面向对象专题练习

单选题

1. 构造函数是在（　　）时被执行的：B

A. 程序编译　　B. 创建对象　　C. 创建类　　D. 程序装入内存

分析：对象的生命周期：分配内存->构造->使用->析构->回收内存

2. 下面哪个不是构造函数的特征：D

A. 构造函数的函数名与类名相同　　B. 构造函数可以重载

C. 构造函数可以设置缺省参数　　D. 构造函数必须指定类型说明

分析：构造函数不可以有返回值，其专有目的就是初始化对象，因此不给程序员指定返回值的权利。

3. 关于成员函数特征的下列描述中，错误的是：A

A. 成员函数一定是内联函数　　B. 成员函数可以重载

C. 成员函数可以设置缺省参数值　　D. 成员函数可以是静态的

分析：类成员函数可以在类声明内同时定义也可以在类外定义，因此不一定是内联函数

4. 关于析构函数特征描述正确的是：C

A. 一个类中可以有多个析构函数　　B. 析构函数名与类名完全相同

C. 析构函数不能指定返回类型　　D. 析构函数可以有一个或多个参数

分析：析构函数是一个专有目的函数，就是为了对象收尾清理，因此不可以有返回值也不可以有参数，也因此只能有一个析构函数

5. 定义A是一个类，那么执行语句“A a, b(3),*p;”调用了几次构造函数：A

A. 2　　B. 3　　C. 4　　D. 5

分析：定义了两个对象a和b以及一个对象指针p，实际只创建了两个真实存在的对象，即调用了两次构造函数

6. 通常，拷贝构造函数的参数是：C

A. 某类的对象名　　B. 某类对象的成员名

C. 某类对象的引用　　D. 某类对象的指针

分析：参数为另一个本类对象的引用的构造函数成为拷贝构造函数，其目的为拷贝另一个对象的内容以初始化本对象，通过引用传参的目的是为了避免不必要的复制。通常还会将这个参数标记为const，如 A(const A& a) { // 拷贝行为 }

7. 没有使用访问控制属性关键字标记的类成员的访问控制属性为：A

A. private　　B. public　　C. protected　　D. friend

分析：如未显式注明，类成员的默认访问控制属性为private

8. 如果一个类的成员函数func()不应修改类的数据成员值，则应将其声明为：A

A. void func() const;　　B. const void func();

C. void const func();　　D. void func(const);

分析：在成员函数参数列表后使用const将其标记为常成员函数，常成员函数不可修改成员变量

9. 能通过对象名直接访问的是：B

A. private成员和protected成员　　B. public成员

C. public成员和protected成员　　D. public成员和没有指明访问属性的成员

分析：只有public(公有)成员可以通过 *对象名.成员名* 直接访问。没有指明访问控制属性的成员默认为private类型。注意区分“通过对象名访问”和“在类(定义)内部访问”

10. 在程序代码：M::M(int a, int *b) { this->x = a; this->y = b; }中，this的类型是：D

A. int*　　B. M&　　C. M　　D. M*

分析：this关键字在类定义内部表示“指向当前对象”的指针，因此其类型是 M*

11. this指针是C++实现（ B ）的一种机制

A. 抽象　　B. 封装　　C. 继承　　D. 多态

分析：this指针将对象与其成员函数在调用时联系起来，使得在外部看来每个对象都有自己的成员函数

12. 基类的_____成员在_____继承中成为派生类中的_____成员：C

A. protected, public, public　　B. private, public, protected

C. public, protected, protected　　D. protected, private, protected

分析：①访问控制属性在继承中不可能变宽松，只可能越来越私密

②基类的private成员在子类中不可见

13. 实现运行时的多态性要使用：D

A. 重载函数　　B. 构造函数　　C. 内联函数　　D. 虚函数

分析：运行时多态 = 虚函数重写 + 指向子类对象的父类指针

14. 要实现动态联编，必须通过(C)调用虚函数

A. 父类名字空间　　B. 子类指针　　C. 父类指针　　D. 子类对象名

分析：动态联编就是运行时多态，指通过指向**实际是子类对象**的父类指针调用子类中重写的虚函数

15. 设基类BASE中将成员函数func()标注为virtual，并在派生类DERIVED中对func()进行了重写，则下列代码实际调用的两个func()函数分别是_____和_____：D

```
DERIVED a; BASE *p = &a;
(*p).func();  p->func();
```

