# new和malloc的区别

**a.属性**

　　new/delete是C++关键字，需要编译器支持。malloc/free是库函数，需要头文件支持c。

**b.参数**

　　使用new操作符申请内存分配时无须指定内存块的大小，编译器会根据类型信息自行计算。而malloc则需要显式地指出所需内存的尺寸。

**c.返回类型**

　　new操作符内存分配成功时，返回的是对象类型的指针，类型严格与对象匹配，无须进行类型转换，故new是符合类型安全性的操作符。而malloc内存分配成功则是返回void \* ，需要通过强制类型转换将void\*指针转换成我们需要的类型。

**e. 分配失败**

　　new内存分配失败时，会抛出bac\_alloc异常。malloc分配内存失败时返回NULL。

**f.自定义类型**

  new会先调用operator new函数，申请足够的内存（通常底层使用malloc实现）。然后调用类型的构造函数，初始化成员变量，最后返回自定义类型指针。delete先调用析构函数，然后调用operator delete函数释放内存（通常底层使用free实现）。

  malloc/free是库函数，只能动态的申请和释放内存，无法强制要求其做自定义类型对象构造和析构工作。

**g.重载**

　　C++允许重载new/delete操作符，特别的，布局new的就不需要为对象分配内存，而是指定了一个地址作为内存起始区域，new在这段内存上为对象调用构造函数完成初始化工作，并返回此地址。而malloc不允许重载。

**h.内存区域**

new操作符从自由存储区（free store）上为对象动态分配内存空间，而malloc函数从堆上动态分配内存。自由存储区是C++基于new操作符的一个抽象概念，凡是通过new操作符进行内存申请，该内存即为自由存储区。而堆是操作系统中的术语，是操作系统所维护的一块特殊内存，用于程序的内存动态分配，C语言使用malloc从堆上分配内存，使用free释放已分配的对应内存。自由存储区不等于堆，如上所述，布局new就可以不位于堆中。

# IPV6中规定的长度为128位

# 在单链表中设置头节点的作用是什么？

在单向链表中，在单链表中设置头节点的作用是(简化插入、删除操作)，除首节点外，任何一个节点的存储位置由(前驱节点的后继指针)表示。

# 计算机网络采用层级结构的模型有什么好处?

1.各层之间是独立的

2.灵活性好

3.结构上可分割开

4.易于实现和维护

5.能促进标准化工作

# 观察ping返回包的TTL值,可以了解到什么?

TTL是数据包生存周期,根据减少的数可以判断经过了多少中间设备；

ping返回的TTL值可以判断是系统还是路由；

# ICANN规定的私有地址为？

ICANN规定一部分IP地址为私有地址：

A类：10.0.0.0/8

B类：172.16.0.0-172.31.0.0/16

C类：192.168.0.0-192.168.255.0/24

# const和#define的区别？

* const 定义的常数是变量 也带类型， #define 定义的只是个常数 不带类型；
* define是在编译的预处理阶段起作用，而const是在 编译、运行的时候起作用；
* define只是简单的字符串替换，没有类型检查。而const有对应的数据类型，是要进行判断的，可以避免一些低级的错误；

正因为define只是简单的字符串替换会导致边界效应，例如：



* const常量可以进行调试的，define是不能进行调试的，因为在预编译阶段就已经替换掉了；
* const不足的地方，是与生俱来的，const不能重定义，而#define可以通过#undef取消某个符号的定义，再重新定义；
* define可以用来防止头文件重复引用，而const不能；

# 指针和引用的联系与区别

★ 相同点：

1. 都是地址的概念；

指针指向一块内存，它的内容是所指内存的地址；引用是某块内存的别名。

★ 区别：

1. 指针是一个实体，而引用仅是个别名；

2. 引用使用时无需解引用(\*)，指针需要解引用；

3. 引用只能在定义时被初始化一次，之后不可变；指针可变；

4. 引用没有 const，指针有 const；

5. 引用不能为空，指针可以为空；

6. “sizeof 引用”得到的是所指向的变量(对象)的大小，而“sizeof 指针”得到的是指针本身(所指向的变量或对象的地址)的大小；

7. 指针和引用的自增(++)运算意义不一样；

8.从内存分配上看：程序为指针变量分配内存区域，而引用不需要分配内存区域。

# 使用动态链接库的好处

1. 可以采用多种编程语言来编写。
2. 增强产品的功能。
3. 提供二次开发的平台。
4. 简化项目管理(降低程序的耦合性，改动一个功能，只需编译当前模块即可)。
5. 可以节省磁盘空间和内存。
6. 有助于资源的共享。
7. 有助于实现应用程序的本地化。

# 进程和线程的区别

1. 进程是资源分配的最小单位，线程是程序执行的最小单位。
2. 进程有自己的独立地址空间，每启动一个进程，系统就会为它分配地址空间，建立数据表来维护代码段、堆栈段和数据段，这种操作非常昂贵。而线程是共享进程中的数据的，使用相同的地址空间，因此CPU切换一个线程的花费远比进程要小很多，同时创建一个线程的开销也比进程要小很多。
3. 线程之间的通信更方便，同一进程下的线程共享全局变量、静态变量等数据，而进程之间的通信需要以通信的方式（IPC)进行。不过如何处理好同步与互斥是编写多线程程序的难点。
4. 但是多进程程序更健壮，多线程程序只要有一个线程死掉，整个进程也死掉了，而一个进程死掉并不会对另外一个进程造成影响，因为进程有自己独立的地址空间。

# 封装、继承和多态

**代码重用：**

封装：隐藏实现细节，使得代码模块化

继承：扩展已经存在的代码模块（类）

**接口重用：**

多态

实现多态，有二种方式，覆盖，重载。覆盖：**是指子类重新定义父类的虚函数的做法**。重载：**是指允许存在多个同名函数，而这些函数的参数表不同**（或许参数个数不同，或许参数类型不同，或许两者都不同）。