

1. 已知f1和f2是同一类中的两个成员函数,若f1的实现代码体内不能调用f2,则可能的情况是:

- (A) f1和f2都是静态函数 (B) f1是静态的, f2不是  
(C) f1不是静态的, f2是静态的 (D) f1和f2都不是静态函数

2. 一个对象所占的内存空间中可以保存下面哪类数据?

- (A) 静态数据成员 (B) 内联函数代码  
(C) 所有成员函数的入口地址 (D) 虚函数表的入口地址

3. 下面关于new和delete操作符的说法,哪个是不正确的:

- (A) 使用new操作符,可以动态分配全局堆中的内存资源。  
(B) 用new申请的数组,必须用delete[]释放。  
(C) 用new申请的空间,即使不调用delete释放掉,当程序结束时也会自动释放掉。  
(D) 执行语句A \* p=new A[100];时,类A的构造函数会被调用 100 次。

4. 下列哪种函数可以是虚函数:

- (A) 自定义的构造函数 (B) 拷贝构造函数 (C) 静态成员函数 (D) 析构函数

5. C++的最小编译单位是:

- (A) 工程中每个.cpp和.h文件 (B) 工程中每个.cpp文件  
(C) 工程中每个.h文件 (D) 工程中所有文件

6. 下面表达式中不可能作为左值(赋值运算符左侧)的是:

- (A) a=b (B) \*p (C) f(a,b) (D) &a

7. 在实现函数 A&f(A&obj)时,下面的哪一个表达式可以出现在return语句中?

- (A) new A() (B) obj (C) &obj (D) new A(obj)

8. 判断类A的两个对象a1与a2是否是同一个对象的方法是:

- (A) 利用调试器,查看a1与a2各成员数据的值是否相等  
(B) 重载 == 运算符,用它来判断各成员数据的值是否相等。  
(C) 判断这两个对象的地址值是否相等。  
(D) 比较创建时使用的构造函数的实参是否相同。

9. 已知obj是一个对象,下面哪一个表达式是不可能出现的?

- (A) obj.100 (B) !obj (C) obj++ (D) obj.100

10. 下面哪种情况不属于函数重载:

- (A) 类中定义的运算符函数  
(B) 同一个名字,参数个数不同  
(C) 派生类中重新定义了一个和基类中的原型完全相同的函数  
(D) 类中定义了两个同名、同参数表的函数,但其中一个是常成员函数

二、判断正误,对于你认为错误的论述,说明原因或举出反例。(每题2分,共20分)

1.使用语句A a=dynamic\_cast<A>(b);,可以将派生类B的对象b强制转换为基类A的对象。

2.在同一个类中,可以定义重载的成员函数 void f(int& a,int);和void f(int a,int);

3.虚拟继承的概念是为了实现多态性的要求而引入的。

4.若常量成员函数(用const修饰的成员函数)中调用同一个类中的虚函数f,那么函数f也一定是一个常量成员函数。

5.类D以public方式继承类B,若在这两个类中以同样的方式分别声明了一个同名的整型成员变量x,那么,为类D的对象分配内存空间时,不需要为类B中声明的x分配空间。

6.实例化派生类对象时,一定会调用到基类的某一个构造函数。

7.如果类A是类B的友员,类D以public继承方式从类B继承,则类A也是类D的友员。

8.对于任意按照语言规范定义并实现的类A,系统都可以为它提供一个无参数的构造函数,因此,在任何情况下都可以用new A的方式产生A类的对象。

9.异常是程序运行过程中产生的错误。

10. 设有函数说明 void f(const int&);则在调用该函数时,提供的参数既可以 int 型变量,又可以是 int 型常量。

三、指出下列程序代码中存在的错误并说明错误原因。(每题5分,共10分)

1.

<pre>#include&lt;iostream.h&gt; class A{ public:     A() {} private:     int a; };</pre>	<pre>class B{ public:     B() {} private:     int b; };</pre>	<pre>class C : public A, B { public:     C() {} private:     int c; };</pre>	<pre>void main() {     C c;     cout &lt;&lt; c.a &lt;&lt; c.b ;     cout &lt;&lt; c.c &lt;&lt; endl ; }</pre>
--	---	--	--

2.