A. BASE::func(), BASE::func()　　B. BASE::func(), DERIVED::func()

C. DERIVED::func(), BASE::func()　　D. DERIVED::func(), DERIVED::func()

分析：指针p虽然是BASE*类型的，但它实际指向的对象a本质是一个DERIVED对象，它重写了虚函数func()，覆盖了父类的版本，因此p->func()实际调用的是子类版本。(*p).func()等价于p->func()

16. 以下哪项正确表示纯虚成员函数：C

A. virtual int vf(int);　　B. void vf(int) = 0;

C. virtual void vf() = 0;　　D. virtual void vf(int){}

分析：在虚函数声明后用 = 0 标记为纯虚函数

17. 关于纯虚函数和抽象类叙述正确的是：C

A. 成员函数全为纯虚函数的类才叫抽象类

B. 成员函数全为虚函数的类叫做抽象类

C. 包含纯虚函数的类就叫做抽象类

D. 若纯虚函数有定义，则抽象类可以有实例化对象，否则不行

分析：纯虚函数不可以被定义；只要包含纯虚函数的类就叫抽象类；抽象类可以有其他的成员函数；抽象类表示抽象事物因此不能实例化对象

18. 设A←B←C←D。A中有virtual int func() = 0; B中有virtual int func(); C中有int func(); 则：A

A. A无法实例化　　B. B无需定义func　　C. C是抽象类　　D. D可以继续重写func

分析：A中定义了纯虚函数，因此A是抽象类，不允许实例化。B将函数func重写了，虽然仍标记为虚函数，但不是纯虚函数，因此必须要定义。C中继续重写了func，但不再标记为虚函数，因此其派生类D中就无法继续重写func了。

19. 设A←B←C←D且它们仅有默认构造函数，实例化一个D类对象和销毁它依次调用的函数分别为：B

A. D()和~D()

B. A(), B(), C(), D() 和 ~D(),~C(),~B(),~A()

C. D(), C(), B(), A() 和 ~D(),~C(),~B(),~A();

D. A(), B(), C(), D() 和 ~A(),~B(),~C(),~D()

分析：构造对象时，先构造父类，再构造子类；析构时，先析构子类，再析构父类。类比记忆：栈

20. 下面对类静态成员的描述中，正确的是：B

A. 类的不同对象有不同的静态数据成员值

B. 静态成员是不属于该类的任何一个对象

C. 类的每个对象都有自己的静态成员

D. 静态成员函数定义内可以使用this关键字

分析：静态成员是属于该类的，和其实例化对象无关系，因此静态成员函数内不能使用this指针。成员变量和成员函数都可以声明为静态的。

21. 若F←S，且F中定义了公有属性的静态成员变量a，则在类外部对a的访问错误的是：C

A. cout << F::a;

B. cout << S::a;

C. F obj; cout << obj.a;

D. S obj; cout << F::a;

分析：a本是F中的静态变量，因其访问属性是公有，会被继承到类S中，则S中也含有这个静态变量a，因此F::a和S::a都可以正确访问a。

22. 有如下类声明，B构造函数定义正确的是：B

A. B::B(int a,int b): x(a),y(b){ }

B. B::B(int a,int b): A(a),y(b){ }

C. B::B(int a,int b): x(a),B(b){ }

D. B::B(int a,int b): A(a),B(b){ }

```
class A {
    private: int x;
    public:  A(int n){ x=n;}
};
class B : public A {
    private: int y;
    public:  B(int a,int b);
};
```

分析：A是B的基类，构造B类对象时构造函数的调用次序为先A再B，然而A没有默认构造函数，要构造A类对象必须显式调用有参构造函数A(int)，所以需要将此次调用显式放在B类构造函数的**初始化列表**中，表明在B(int, int)执行前执行A(int)。

23. 有如下类声明，B构造函数定义正确的是：C

A. B::B(int a,int b): x(a),y(b){ }

B. B::B(int a,int b): A(a),y(b){ }

C. B::B(int a,int b): obj(a),y(b){ }

D. B::B(int a,int b) { y = a; obj.x = b; }

```
class A {
    public:  int x;
             A(int n){ x=n;}
};
class B {
    public:  int y;
             A obj;
             B(int a,int b);
};
```

分析：B类中有类型为A的成员变量obj，类A没有默认构造函数，那么在B的构造函数中就不能先构造B再通过赋值初始化obj，选项D错误。必须在构造B时，执行B构造函数之前就将obj构造好，这个微妙的位置就是初始化参数列表，选项C正确，其含义是在B构造逻辑执行之前执行 A obj(a); 构造好obj。请对比此题和22题的异同。

