl;**

}

};

//主函数1

void main( )

{

Server srv2(2),srv5(5);

ClientA a;

a.RequestA(srv2); //输出3

a.RequestA(srv5); //输出15

}

//主函数2

void main( )

{

Server srv2(2), srv3(3);

ClientB b;

b.RequestB(srv2);

b.RequestB(srv3);

}

1)请分别给出空格1-5中正确的代码，使得主函数1的输出为3和15.

2)请分别给出空格6-7中正确的代码，使得主函数2的输出为：

平方和=5

平方和=14

3)请重新实现ClientB中的相关函数，使得主函数2的输出为：

1 2 的立方和=9

1 2 3 的立方和=36

答案：

(一)

1)data = new int[len]

2)delete[] data

3)sum += obj.CallBackFunc(data[i])

4)virtual int CallBackFunc(int val) = 0

5)srv.Total(\*this)

(二)

6)return val\*val;

7)cout<<”平方和=”<<srv.Total(\*this)<<endl;

(三)

virtual int CallBackFunc(int val)

{

cout<<val<<” ”;

return val\*val;

}

void RequestB(Server& srv)

{

int n = srv.Total(\*this);

cout<<”的立方和=”<<n<<endl;

}