高级程序设计期末PPT复习百问

2018年12月24日 23:12

1. 数据抽象?
   数据的使用者只需要知道对数据所能实施的操作以及这些操作之间的关系，而不必知道数据的具体表示形式。
2. 数据封装?
   把数据及其操作作为一个整体来进行实现。数据的具体表示对使用者是不可见的（封装），对数据的访问（使用）只能通过封装体所提供的对外接口（允许的操作）来完成。
3. 数据抽象和数据封装的区别?
   抽象强调的是外部行为；封装强调的是内部实现。
4. C/C++函数和数学意义上的函数的区别?
   a. C/C++函数可能无返回值
   b. C/C++函数可能存在副作用，改变了变量
   c. 函数可能存在静态局部变量
5. 直接使用栈类型数据存在的问题?
   a. 操作必须知道数据的表示，并且当数据表示发生变化时将要修改对数据操作的代码
   b. 麻烦且易产生误操作
6. 基于过程抽象的使用栈类型的数据存在的问题?
   a. 数据类型的定义与操作的定义是分开的，二者之间没有显式的联系，
      push、pop在形式上与下面的函数f没有区别，函数f也能作用于st：
      ```
      void f(Stack &s) { ..... }
      f(st); //操作st之后，st可能不再是一个“栈”了!
      ```
   b. 数据表示仍然是公开的，无法防止使用者直接操作栈数据，因此也会面临直接操作栈数据所带来的问题：
      ```
      st.top--;
      st.buffer[st.top]=12;
      ```
   c. 忘记初始化
7. 对象？通信？类？继承？书P201
8. 对象和类体现了?
   数据抽象和封装的概念
9. 单继承和多继承?
   单继承：一个类最多有一个直接父类
   多继承：一个类可以有多个直接父类
10. 多态？某一论域中的一个元素存在多种解释

a. 一名多用：函数名重载
b. 类属性：类属函数、类属类型
c. 对象类型的多态：子类对象既属于子类，也属于父类。
d. 对象标识的多态：父类的引用或指针可以引用或指向父类对象，也可以引用或指向子类对象。
e. 消息的多态：发给父类对象的消息也可以发给子类对象，即，一个消息有多种解释（父类与子类有不同解释）。

11. 多态的好处?
    a. 易于实现程序高层（上层）代码的复用。
    b. 使得程序扩充变得容易（只要增加底层的具体实现）。
    c. 增强语言的可扩充性（操作符重载等）。
12. 成员的访问控制
    a. public：成员的访问不受限制
    b. private：成员只能在本类和友元中访问
    c. protected：成员只能在本类、友元和派生类中访问
13. 对象的创建方式?
    a. 直接方式：例如A a;
    b. 间接方式：new、delete
14. 默认构造函数的作用?
    a. 在创建对象时，可以显式地指定调用对象类的某个构造函数。
    b. 如果没有指定调用何种构造函数，则调用默认构造函数初始化。
15. 对象创建后能再调用构造函数？ (X)
16. 创建动态的对象数组（new <class_name>[n]）时只能用默认构造函数初始化
17. const常量数据成员和引用类型数据成员
    a. 不能在说明他们的时候初始化，也不能用赋值操作初始化
    b. 应该使用成员初始化表初始化
18. 成员初始化表中成员初始化的书写次序并不决定它们的初始化次序，只有类定义中的说明次序才能决定
19. 创建含有成员对象类的对象：
    a. 调用本身类的构造函数->进入函数体之前，调用成员对象类的构造函数（按说明次序调用）->执行本身类的构造函数的函数体
20. 消亡含有成员对象类的对象：
    a. 调用本身类的析构函数->本身类的析构函数的函数体执行完毕->按构造时的逆序逐个调用成员对象的析构函数
21. 调用拷贝构造函数需要<类名>(const <类名>&)?
    a. 如果不用引用类型，就会陷入无穷的递归调用拷贝构造函数之中
22. 何时调用类的拷贝构造函数
    a. 定义对象