24. 下面关于友元函数的描述中，正确的说法是：A

A. 友元函数是独立于当前类的外部函数　　B. 一个友元函数不能同时定义为两个类的友元函数

C. 友元函数必须在类的外部声明　　D. 在外部声明友元函数时，必须加关键字friend

分析：友元函数是不属于类的普通函数，类声明中可使用friend关键字将外部函数声明为自己的友元，使其能访问该类对象的私有/保护成员

25. 友元的作用之一是：D

A. 丰富成员函数的种类　　B. 加强类的封装性

C. 实现数据的隐藏性　　D. 提高程序的运行效率

分析：友元使得外部函数可以访问类封装的成员，就像墙上开了一个小孔、房子开了一个后门，实际上削弱了类的封装性，换来了更高的程序运行效率(直接访问私有成员而无需通过getter/setter等接口)。

26. 类A声明中有：A operator+(int n); 则其目的是让A类对象a支持下面哪种写法：C

A. a.+(2)　　B. a.add(2)　　C. a + 2　　D. 2 + a

分析：A重载了运算符加号，operator+函数的参数是一个int数据，表示可以使用加号时对一个A类对象和一个整数进行运算，且加号左端是对象，右端是int数据，它将作为operator+函数的参数传递。

判断题

27.判断下列说法的正误：

无论什么方式的继承，基类的私有成员都不能被派生类访问。（ √ ）

任何一个对象只能属于一个具体的类。（ √ ）

如果派生类没有实现虚函数，那么它将使用他的基类的虚函数。（ √ ）

在C++中，既允许单继承，又允许多继承。（ √ ）

派生类从基类派生出来，它不能派生新的派生类。（ × ）

构造函数和析构函数都不能重载。（ × ）

在C++项目中，通常类的声明位于.h文件中，类的定义位于.cpp文件中。（ √ ）

简答题

28. 简答题：

（1）请简述overload和override的异同。

同：①目的都是实现**代码复用**、减少代码冗余　②核心方法都是**自动匹配**实际需要调用的函数

异：①overload是重载，在**编译期**根据函数参数类型决定实际调用哪个函数，是**静态联编**

②override是重写，在**运行时**调用指针指向的实际对象重写的虚函数，是**动态联编**

（2）请简述面向过程和面向对象思想的主要区别。

面向过程思想旨在将过程进行**流程化拆解**，对**①参数②返回值③处理流程**进行定义，并通过**函数调用**的方式实现程序系统功能。

面向对象思想旨在**识别**系统中的**事物**，并**抽象**得到它们的**①属性②行为**，将其定义为类的成员数据和成员方法来满足系统功能。

（3）请分别从代码编写和底层原理的角度描述动态联编的实现方式。

代码编写：**虚函数重写** + 通过**指向子类对象的父类指针**调用虚函数

底层原理：C++底层使用了**虚函数表**的方式实现对动态联编的支持

具体方式为当子类重写父类的虚函数时，会**覆盖**虚函数表中相应的函数指针，这样父类指针实际指向子类对象时通过指针调用的就是子类重写的虚函数，因该对象的虚函数表内此函数已经是子类重写的版本

代码阅读题

29. 写出下面程序的输出结果

```
class A {
public:
    virtual void act1(){ cout << "A" << endl; }
    void act2(){ act1(); }
};
class B : public A {
public:
    void act1(){ cout << "B" << endl; }
};
void main() {
    A a,*p; B b; p=&b;
    b.act1 ();
    p->act1();
    p->act2();
}
```

指A::act1()

分析：

①b.act1()：直接通过对象名调用的是B类对象b的成员函数B::act1()

②p->act1()：p是A*指针，但实际指向B类对象b，因此实际调用B::act1（动态联编）

③p->act2()：B类继承了A的act2，而act2定义中调用的是A::act1()

B
B
A

30. 写出下面程序的输出结果

```
class A {
public:
        int n;
        A(int n) : n(n) { cout << "A cons\n"; }
        ~A() { cout << "A dest" << endl; }
};
class B : public A{
public:
        int n;
        B(int n1, int n2) : A(n1), n(n2) { cout << "B cons" << endl; }
        void print() { cout << A::n << endl; }
        ~B() { cout << "B dest" << endl; }
};
void main() {
        B a(1, 2), b(3, 4);
        a.print();
}
```

这里构造了两个对象

这里析构了两个对象

```
A cons
B cons
A cons
B cons
1
B dest
A dest
B dest
A dest
```

分析：

①构造顺序先父后子，析构顺序先子后父

②在类继承中有重名成员时，通过类名字空间分辨

③对象在定义处构造，在生命周期结束时析构

31. 写出下面程序的输出结果

```
class A {
private:
        int n;
public:
        A(int n) : n(n) {}
        void func() { cout << "n = " << n << endl; }
        void func() const { cout << n << " = n" << endl; }
};
void main() {
        A a1(1);
        const A a2(2);
        a1.func();
        a2.func();
}
```

