**static 和 const分别怎么用，类里面的static和const可以同时修饰成员函数吗？**

static的作用：

对变量：

1. 局部变量：
2. 内存中的位置：静态存储区
3. 初始化：未经初始化的全局静态变量会自动初始化为0
4. 作用域：局部作用域。

当static用来修饰局部变量的时候，它就改变了局部变量的存储位置及其生命周期，当未改变其作用域。

1. 全局变量：
2. 内存位置：静态存储区
3. 初始化：
4. 作用域：从定义开始之处到文件结尾。

当static修饰全局变量，并未改变其存储位置及生命周期，而是改变了其作用域，使当前文件的源文件无法访问该变量，好处如下：1）不会被其他文件所访问，修改；2）其他文件使用相同名字的变量，不会发生冲突。对全局函数也是有隐藏作用。

对类中：

1. 成员变量

用static修饰类的数据成员实际使其称为类的全局变量，会被类的所有对象共享，包括派生类的对象。因此，static成员必须在类外进行初始化，而不能在构造函数内进行初始化，不过也可以用const修饰static数据成员在类内初始化。

1. 成员函数
2. 用static修饰成员函数，使这个类只存在这一份函数，所有对象共享该函数，不含有this指针
3. 静态成员可以独立访问，也就是说，无须创建任何对象实例就可以访问。当static成员函数在类外定义时不需要加static修饰符。
4. 静态成员函数的实现中不能直接引用类中说明的非静态成员，可以引用类中说明的静态成员。因为静态成员函数不含有this指针。

**不可以同时用const和static修饰成员函数**：

C++编译器在实现const的成员函数的时候为了确保该函数不能确保该函数修改类的实例的状态，会在函数添加一个隐式的参数const this\*。当当一个成员为static的时候，该函数是没有this指针的。也就是说此时const用法和static是冲突的。

**const的作用**：

1. 限定变量不可修改
2. 限定成员函数不可以修改任何数据成员。
3. 修饰指针
4. 修饰函数返回值及函数的参数

**多态和非多态的区别**：多态与非多态的实质区别就是函数地址是早绑定还是晚绑定。如果函数的调用，在编译器编译期间就可以确定函数的调用地址，并生产代码，是静态的，就是说地址是早绑定的。而如果函数调用的地址不能在编译器期间确定，需要在运行时才确定，这就属于晚绑定。

**Overload(重载)：在C++程序中，可以将语义、功能相似的几个函数用同一个名字表示，但参数或返回值不同（包括类型、顺序不同），即函数重载。**

（1）相同的范围（在同一个类中）；

（2）函数名字相同；

（3）参数不同；

（4）virtual 关键字可有可无。

**Override(覆盖)**：**是指派生类函数覆盖基类函数，特征是：**

（1）不同的范围（分别位于派生类与基类）；

（2）函数名字相同；

（3）参数相同；

（4）基类函数必须有virtual 关键字。

**注：重写基类虚函数的时候，会自动转换这个函数为virtual函数，不管有没有加virtual，因此重写的时候不加virtual也是可以的，不过为了易读性，还是加上比较好。**

**Overwrite(重写)：隐藏，是指派生类的函数屏蔽了与其同名的基类函数，规则如下：**

（1）如果派生类的函数与基类的函数同名，但是参数不同。此时，不论有无virtual关键字，基类的函数将被隐藏（注意别与重载混淆）。

（2）如果派生类的函数与基类的函数同名，并且参数也相同，但是基类函数没有virtual关键字。此时，基类的函数被隐藏（注意别与覆盖混淆）。

**纯虚函数**：使派生类仅仅只是继承函数的接口。纯虚函数不能在基类中实现，编译器要求在派生类中必须 予以重写以实现多态性。拥有纯虚函数的类为抽象类，不能生成对象。

**多重继承**：派生类继承多个基类，派生类为每个基类（显式或隐式）指定了访问级别。

构造：

1. 派生类对象包含每个基类的基类对象。
2. 派生类构造函数初始化所有基类，派生类只能初始化自己的基类，并不需要考虑基类的基类怎么初始化
3. 多重继承时，基类构造函数按照基类构造函数在类派生列表中的出现次序调用。