<pre>//交换 A 类的两个对象 a 和 b 的值 void MySwap(A&amp; a, A&amp; b) {     A&amp; temp=a;     a=b;     b=temp; }</pre>
---

四、回答下列各题（每题 4 分，共 20 分）

- 说明为什么不能在类的静态成员函数的实现体中使用 **this** 指针。
- 举例说明 **protected** 关键字的两种用法和相应目的。
- 类的数据成员在哪些情况下必须在初始化列表中进行初始化。
- 说明出现在下面头文件中的预处理指令的作用。

```
#ifndef __MYFILE_H_
#define __MYFILE_H_
... //头文件内容
#endif //__MYFILE_H_
```

- 为了能够将类 **B** 的对象赋值给类 **A** 的对象，在定义这两个类时可以采取哪些手段？至少说明两种不同的处理方法。

五、阅读下面两个类的定义和部分实现代码，完成 3 个问题。（共 10 分）

<pre>#include&lt;iostream.h&gt; class Base { friend ostream&amp; operator&lt;&lt;(ostream&amp;, const Base&amp; ); public:     virtual ~Base( ) {} private:     virtual void Out(ostream &amp; os) const =0; };</pre>	<pre>class D2:public Base { public:     D2(int n):y(n) {}     virtual ~D2( ) {} private:     virtual void Out(ostream &amp; os) const         { os &lt;&lt; "Data=" &lt;&lt; y &lt;&lt; endl; }     int y; };</pre>
<pre>class D1:public Base { public:     D1(int n):x(n),y(n*n) {}     virtual ~D1( ) {} private:     virtual void Out(ostream &amp; os) const         { os &lt;&lt; "Data=" &lt;&lt; x+y &lt;&lt; endl; }     int x, y; };</pre>	<pre>int main( ) {     D1 d1(10);     D2 d2(20);     cout&lt;&lt;d1;     cout&lt;&lt;d2;     return 0; }</pre>

1.

实现 **Base** 类中声明的友元函数 **operator<<**，使得程序的两行输出分别为 **Data=110** 和 **Data=20**。（4 分）

friend ostream& operator<<(ostream& os, const Base& b)

```
{ b.Out(os); }
```

2. 说明为什么重载<<操作符时，不能将其定义成类的成员函数形式。(3分)

<<操作符是一个二元运算符，其左操作数总是一个流对象，不可能是本类对象。

3. 举例说明将Base类的析构函数定义成虚函数的目的或作用。(3分)

指向派生类对象的指针（或引用）总可以赋值给Base类的指针（或引用），如

Base \* pb=new D1(10,20); 若析构函数定义成非虚函数，则当释放时，只调用基类的析构函数，没有调用派生类的析构函数，这样会造成释放不完整。

六、写出下面程序的运行结果（每题5分，共10分）

1. ABCDCBA

2.

A0

B1

B2

B1

A0

七、(共20分，每问题10分)

```
#include<iostream.h>
```

```
class Monster
```

```
{
```

```
public:
```

```
    Monster(int hp,int att,int def)
```

```
        :hitpoint(hp),damage(att),defense(def) {}
```

```
    virtual ~Monster() {}
```

```
    bool fight(Monster & other);
```

```
    virtual void attack(Monster & other)=0;
```

```
    void ReduceHP(int harm)
```

```
    { hitpoint-=harm; if(hitpoint<0) hitpoint=0; }
```

```
    int GetHP() const {return hitpoint;}
```

```
    int GetDamage() const {return damage;}
```

```
    int GetDefense() const {return defense;}
```

```
protected:
```

```
    int hitpoint;
```

```
    int damage;
```

```
    int defense;
```

```
};
```

```
bool Monster::fight(Monster & other)
```

```
{
```

```
    while(true) {
```

```
        attack(other);
```

```
        if(other.GetHP()<=0) return true;
```

```
        other.attack(*this);
```

```
        if(hitpoint<=0) return false;
```

```
    }
```

```
}
```

```
class Dog:public Monster
```

```
{
```

```
public:
```

```
    Dog(int hp,int att,int def):Monster(hp,att,def) {}
```

```
    virtual ~Dog() {}
```

```
    virtual void attack(Monster & other)
```

```
{
```

```
    int harm=(damage-other.GetDefense()+5)*2;
```

```
    if (harm<2) harm=2;
```

```
    other.ReduceHP(harm);
```

```
}
```

```
};
```

```
class Cat:public Monster
```

```
{
```

```
public:
```

```
    Cat(int hp,int att,int def):Monster(hp,att,def) {}
```

```
    virtual ~Cat() {}
```

```
    virtual void attack(Monster & other)
```

```
{
```

```
    int harm=damage*2-other.GetDefense();
```

```
    if (harm<1) harm=1;
```

```
    other.ReduceHP(harm);
```

```
}
```

```
};
```

```
void main()
```

```
{
```

```
    Dog d(100,10,7);
```

```
    Cat c(120,8,9);
```

```
    if(d.fight(c))
```

```
        cout<<"Win"<<endl;
```

```
    else
```

```
        cout<<"Lost"<<endl;
```

```
}
```