分析：

const A a2(2) 将a2定义为一个常对象

void func() 和 void func() const 实际上构成了**函数重载**

普通对象a1调用的是普通成员函数void func()

常对象a2调用的是常成员函数void func() const

n = 1
2 = n

代码补全题

32. 补全下列代码段，利用运行时多态技术使程序输出为2

```
class Base{
public:
    virtual void fun(){ cout<<1; }
};
class Derived: public Base {
public:
    void fun( ) { cout<<2; }
};
int main( ) {
    Derived d;
    Base *p = &d;
    p->fun();
    return 0;
}
```

分析：

运行时多态 = 虚函数重写 + 通过指向子类对象的父类指针调用虚函数

33. 以下程序是定义一个计数器类Counter，对其重载运算符“+”，请补全代码

```
class Counter { private: int n;
public:
    Counter() { n=0;}
    Counter(int i){ n=i;}
    Counter operator+ (Counter &c)  {
        return Counter(n + c.n);
    } // 运算符重载
    void display() { cout<<"n="<<n<<endl; }
};
void main() {
    Counter c1(5), c2(10), c3;
    c3 = c1 + c2;
    c1.display(); c2.display(); c3.display();
}
```

分析：根据函数返回内容和c3 = c1 + c2; 这一行，可以判断返回值为Counter，另外参数为 const Counter &c 或 Counter c 也可以

程序设计题

34. 设计一个汽车类Vehicle，包含数据成员车轮数，由它派生出类Car和类Truck，前者包含载客数，后者包含载重量。要求实现类的封装性并提供相应的数据读写接口。

```
class Vehicle {
private:
  int wheels;
public:
  Vehicle(int wheels) : wheels(wheels) { }
  int get_wheels() const { return wheels; }
  void set_wheels(int wheels) {
    this->wheels = wheels;
  }
};
```

```
class Car : public Vehicle {
private:
  int passenger;
public:
  Car(int passenger, int wheels) : Vehicle(wheels), passenger(passenger) {}
  int get_passenger() const { return passenger; }
  void set_passenger(int passenger) { this->passenger = passenger; }
};
```

分析：

①封装性通过private数据成员体现

②数据读写接口指getter/setter函数

③Truck的代码类似，略

35. 请设计并实现类A←B←C，要求支持统计各类的实时对象数量。

```
class A {
private:
    static int count_A;
public:
    A() { count_A++; }
    ~A() { count_A--; }
    virtual int get_count() {
          return count_A;
    }
};
int A::count_A = 0;
```

```
class B : public A {
private:
    static int count_B;
public:
    B() { count_B++; }
    ~B() { count_B--; }
    virtual int get_count() {
          return count_B;
    }
};
int B::count_B = 0;
```

```
class C : public B {
private:
    static int count_C;
public:
    C() { count_C++; }
    ~C() { count_C--; }
    int get_count() {
          return count_C;
    }
};
int C::count_C = 0;
```

分析：

①对象计数需要通过类的静态变量来实现

②注意get_count需要设置成虚函数以便于调用实际对象真正的get_count函数

③静态数据成员必须在类声明外初始化，static关键字只需要在声明处标记

35. 燃烧我的卡路里。小明(130斤)和小红(100斤)决定减肥，请你编写类和对象实现这一过程：

① 吃东西(eat)增重1斤 ② 跑步(run)减重0.5斤

③ 小明跑步三次进食一次，小红跑步两次进食两次，请判断他们是否减肥成功

```
class Person {
private:
    float weight;
    float weight0;
public:
    string name;
    Person(float weight, string name) : weight(weight), weight0(weight), name(name) {}
    void eat() { weight += 1.f; }
    void run() { weight -= 0.5f; }
    bool is_success() {
        return weight < weight0;
    }
};
```

```
int main() {
    Person p1(130, "小明");
    Person p2(100, "小红");

    p1.run();
    p1.run();
    p1.run();
    p1.eat();

    p2.run();
    p2.run();
    p2.eat();
    p2.eat();

    cout << p1.name
         << (p1.is_success() ? "成功" : "失败") << endl;
    cout << p2.name
         << (p2.is_success() ? "成功" : "失败") << endl;

    return 0;
}
```

分析：面向对象解决编程问题步骤：

①识别并抽象出事物及其属性和方法

②定义类

③实例化对象，在系统中运行

中国大学MOOC
搜索：C++不挂科