**深拷贝与浅拷贝：**

**浅拷贝：**默认的复制构造函数只是完成了对象之间的位拷贝，也就是把对象里的值完全复制给另一个对象，如A=B。这时，如果B中有一个成员变量指针已经申请了内存，那A中的那个成员变量也指向同一块内存。

　　　　这就出现了问题：当B把内存释放了（如：析构），这时A内的指针就是野指针了，出现运行错误。

**深拷贝**：自定义复制构造函数需要注意，对象之间发生复制，资源重新分配，即A有5个空间，B也应该有5个空间，而不是指向A的5个空间。

各个排序算法的时间复杂度和稳定性，快排的原理。

**插入排序**：每次将一个待排序的数据，跟前面已经有序的数字——比较找到合适的位置，插入到序列中，直到全部数据插入完成。

**冒泡排序**：通过交换使相邻的两个数变成小数在前大数在后，这样每次遍历之后，最大的数就沉到最后面。重复N次即使数组有序。

**快速排序**：先从数列中取出一个 数作为基准数，将比这个数大的数全部放到它的右边，小于或等于它的数全部放到它右边。在对左右区间重复第二步，直到个区间只有一个数。

**选择排序**：数组分成有序区和无序区，然后每次从无序区选一个最小的元素直接放到有序区的最后，知道整个数组变成有序。

**堆排序**：将待排序序列造成一个大顶堆，此时，整个序列的最大值就是堆顶的根结点。将其与末尾元素进行交换，此时末尾就为最大值。然后将剩余n-1个元素重新构造成一个堆，这样会得到n个元素的次小值。如此反复便能得到一个有序序列了。

**归并排序**：采用分治的方法将两个有序的序列合并一个有序的序列。采用递归的方法。

桶排序：

map和set的原理：

**TCP的三次握手过程**

**第一次握手**：建立连接时，客户端发送syn包到服务端，等待服务端确认；syn：同步序列编号

**第二次握手**：服务器收到syn后，必须确认客户的syn，同时自己自己发送一个syn包。

**第三次握手**：客户端接收服务端的syn和ack包之后，发送确认包ack

**TCP四次挥手过程**

**第一次挥手**：客户端发送fin包，用来关闭client到server的数据传输，客户端进入等待fin终止确认信号。

**第二次挥手**：服务端收到fin包之后，，发送确认包ack给客户端，进入关闭等待状态。

**第三次挥手**：服务端发送一个fin，用来关闭服务端和客户端的数据传输

**第四次挥手**：客户端收到fin之后，发送确认信号，服务端进入closed状态。

**TCP三次握手和四次挥手的原因**：

建立三次握手主要是因为A发送了再一次的确认，那么A为什么会再确认一次呢，主要是为了防止已失效的连接请求报文段又突然传送给B，从而产生了错误。

所谓“已失效的连接请求报文”是这样产生的，正常情况下，A发出连接请求，但是因为连接报文请求丢失而未收到确认，于是A再重传一次连接请求，后来收到了请求，并收到了确认，建立了连接，数据传输完毕后，就释放链接，A共发送了两次连接请求报文段，其中第一个丢失，第二个到达了B，没有“已失效的连接请求报文段”，但是还有异常情况下，A发送的请求报文连接段并没有丢失，而是在某个网络节点滞留较长时间，以致延误到请求释放后的某个时间到达B，本来是一个早已失效的报文段，但是B收到了此失效连接请求报文段后，就误以为A又重新发送的连接请求报文段，并发送确认报文段给A，同意建立连接，如果没有三次握手，那么B发送确认后，连接就建立了，而此时A没有发送建立连接的请求报文段，于是不理会B的确认，也不会给B发送数据，而B却一直等待A发送数据，因此B的许多资源就浪费了，采用三次握手的方式就可以防止这种事情发生，例如刚刚，A不理会B，就不会给B发送确认，B收不到A的确认，就知道A不要求建立连接，就不会白白浪费资源。

函数调用和系统调用的区别：

**系统调用：**所谓系统调用就是用户在程序中调用操作系统所提供的一个子功能，也就是系统API，系统调用可以被看做特殊的公共子程序。系统中的各种共享资源都由操作系统统一掌管，因此在用户程序中，凡是与资源有关的操作（如存储分配、进行I/O传输及管理文件等），都必须通过系统调用方式向操作系统提出服务请求，并由操作系统代为完成。

**这些系统调用按照功能大致可以分为以下几类：**

设备管理：完成设备的请求或释放，以及设备启动等功能。

文件管理：完成文件的读、写、创建及删除等功能

进程控制：完成进程的创建、撤销、阻塞、及唤醒的功能

进程通信：完成进程之间的消息传递或信号的传递

内存管理：完成内存的分配、回收以及获取作业占用内存区大小及始址等功能。

**系统调用的过程：**首先将API函数参数压到栈上，然后将函数内调用系统调用的代码放入寄存器，通过陷入中断，进入内核将控制权交给操作系统，操作系统获得控制后，将系统调用代码拿出来，跟操作系统维护的一张系统调用表做比较，已找到该系统调用程序体的内存地址，接着访问该地址，执行系统调用。执行完毕后，返回用户程序。

