《面向对象程序设计》期末考试答题页

姓名： 班级： 学号：

1. 单选题（每题4分，共72分）

1. 7. 13.

2. 8. 14.

3. 9. 15.

4. 10. 16.

5. 11. 17.

6. 12. 18.

1. 填空题（每题2分，共28分）

19.

20. 21.

22.

23．

24．

25．

《面向对象程序设计》期末考试

**一、单选题（每题4分，共72分）**

**01、**根据下面程序，判断下列说法不正确的是：D

#include <iostream>

using namespace std;

class X {

private:

int data = 0;

friend class Y;

friend void print(X &x);

};

class X\_Son : public X {

private:

int data = 1;

};

class Y {

private:

int data = 2;

friend class Z;

};

class Z {

private:

int data = 3;

};

void print(X &x) {

cout << x.data << endl;

}

A) print是X的友元函数

B) Z是Y的友元类

C) Y是X的友元类

D) Z是X的友元类

**02、**下列关于静态成员说法正确的是：A

A) 同一个类的所有类对象，共享该类的静态数据成员，即所有对象中的该数据成员存储在同一内存位置

B) 类的常量静态成员数据只能在类外进行初始化

C) 类的静态成员数据只能通过类名来访问

D) 静态成员函数属于整个类，在第一个类实例化对象创建的时候分配它的内存空间

**03、**下列关于类对象说法错误的是：C

A) 函数静态对象，在程序执行到该局部静态对象的代码时被初始化；程序第二次执行到该对象代码时，不再初始化

B) 在不同编译单元中，全局对象的初始化顺序不确定

C) 当A类对象是B类的常量静态成员时，可以在B类内初始化

D) 当A类对象是B类的静态非常量成员变量时，只能在B类外初始化

**04、**定义如下的Test类，下列说法正确的是：A

class Test {

const int member;

float another\_member;

public:

Test(int mem):member(mem) {another\_member = 1.0;}

int MyMember() const {return member;}

float MyAnotherMember() {return another\_member;}

}

A) member的值在不同的Test对象中可以不同

B) 初始化数据成员member时，可以采用类似于another\_member的初始化方式，在构造函数的函数体中赋值

C) 成员函数MyMember的函数体内可以增加语句，修改another\_member的值

D) 定义一个Test类的常量对象，其可以调用MyMember和MyAnotherMember两个成员函数

**05、**阅读代码，下列说法错误的是：C

#include <iostream>

#include <string.h>

using namespace std;

class A {

int \*m\_arr;

int m\_size;

public:

A(int i):m\_size(i) {

m\_arr = new int[m\_size];

memset(m\_arr, 0, m\_size\*sizeof(int));

}

~A() {

delete []m\_arr; //(2)

}

void set(int index, int value) {

m\_arr[index] = value;

}

void print() {

cout << "m\_arr: ";

for (int i = 0; i < m\_size; ++ i) {

cout << " " << m\_arr[i];

}

cout << endl;

}

};

int main() {

A a(5);

A b = a; //(1)

b.set(3, 10);

a.print(); //(3)

return 0;

}

A) (3)执行完之后，该程序的输出为: m\_arr: 0 0 0 10 0

B) 程序中的语句(1)调用了编译器自动合成的拷贝构造函数

C) 该程序可以正常执行，且程序的返回值为0

D) 如果将程序中的语句(2)改成delete m\_arr，则可能会出现内存泄漏

**06、**定义如下所示的Base类和Derive类，下列说法错误的是：C

#include <iostream>

using namespace std;

class Base {

int data;

public:

Base() : data(0) { cout << "Base::Base(" << data << ")\n"; }

Base(int i) : data(i) { cout << "Base::Base(" << i << ")\n"; }

};

class Derive : public Base {

int data{2018};

public:

Derive() {}

Derive(int i) : Base(i) {}

void print() { cout << "data = " << data << endl; }

};

int main() {

Derive obj1;

Derive obj2(2019);

obj1.print();

obj2.print();

return 0;

}

A) Derive类的默认构造函数没有显式调用Base类的构造函数，此时编译器会自动调用Base类的默认构造函数

B) Derive类的普通构造函数可以在初始化列表中显式调用Base类的普通构造函数

C) 该程序的输出为Base::Base(0)\nBase::Base(2019)\ndata = 2018\ndata = 2019\n

D) obj1析构时先执行Derive类的析构函数，再执行Base类的析构函数

**07、**有如下程序段，程序的输出为：B

#include<cstdio>

using namespace std;

class A {

public:

A() { this->f(); }

virtual void f() { printf("A"); }

};

class B: public A {

void f() { printf("B"); }

};

int main() {

A\* t = new B();

t->f();

delete t;

return 0;

}

A) AA B) AB C) BA D) BB

**08、**对于std::string和std::vector<char>定义的字符串str，以下选项正确的是：B

A) 都可以使用str.length();获得字符串长度

B) 都可以使用for(char c: str)遍历字符串中的所有字符

C) 都可以使用cin>>str;从标准输入输入字符串

D) 都可以使用str+="abc";向字符串尾部添加字符

**09、**有如下程序段，则以下选项中哪一项不可能导致it失效：D

#include<vector>

using namespace std;

int main() {

vector<int> vec;

...

auto it = vec.begin() + 5; //假设此时vec的长度大于5

...

