C++简述题

什么是面向对象程序设计?

面向对象的程序设计（Object-Oriented Programming，OOP）是一种编程范式，它使用对象和类的概念来组织代码和数据。在面向对象的程序设计中，数据和操作数据的方法被组合在一起，形成了对象。这些对象可以相互协作，通过消息传递来完成任务。

面向对象的程序设计通常具有以下特征：

1. **封装（Encapsulation）**：
   - 封装是指将数据和操作数据的方法组合在一起，形成类（Class）。类将数据（属性）和方法（行为）封装在一起，隐藏了内部实现的细节，只暴露出对外部可见的接口。
2. **继承（Inheritance）**：
   - 继承允许一个类（子类）继承另一个类（父类）的属性和方法。子类可以扩展或修改父类的行为，并且可以通过继承实现代码的重用。
3. **多态（Polymorphism）**：
   - 多态允许不同类的对象对同一消息做出不同的响应。这意味着可以使用统一的接口来处理不同类的对象，而具体的操作会根据对象的实际类型而变化。

面向对象的程序设计有助于提高代码的可维护性、可扩展性和重用性。它能够更好地模拟现实世界中的问题，使得软件开发更加灵活和易于理解。

从概念上讲，抽象与封装有什么不同?

抽象是指该程序实体外部可观察到的行为，使用者不考虑该程序实体的内部是如何实现的。是将复杂的现实世界问题简化为程序设计中的模型或者接口。在抽象中，我们关注的是对象的行为和特征，而不关注具体的实现细节。
封装是指把该程序实体内部的具体实现细节对使用者隐藏起来，只对外提供一个接口。是将数据和操作捆绑在一起，形成一个相对独立的单元。在封装中，我们通过访问控制来限制对数据和操作的访问，只暴露必要的接口供外部使用。同时，封装可以保护数据的完整性和安全性，同时隐藏实现细节，降低了模块间的耦合度。
总的来说，抽象侧重于对问题的概括和简化，而封装则侧重于对数据和操作的封装和访问控制。

从影响软件开发效率和软件质量的因素的角度出发，解释面向对象程序设计带来的好处

模块化：组织和管理大型程序
软件复用：缩短开发周期
可维护性：由于OOP 的模块化特性，代码更易于维护和更新。当需要修改一个模块时，不会影响到其他模块，这降低了维护成本并提高了软件质量。
软件模型的自然度：缩小解题空间与问题空间之间的语义间隙，实现从问题到解决方案的自然过渡
继承性：通过继承机制，子类可以继承父类的属性和方法，这不仅减少了代码量，也提高了代码的可维护性。继承使得新的对象模型可以在现有模型的基础上进行构建，从而加快开发速度并提高效率。
多态性：OOP 的多态性允许同一操作应用于不同的对象，使得一个接口可以有多个实现方法。这增加了程序的灵活性和可扩展性，有助于减少错误并简化复杂的条件语句。

什么是对象？什么是类？二者之间的关系是怎样的?

对象是由数据及能对其实施的操作所构成的封装体，它构成了面向对象计算模型的基本计算单位。
类用于描述对象的特征，一个类描述了一组具有相同特征的对象。
对象属于值的范畴，而类属于类型的范畴。一个类刻画了一组具有相同特性的对象，它是对象的集合，而对象则是类的实例。类也是一种用于创建对象的模板，在静态程序中，只有类没有对象，当程序运行起来时，对象开始存在。

简述C++ 中this 指针的作用

类定义中的成员函数对该类的所有对象只有一个拷贝，因此对于每一个成员函数一般都有一个隐藏的形参this，当通过对象来调用类的成员函数时，编译程序会把相应对象的地址作为实参传给成员函数的隐藏的形参this，于是成员函数通过this就能知道对哪一个对象进行访问了。

举例说明什么情况下需要自定义析构函数？析构函数中归还的资源包括哪些？不包括哪些？

如果对象在创建后申请了一些资源，并且在对象消亡前没有归还这些资源，则应自定义析构函数以在对象消亡时归还对象申请的资源。
析构函数中归还的资源包括创建的局部对象，用new操作符动态分配的内存，用new[ ]动态分配的数组等。
不包括程序中的全局变量和静态成员变量等。

在哪些情况下会调用拷贝构造函数？解释为什么隐式拷贝构造函数可能导致运行错误？

（1）定义对象时 （2）把对象作为值参数传给函数时 （3）把对象作为值返回时
某些情况下隐式的拷贝构造函数将会使得s1和s2的成员指针str指向同一块内存区域，进而如果对一个对象（s1或s2）操作之后修改了这块空间的内容，则另一个对象将会受到影响。当对象s1和s2消亡时，将会分别去调用它们的析构函数，这会使得同一块内存区域将被归还两次，从而导致程序运行错误。当对象s1和s2中有一个消亡，另一个还没消亡时，则会出现使用已被归还的空间问题。