**库函数**：系统调用所提供给用户的是直接而纯粹的高级服务，如果想要更人性化，具有更符合特定情况的功能，那么就要我们用户自己来定义，因此就衍生了库函数，它把部分系统调用包装起来，一方面把系统调用抽象了，一方面方便了用户级的调用。系统调用是运行于内核状态；而库函数由用户调用，运行于用户态。

**系统调用**：是操作系统为用户态运行的进程和硬件设备进行交互提供的一组接口，即就是设置在应用程序和硬件设别之间的一个接口层。

**进程**：是并发执行的程序执行过程分配和管理资源的基本单位。进程有五种状态：初始态，就绪，等待，运行，终止

**线程：**在网络或多用户环境下，一个服务器通常需要接收大量且不确定数量用户的并发请求，为每个请求创建一个进程显然是行不通的，无论是从系统资源开销方面或是响应用户请求的效率方面来看。

**指针数组和数组指针的区别**：指针数组是存放指针的数组；而数组指针是指向数组的指针。

**在函数内定义一个字符数组，用gets函数输入字符串的时候，如果输入越界，为什么程序会崩溃？**

答：因为gets无法截断数组越界部分，会将所有输入都写入内存，这样越界部分就可能覆盖其他内容，造成程序崩溃。

C++内存分区：

1. 栈区：由编译器自动分配释放，存放函数的参数值，局部变量的值等。其操作方式类似与数据结构中的栈。
2. 堆区：一般有程序员分配释放，若程序员不释放，程序结束时可能由OS回收。
3. 全局区：全局变量和静态变量的存储是放在一起的，初始化全局变量和静态变量在一块区域，未初始化的全局变量和未初始化的静态变量在响铃的另一块区域。程序结束后由系统释放。
4. 文字常量区：常量字符串放在这里。
5. 程序代码区：存放函数体的二级制代码。

**C和C++有什么不同?**

从机制上：c是面向过程的（但c也可以编写面向对象的程序）；c++是面向对象的，提供了类。但是，c++编写面向对象的程序比c容易

从适用的方向：c适合要求代码体积小的，效率高的场合，如嵌入式；c++适合更上层的，复杂的； llinux核心大部分是c写的，因为它是系统软件，效率要求极高。

C语言是结构化编程语言，C++是面向对象编程语言。

C++侧重于对象而不是过程，侧重于类的设计而不是逻辑的设计。

sizeof的概念：

sizeof是C语言的一种单目操作符号，它并不是函数。sizeof操作符以字节的形式给出了其操作数的存储大小。操作数可以是一个表达式或括在括号内的类型名。操作数的存储大小由操作数的类型决定。

sizeof的使用方法：sizeof操作符不能用于函数类型，不完全类型或位字段。不完全类型指具有位置存储大小的数据类型，如未知存储大小的数组类型。

sizeof的结果：

1. 若操作数具有类型char，unsigned char或signed char，其结果等于1.
2. int，unsigned int，short int，unsigned short等基本类型大小依赖于实现。
3. 当操作数是指针时，sizeof依赖于编译器。
4. 当操作数具有数组类型时，其结果是数组的总字节数。
5. 联合类型操作数的sizeof是其最大字节成员的字节数。
6. 结构体类型是对象的总字节数，注意编译器会考虑对齐问题时，在结构中插入空位以控制各成员对象的地址对齐。
7. 如果操作数是函数中的数组形参或函数类型的形参，sizeof给出其指针的大小。

由于操作数的字节数在实现时可能出现变化，建议在涉及到操作数字节大小时用sizeof来代替常量计算。

**封装**：实现面向对象程序设计的第一步，封装就是将数据或函数等集合在一个个的单元中。封装的意义在于保护或者防止代码被我们无意破坏。

**继承**：主要实现重用代码，节省开发时间。

**多态**：相同的操作或函数、过程可作用于多种类型的对象上并获得不同的结果。不同的对象，收到同一消息产生不同的结果，这种现象称为多态。

**常见的STL容器有哪些，算法有哪些。**

**容器：**存放数据的地方。

**数组和链表的区别：**

**从逻辑结构来看：**

数组必须事先定义固定的长度（元素个数），不能适应数据动态地增减的情况，即数组的大小一旦定义就不能改变。当数据增加时，可能超出原先 定义的元素个数；当数据减少时，造成内存浪费；链表动态地进行存储分配，可以适应数据动态地增减的情况，且可以方便地插入、删除数据项。（数组中插入、删 除数据项时，需要移动其它数据项）。

**从内存存储来看：**

（静态)数组从栈中分配空间（用NEW创建的在堆中）, 对于程序员方便快速,但是自由度小；链表从堆中分配空间, 自由度大但是申请管理比较麻烦.