b. 把对象作为值传给函数
c. 把对象作为值返回

23. 常成员函数：用于获取对象的状态而不改变对象的值

```
class A{
    void F() const;
}
```

24. 成员函数加上const修饰符的作用?
    a. 在const成员定义的地方告诉编译程序该成员函数不应该改变对象数据成员的值
    b. 在使用（调用）const成员函数的地方，告诉编译程序该成员函数不会改变对象数据成员的值
25. 静态数据成员（static）？
    a. 作用：完成同类对象之间的数据共享
    b. 静态数据成员对于该类的所有对象只存在一个拷贝
26. 静态成员函数?
    a. 只能访问静态成员，没有隐藏的this指针，因为根本就不需要（总共只有一份拷贝）
    b. 访问方式：
        i. 通过对象访问
        ii. 通过类名前加域名受限访问：e.g：A::getshared();
27. 友元函数、友元类、友元类成员函数？（以class A为例）
    a. friend void func();
    b. friend class B();
    c. friend void C::f();
    d. 意思是，func、B、C::f()可以访问类A中的private成员
28. 注意等号的区别?
    a. 初始化时候的等号：调用拷贝构造函数
    b. 赋值时的等号：调用赋值操作符“=”的重载函数
29. 一些重载的实例
    a. 作为成员函数重载+：Complex operator + (const Complex &x) const;
    b. 作为全局函数重载+：Complex operator + (const Complex &x1,const Complex &x2);
    c. 复合赋值运算符的重载：
        i. Complex &operator +=(const Complex &rhs);
    d. 前置的++重载函数：

```
Counter &operator++(){
    value++;
```

```
        return *this;
    };
```

e. 后置的++重载函数：

```
    const Counter operator ++(int){
        Counter temp=*this;
        ++(*this);
        return temp;
    }
```

f. 赋值操作符“=”的重载：A& A::operator = (const A& a);

g. 为了提高临时变量消亡又赋值时候的程序效率C++11增加的新转移赋值操作符：A& operator=(A&& x)

h. 操作符“[]”的重载：char/int & operator [](int i)

i. 操作符new的重载：void *operator new(size_t size,para a,...);//必须作为静态的成员函数重载，size意思是分配的空间大小，static可不写，调用时p=new(para a,...)A(A类构造函数所需要的参数)

j. 操作符delete的重载：void operator delete(void *p, size_t size);//返回类型必须是void，第一个必须是void*指向对象的内存空间，第二个参数可有可无，有则定是size

k. 函数调用操作符“（）”的重载：int operator()(按照需要添加形参)

l. 类成员访问操作符“->”的重载：

```
    class B{ ‘
        A *p_a;
        int count;
        B(A *p){
            p_a=p;
            count=0;
        }
        A *operator ->(){……}
    }
```

   i. 使用：记住加上引用

m. A类下的自定义类型转换：operator int(){……}

   i. 为了避免歧义问题（a转b，b转a均可），可以用显示类型转换或者为类A的构造函数定义explicit：explicit A(<参数列表>){……}，禁止其用于隐式类型转换符

n. 操作符“<<”和“>>”的重载：

   i. ostream & operator<<(ostream &out, <CLASS NAME> &obj);

   ii. istream & operator>>(istream &in, <CLASS NAME> &obj);

iii. 同时类中要声明上面两个全局函数为友元函数：

1) friend ostream & operator<<(ostream &out, <CLASS NAME> &obj);
2) friend istream & operator >> (istream &in, <CLASS NAME> &obj);

30. 为什么对某个类重载插入操作符“<<”和抽取操作符“>>”时，须作为全局函数来重载?

a. 如果作为友元全局函数重载，在传递参数时左右操作数都是通过参数传递，这样可以避免作为成员函数传递参数时，左操作数通过this指针传递而引起的this指针混淆的问题，同样也可以避免不能在已封装好的istream、ostream类中重载流插入和流抽取操作符的问题

31. 重载<<的三个引用的作用？为什么第一个引用不能用const，第二个往往可以用?

ostream & operator<<(ostream &out, const <CLASS NAME> &obj);

a. 首先因为ostream对象不能复制，所以必须是引用；其次引用可以减少一次拷贝，提高效率；最后，为了体现实现流的连续输出，达到用多个输出操作符操作同一个ostream对象的效果应该返回引用类型。相反，如果不是引用，程序返回的时候就会生成新的临时对象，多次调用了拷贝构造函数，不仅浪费了空间和效率，也可能出错。
b. 因为我们无法直接复制一个ostream对象，所以第一个形参应该是引用；该形参不能是const是因为向流写入内容会改变其状态
c. 第二个形参一般应是对要输出的类类型的引用，该形参是一个引用以避免复制实参，减少一次拷贝；输出时设为const，因为输出一般不会改变该对象，输入时则不能用const，因为输入一般会改变改变该对象。

32. 赋值操作符A& A::operator = (const A& a);为什么一定要重载为成员函数?