}

A) vec.push\_back(2);

B) vec.erase(vec.begin());

C) vec.insert(it, 2);

D) vec[5] = 2;

**10、**以下哪个不是使用设计模式进行编程的特点：A

A) 牺牲了代码维护的代价，降低模型复杂程度

B) 不优先使用继承

C) 指导经验不足的程序员灵活运用面向对象的特性

D) 快速构建适用于不同场景的程序框架

**11、**设计模式的三个分类中不包括：C

A) 行为型 B) 结构型 C) 组合型 D) 创建型

**12、**以下对模板方法和策略模式说法不正确的是：A

A) 在设计思路上，模板方法模式优先考虑组合，策略模式优先考虑继承

B) 使用模板方法可以定义抽象概念，拥有不同的实现

C) 使用策略模式可以减小单一类承担的指责

D) 使用模板方法和策略模式定义的抽象概念时，都直接使用接口类来引用这一概念

**13、**以下关于迭代器模式的说法不正确的是：C

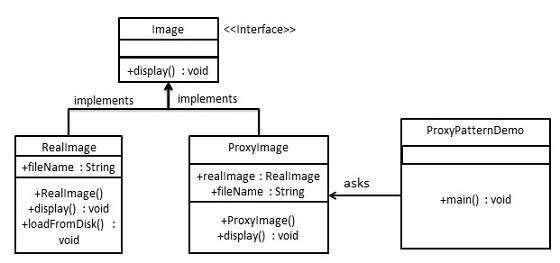
A) 迭代器是对“数据访问”设计的接口

B) 迭代器提供一种方法访问一个聚合对象中各个元素，而不用关心具体存储类的算法

C) 迭代器模式中，编写上层代码时需要根据使用需求和性能需求同时使用迭代器和存储类的方法对存储的数据进行访问

D) 迭代器类通常设计为存储类的友元，从而迭代器类可以访问存储类内部的数据

**14、**以下UML体现的设计模式是：C



A) 适配器模式 B) 接口模式 C) 代理/委托模式 D) 装饰器模式

**15、**现有一个实现数据结构“栈”的需求，程序员X对栈的设计进行了如下思考，其中你认为有问题的是：C

A) 可以利用C++ STL中的vector，使用适配器模式将其适配为符合栈的抽象接口的类

B) 对于栈中元素的内存分配和释放，可以使用策略模式来实现不同内存分配/释放算法

C) 如果使用装饰器模式对一个栈数据结构进行多次功能扩展，就不会出现多重嵌套的问题

D) 可以使用代理/委托模式设计智能指针，对栈中元素进行内存分配和释放

**16、**以下关于装饰模式和代理模式说法错误的是：B

A) 可以把“装饰”看成是一连串的“代理”。

B) 在对类进行装饰时，被装饰对象增加的行为通常重新实现原有功能。

C) 代理模式中少见多重嵌套。

D) 装饰模式通过对象的组合修改对象的功能。

**17、**以下关于工厂模式和抽象工厂模式说法正确的是：A

A) 工厂方法的目的是构造单个类的对象

B) 在工厂方法中，构造多个有特定组合方式的对象，应当为每一个组成成分单独设立工厂

C) 工厂方法不能处理不同重载的构造函数

D) 工厂模式中，对象的删除不可以交由用户负责

**18、**以下关于单例模式说法不正确的是：C

A) 单例模式只需要显式删除构造函数与拷贝构造函数即可保证只能构造一份类的实例

B) CRPT模式与虚函数不同，实现了编译时多态

C) 静态虚函数可以实现和单例模式类似的“全局访问”功能

D) 我们可以使用惰性初始化，返回单例实例的引用。但这种方法仍然存在用户自行delete所生成单例实例，引发错误的可能

**二、填空题（每空2分，共28分）**

**19、**含有纯虚函数的类称为 抽象类 ， 不可以 (可以/不可以)构造该类型的对象，该纯虚函数 可以 (可以/不可以)提供实现。

**20、**赋值运算符的重载函数 不能 （能/不能）定义为全局函数。

**21、**在派生类中，通过 using 关键字能去对除基类成员函数的屏蔽，使之重新可用。

**22、**内联函数 可以 执行类型检查， 可以 进行编译期错误检查， 可以 操作私有数据成员。（三个空都选填可以/不可以）

**23、**代理/委托模式与适配器模式均是在被访问对象之上进行封装，提供被封装对象的功能接口供外部使用。其中\_适配器\_模式可能会改变接口,\_代理/委托\_模式可能会增加控制。

**24、**单例模式可以用于实现全局计数器。如果我们有多种不同的全局计数器，可以使用奇特的递归模板模式(Curiously Recurring Template Pattern,CRTP)实现，此时计数器逻辑实现在\_派生类\_(基类/派生类)，创建单例的静态函数实现在\_基类\_(基类/派生类)

**25、**快餐店(Class Store)提供披萨(void salepizza), 提供汉堡(void saleburger)、提供可乐(void salecola)，假设披萨/汉堡/可乐分别有n、m、k三种，不同披萨/汉堡/可乐品种之间实现方法不同，为适配不同的销售组合：若采用模板方法，建立快餐店基类后还需建立(n\*m\*k)个子类；若采用策略模式，建立快餐店基类和3个抽象策略类后还需要实现 (n+m+k)个策略实现类。