什么是常成员函数及静态成员，并说明它们在程序设计中的主要应用场景

常成员函数：
在定义一个成员函数时，可以给它加上一个const说明，表示它是一个获取对象状态的常成员函数。
主要应用场景：
1）在const成员函数定义的地方，告诉编译程序该成员函数不应该改变对象数据成员的值
2）在使用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
静态成员：
用于解决同一个类的对象共享数据的问题，分为静态数据成员和静态成员函数。
主要应用场景：
1）当通过一个变量改变静态数据成员的值，通过同类的其他对象可以看到这个修改，并且静态数据成员较为安全，它可以受到类的访问控制的保护
2）静态成员函数只能访问静态成员，既可以通过对象访问，也可以通过类名受限访问

为什么需要引入友元的概念，并描述如何在类中声明友元

为什么引入友元：
提高对象私有数据成员的访问效率，同时提高面向对象程序设计的灵活性。
如何在类中声明友元：
Class A
{ ……

```
friend void func(); //友元函数
friend class B; //友元类
friend void C::f(); //友元类成员函数，假定void f()是类C的成员函数
};
```

简述重载单目运算符++、--，前置和后置时的差别

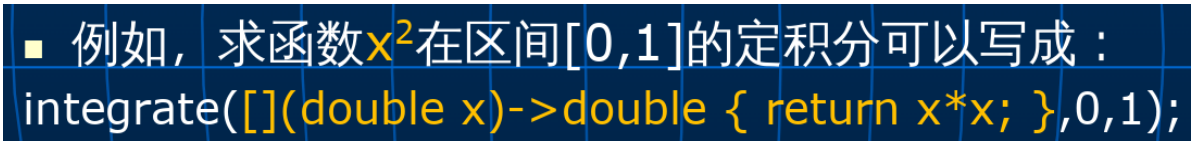
1. **前置版本**：
   - 前置版本的重载形式为 `++var` 和 `--var`。
   - 在前置版本中，操作符被应用于变量之前，然后返回变量的新值。
   - 例如，对于 `++var`，它会先使 `var` 的值加一，然后返回加一后的值。
2. **后置版本**：
   - 后置版本的重载形式为 `var++` 和 `var--`。
   - 在后置版本中，操作符被应用于变量之后，但它返回的是变量的旧值。
   - 例如，对于 `var++`，它会返回 `var` 的当前值，然后再将 `var` 的值加一。

简述为什么C++ 编程中有时候需要重载new 以及delete 操作符

重载new和delete运算符是为了提供对动态内存分配和释放过程的更精细控制，使得该类能以自己的方式来实现动态对象空间的分配和释放功能。

什么是λ 表达式？举一个例子。λ 表达式在编程中提供了什么便利？

λ 表达式是指匿名函数，它可以将函数定义和使用二者合一



便利：对于一些临时用一下的简单函数，不需要先给出这个函数的定义并为之取个名字，再通过名字去使用它，而是可以将函数定义和使用合二为一，提高编程的效率。

简述继承与封装的矛盾。如何缓解？

矛盾：在派生类中定义新的成员函数或对基类已有成员函数重定义时，往往需要直接访问基类的一些private成员（特别是private数据成员），否则新的功能无法实现，而类的private成员是不允许外界使用的（数据封装）。
缓解方法：通过protected访问控制

什么纯虚函数和抽象类？抽象类的作用？

纯虚函数：只给出函数声明而没给出实现（包括在类定义内部和外部）的虚成员函数。
抽象类：包含纯虚函数的类称为抽象类。
抽象类的作用：为派生类提供一个基本框架和一个公共的对外接口。

什么是虚函数？构造函数和析构函数是否可以是虚函数？请解释原因

虚函数是指加了关键词virtual的成员函数。
构造函数不能是虚函数，因为对象的构造过程发生在其生命周期的开始，而虚函数调用依赖于对象的类型信息，这通常在对象构造完成后才完全确定。
析构函数可以（往往）是虚函数，因为在多态情况下，可能会通过基类指针或引用删除派生类的对象。
如果析构函数不是虚的，那么只会调用基类的析构函数，而派生类特有的资源释放和清理代码将不会被执行。

请阐述C++ 中动态绑定和静态绑定的概念，并说明在什么情况下会发生动态绑定

静态绑定：一般情况下，在编译时刻根据对象的类型来决定采用哪一个消息处理函数，即采用静态绑定。
动态绑定：通常情况下，需要在运行时刻，根据func1（或func2）中x（或p）实际引用（或指向）的对象来决定是调用A::f还是B::f，即采用动态绑定。
何时发生动态绑定：当基类中的一个成员函数被定义为虚函数时。

在C++ 中，继承方式的作用是什么？public 继承方式有什么特殊之处?