同理分析[]、()、->。

程序中如果没有为一个C++类定义赋值操作符重载函数，编译器会进行隐式的定义，这样倘若再定义全局赋值运算符重载函数，将会产生二义性，编译器会报错；而使用成员函数重载时类的this指针会被绑定到运算符的左侧运算对象，这就能与隐式定义的赋值操作符重载函数区分开来，从而避免二义性的问题。

33. 赋值操作符重载注意自赋值问题！要排除这种情况：如果类里面含有指针指向动态分配的资源的话，那么自身赋值就可能把自己释放，例如

```
A& operator=(const A&a) {
    delete ptr;
    ptr = new char[strlen(a.ptr)+1];
    strcpy(ptr,a.ptr);
    return *this;
```

}

就可能在自赋值的过程中先释放了自己的空间，导致指针指向已经被删除的对象。

==>修改方案：添加：if(&a==this)return *this;

34. 调用delete撤销一个对象的时候，调用对象的析构函数->调用delete的重载函数传对象地址到第一个形参p->若有第二个参数，则会把欲撤销的对象的大小传给它。
35. 一些λ表达式：
    a. cout << [](int x)->int { return x*x; }(3);//输出9
    b. f([](int x)->int { return x*x; });//将λ表达式的结果作为实参传给f
36. 常用的λ表达式格式：
    a. [<环境变量使用说明>]<形式参数>-><返回值类型><函数体>
    b. 环境变量使用说明：
        i. 空：不能使用外层作用域的变量
        ii. &：按引用方式使用外层作用域的变量（可以改变这些变量的值）
        iii. =：按值方式使用外层作用域的变量（不可改变这些变量的值）
37. 子类型？
    a. 定义：把以public方式继承的派生类看作是基类的子类型。
        i. 对基类对象能实施的操作也能作用于派生类对象。
        ii. 在需要基类对象的地方可以用派生类对象去替代。
38. 在拥有继承机制以后，一个类的成员有哪两种被外界使用的场合？
    a. 通过类的对象（实例）调用
    b. 在派生类中使用
39. 派生类操作不能用于基类对象
40. 派生类指针变量不能指向基类对象，否则可能通过指针向基类发送不能处理的消息，但基类指针可以指向派生类对象。
41. 基类对象不能赋值给派生类对象，否则可能导致派生类实例对象有不能确定的成员数据，但是派生类对象可以赋值给基类对象，多余的数据成员被忽略。
42. 如果我们想将某个类用作基类，则该类必须已经定义而非仅仅声明
43. 继承方式的调整：
    a. 格式：[public/protected/private:] <基类名>::<基类成员名>
44. 创建派生类对象的顺序：
    a. 先执行基类的构造函数，再执行派生类的构造函数
    b. 默认调基类的默认构造函数，调非默认构造函数需要在派生类构造函数的成员初始化表中指出
45. 如果一个类D既有基类B、又有成员对象类M，则在创建D类对象时，构造函数的执行次序为：B->M->D；当D类的对象消亡时，析构函数的执行次序为：D->M->B

46. 派生类的拷贝构造函数：
    a. 隐式拷贝构造函数调用基类的拷贝构造函数
    b. 自定义的拷贝构造函数默认情况下调基类的默认；需要时用成员初始化表显式指出调用的基类的拷贝构造函数
47. 派生类对象的赋值操作?
    a. 隐式的赋值操作：调用基类的赋值操作+对派生类成员按逐个成员赋值
    b. 显式定义的赋值操作：需要显式调用基类的赋值操作符对基类成员赋值
        => *(A*)this = b; //调用基类的赋值操作符对基类成员进行赋值
        =>也可写成：this->A::operator =(b);
48. 继承和聚集?
    a. 聚集：class A{}；class B{A a}；而非继承：class B：public A{}；云云
    b. 聚集可以避免继承与封装的矛盾：一个类只有public对外接口，不需要protected
    c. 继承机制更容易实现类之间的子类型关系
49. 消息的动态绑定：
    a. 基类中使用虚函数
50. 虚函数的作用?
    a. 实现消息的动态绑定
    b. 指出基类中可以被派生类重定义的成员函数
51. 基类的构造函数中对虚函数的调用不进行动态绑定。why?
    a. 从虚函数的作用角度考虑，如果父类的构造函数设置成虚函数，那么子类的构造函数会“覆盖”掉父类的构造函数。而父类的构造函数就失去了一些初始化的功能。这与子类的构造需要先完成父类的构造的流程相违背了。
    b. 从系统的角度来考虑，虚函数的调用是需要通过“虚函数表”来进行的，而虚函数表也需要在对象实例化之后才能够进行调用。在构造对象的过程中，还没有为“虚函数表”分配内存。所以，这个调用也是违背先实例化后调用的准则。
    c. 从运行和编译的角度考虑，虚函数的调用是由父类指针进行完成的，而对象的构造则是由编译器完成的，由于在创建一个对象的过程中，涉及到资源的创建，类型的确定，而这些是无法在运行过程中确定的，需要在编译的过程中就确定下来。而多态是在运行过程中体现出来的，所以是不能够通过虚函数来创建构造函数的。
52. 基类的析构函数对虚函数的调用可以进行动态绑定，而且往往需要虚函数，why?
    a. 由于虚函数是释放对象的时候才执行的，所以一开始也就无法确定析构函数的。而由于在析构的过程中，先析构子类对象，后析构父类对象，因此需要通过虚函数来指引子类对象，而如果不设置成虚函数的话，析构函数