**从访问方式来看：**

数组在内存中是连续存储的，因此，可以利用下标索引进行随机访问；链表是链式存储结构，在访问元素的时候只能通过线性的方式由前到后顺序访问，所以访问效率比数组要低。

**引用作为参数的优点**：

1. 传递引用给函数与传递指针的效果是一样的。
2. 使用引用传递参数，在内存中没有实参副本，直接对实参操作。返回一个局部变量的引用是不可取。因为随着该局部变量生存期的结束，相应的引用也会失效。
3. 使用指针作为函数的参数虽然也能达到与使用引用的效果，但是，在被调函数中同样要给形参分配存储单元，且需要重复使用指针变量名的形式进行运算，这很容易产生错误且程序的阅读性很差；另一方面，在主调函数的调用点出，必须用变量的地址作为实参。而引用更容易使用，更清晰。

**内存的类别**：栈、堆、全局区、程序代码区、文本区

**堆栈溢出的原因**：数组越界、没有回收内存、深层递归调用

**避免内存泄漏**：动态分配的内存需要手动释放。

**const与define相比，有何优点**：const 常量有数据类型，而宏常量没有数据类型。编译器可以对前者进行类型安全检查。而对后者只进行字符替换，没有类型安全检查，并且在字符替换可能会产生意料不到的错误。

**new和malloc的算法**：malloc和free是c语言的标准库函数, new/delete是c++的运算符.都可用来申请动态内存释放.

由于malloc/free是库函数,不是运算符,因此不能将执行构造函数和析构函数的任务强加于malloc/free. c++需要一个能够完成动态内存分配和初始化工作的运算符new. 主要, new/delete不是库函数.

c++可以调用c函数, 而c程序只能用malloc/free管理动态内存.

new可以认为是malloc加构造函数的执行. new出来的指针都是直接带类型信息的, 而malloc返回的都是void指针.

**如何引用一个已经定义过的全局变量**：引用头文件和extern关键字。 如果采用引用头文件， 若变量写错了，则在编译期间便会出错。 如果用extern则在链接阶段报错。

**频繁出现的短小函数， 在c和c++中分别如何实现**：c中使用宏定义， c++中使用inline内联函数

**c++函数值传递的方式**：值传递， 指针传递， 引用传递

**.h头文件中的ifndef/define/endif 的作用**：防止该头文件被重复引用。

**struct 和 class 的区别：**truct 的成员默认是公有的，而类的成员默认是私有的。struct 和 class 在其他方面是功能相当的。

**new和delete的实现原理**：

**如何判断两个浮点数是否相等**：由于浮点数的表示是不精确的，所以不能直接比较两个数是否完全相等。一般都是在允许的某个范围内认为某个个浮点数相等

**互斥锁和信号量的区别**：

1. 互斥量用于线程的互斥，信号线用于线程的同步。

互斥：是指某一资源同时只允许一个访问者对其进行访问，具有唯一性和排他性。但互斥无法限制访问者对资源的访问顺序，即访问是无序的。

同步：在互斥的基础上，通过其他机制实现访问者对资源的有序访问。在大多数情况下，同步已经实现了互斥，特别是写入资源的情况下必定是互斥的。少数情况是允许多个访问者同时访问资源

1. 互斥量只能是0/1，信号值可以为非负整数。

一个互斥量只能用于一个资源的互斥访问，它不能实现多个资源的多线程互斥问题。信号量可以实现多个同类资源线程互斥和同步。当信号量为单值信号时，也可以完成一个资源的互斥访问。

1. 互斥量的加锁和解锁必须有同一线程分别对应使用，信号量可以由一个线程释放，另一个线程得到。
2. 互斥量的线程阻塞时通过忙等待的方式来进入临界区。信号量的线程或进程阻塞时是线程进入等待队列进入等待状态。

**互斥锁弊端**：互斥锁会发生死锁。死锁原因，线程A进入临界区R1，线程B想进入临界区R1而阻塞。线程A进入R1后想进入另一个临界区R2，而R2被线程B占有而阻塞，造成死锁。

**条件变量**：条件变量减少线程的忙等待。加入线程A占用资源R，线程B申请资源R阻塞进入睡眠，当线程A释放资源R后，线程A使用条件变量唤醒线程B。