继承方式的作用：影响该派生类的实例用户和该派生类对从基类继承来的成员的访问限制。
public继承方式的特殊之处：以public方式继承的派生类可看成基类的子类型，而子类型的作用体现在对基类对象所能实施的操作也能
作用于派生类对象，以及在需要基类对象的地方可以用派生类替代。

使用类聚合的方式和类组合的方式复用代码有什么不同? 什么情况下适合使用聚合? 什么情况下适合使用组合? 在编程时需要注意什么?

不同：
（1）在聚合关系中，被包含的对象与包含它的对象独立创建和消亡，被包含的对象可以脱离包含它的对象独立存在。而在组合关系中，被包含的对象随包含它的对象创建和消亡，被包含的对象不能脱离包含它的对象独立存在。
（2）在实现上，聚合类的成员对象一般是采用对象指针表示，用于指向被包含的成员对象，而被包含的成员对象是在外部创建，然后加入到聚合类对象中来的。而组合类的成员对象一般直接是对象，有时也可以采用对象指针表示，但不管是什么表示形式，成员对象一定是在组合类对象内部创建并随着组合类对象的消亡而消亡。
适合使用聚合的情况：当部分对象可以属于多个整体对象时、部分对象的生命周期与整体对象的生命周期无关时，等等。
适合使用组合的情况：整体对象拥有部分对象的生命周期，并且负责创建和销毁部分对象时、部分对象的生命周期依赖于整体对象的生命周期时，等等。
编程时的注意点：在基于聚合/组合的代码复用中，一个类对外只需一个接口：public。具有聚合/组合关系的两个类不具有子类型关系。

聚合/组合相比继承的代码复用有哪些优点? 能否仅仅通过前两者实现代码复用? 为什么?

优点：
（1）聚合/组合与封装则不存在继承与封装之间的矛盾。
（2）在基于聚合/组合的代码复用中，一个类对外只需一个接口：public。
仅仅通过组合/聚合实现代码复用是可能的，但不是最有效的方式，因为组合和聚合的设计初衷是为了表

示对象之间的结构关系，而不是为了共享代码。而且虽然通过组合或聚合可以在一定程度上实现代码复用，但这种方法通常比较间接，并且可能导致设计上的复杂性。

请简述printf、scanf 的缺陷以及cout、cin 的优势

printf、scanf 的缺陷：
类型不安全（不是强类型）：参数个数和类型不固定，编译时刻无法进行参数类型检查，会导致与类型相关的运行错误
cout、cin 的优势：
不需要专门指定数据的类型和个数，编译时刻根据数据本身来决定数据的类型和个数，这样可避免与类型和个数相关的错误！

程序的运行异常错误与程序的逻辑错误有什么不同?

程序的运行异常错误：
指程序设计对程序运行环境考虑不周而造成的程序运行错误。例如：
 对于“x/y”操作，给y输入了“零”。
 由内存空间不足导致的内存访问错误
 输入数据的数量超过存放它们的数组的大小（数组下标越界），导致程序运行错误。
 多任务环境可能导致的文件操作错误。
并且在程序运行环境正常的情况下，运行异常的错误是不会出现的。
程序的逻辑错误：
指程序设计不当造成程序没有完成预期的功能。例如：
 把两个数相加写成了相乘
 排序功能未能正确排序

请简述C++ 结构化异常处理机制（哪三种语句，分别什么作用）。如果不用这些，如何处理在构造函数中发现的异常?

try语句：
启动异常处理机制。格式为：try { <语句序列> }
throw语句：
在发现异常情况时生成异常对象。格式为：throw <表达式>
执行throw语句后，接在其后的语句将不再继续执行，而是转向异常处理（由某个catch语句给出）
catch语句：
捕获throw生成的异常对象并处理相应的异常。
格式为： catch (<类型> [<变量>]) { <语句序列> }
catch语句块要紧接在某个try语句的后面
如果不用这些，则可以通过进行基于断言的程序调试，如利用宏assert

请简述什么是事件驱动的程序设计?

其核心思想是程序的执行流程是由外部事件的发生而触发和驱动的。在事件驱动的编程模型中，程序通常会注册一系列的事件处理函数，这些函数会在特定的事件发生时被调用执行。
事件可以是来自用户交互的，比如鼠标点击、键盘输入、触摸屏操作等；也可以是来自系统或其他组件的通知，比如文件读写完成、网络数据到达、定时器到时等。当这些事件发生时，系统会调用事先注册好的相关事件处理函数，来响应并处理这些事件。

在C++中，迭代器的作用是什么?

迭代器使一种智能指针，指向容器中的元素，用于对容器中元素的遍历和访问

MFC提供的“文档-视”结构为程序设计带来什么好处?

它作为一种应用程序框架（简称应用框架），是一种通用的、可复用的应用程序结构，该结构规定了程序应包含哪些组件以及这些组件之间的关系，它封装了程序处理流程的控制逻辑。

通过复用应用框架，使得应用的开发速度更快、质量更高、成本更低。

C++面向对象程序中存在下面三种多态

1）对象类型的多态

2）对象标识的多态

3）消息的多态