数的。而由于在析构的过程中，先析构子类对象，后析构父类对象，因此需要通过虚函数来指引子类对象，而如果不设置成虚函数的话，析构函数是无法执行子类的析构函数的，从而可能导致内存的泄露。

53. static不能是虚函数
54. 只有类的成员函数才可以是虚函数
55. 通过基类指针访问派生类中新定义的成员：dynamic_cast
    a. B *q=dynamic_cast<B *>(p);
56. 纯虚函数？（在函数末尾加上符号“=0”）
    a. 只给出函数声明而没有实现的虚成员函数：virtual int f()=0;
57. 抽象类?
    a. 包含纯虚函数的类称为抽象类
58. 多继承注意：
    a. 基类的声明次序决定：
        i. 对基类构造函数/析构函数的调用次序
        ii. 对基类数据成员的存储安排
    b. eg：class C：public A，public B
        i. 构造函数的执行次序：A()，B()，C()
        ii. （A()和B()实际是在C()的成员初始化表中调用。）
59. 重复继承问题：D继承B、C，B，C均继承A，若要求D中只能有一个A类成员，则应把A类定义为B，C类的virtual虚基类
60. 虚基类的构造函数（由最新派生出的类的构造函数调用）
    a. 虚基类的构造函数优先非虚基类的构造函数执行
    b. 若同一层次中包含多个虚基类，这些虚基类的构造函数按它们说明的次序调用;
    c. 若虚基类由非虚基类派生而来，则仍先调用基类构造函数，再调用派生类的构造函数
61. 执行次序问题？参考课本P290！！这里不浪费时间打了。
62. 具有类属性的程序实体可以实现一种特殊的多态：参数化多态
63. 有时，编译程序无法根据调用时的实参类型来确定所调用的模板函数，这时，需要在程序中显式地实例化函数模板。例如：对于double x；int x；，则max<double>(x,m);或者 max<int>(x,m);可以显式实例化，或者再多定义一个max的重载函数
64. template <class T, int size> //size为一个int型的普通参数
    void f(T a){……}
    后面要使用f<int,10>(1)显式实例化函数模板的非类型参数
65. 不同类模板实例之间共享类模板中的静态成员吗?
    a. 类模板中的静态成员仅属于实例化后的类（模板类），不同类模板实例之间不共享类模板中的静态成员。

66. 以二进制方式输出文件：
    a. ios::binary
    b. out_file.write((char *)&x,sizeof(x)); //4个字节
67. 程序中为什么要显式关闭文件?
    a. 如果没有显式关闭文件，可能导致系统资源的浪费，并且在这个进程持续的时间内，别的进程就无法再次打开这个文件了。
68. bool strlonger(char *str1,char *str2){
        return strlen(str1)-strlen(str2)>0;
    }
    strlonger("abc","1234")；//abc长度>1234长度
69. 处理异常的基本手段
    a. 就地处理
    b. 异地处理 eg：try、throw、catch、异常的嵌套处理等等
70. 宏assert只有在宏名NDEBUG没有定义时才有效，这时，程序一般处于开发、测试阶段。程序开发结束提交时，应该让宏名NDEBUG有定义，然后重新编译程序，这样，assert就不再有效了。
71. 例如，对于一个多文档的Windows应用程序，
    a. 在程序开始运行时，首先创建应用程序对象；
    b. 由应用程序对象来创建主窗口对象；
    c. 在程序运行过程中，用户选择主窗口菜单项“文件|打开”，创建一个文档对象以及相应的子窗口对象。
72. 转移构造函数：当用一个临时对象或即将消亡的对象去初始化另一个对象时，如果对象类中有转移构造函数，则会去调用转移构造函数来对对象初始化；否则调用拷贝构造函数进行对象初始化
    a. 示例：A(A&& x)
73. 转移赋值操作符重载函数：当用于赋值的对象是一个临时的或即将消亡的对象时，如果对象类中有转移赋值操作符重载函数，则会去调用它来实现对象的赋值；否则调用普通的赋值操作符重载函数来实现对象的赋值（注意防止自身赋值）
    a. 示例：A& operator=(A&& x) //参数类型为右值引用：&&
    b. A& operator=(A&& x){ //参数类型为右值引用
            if (&x == this) return *this;
            delete []p; //归还老空间
            p = x.p; //使用参数对象的空间（资源转移）
            x.p = NULL; //使得参数对象不再拥有空间
            return *this;
        }
74. 拷贝构造函数?

1. A::A(const A& a)