## **● 说一下static关键字的作用**

**1、static在C语言中的作用**

a、static修饰变量：

初始化

static修饰变量的存储位置：静态存储区

修饰局部变量：延长该变量的生命周期、具有记忆功能：会记录上次函数运

行结束之后的内容

修饰全局变量：改变该变量的链接属性----只能在当前文件中进行访问，静

态存储区，在整个程序运行期间一直存在，未经初始化的全

局静态变量会被自动初始化为0；

b、static修饰函数：表明该函数只能在当前文件中进行访问，不能被其他文

件所用；

（warning：不要再头文件中声明static的全局函数，不要在cpp内声明非static的全局函数，如果你要在多个cpp中复用该函数，就把它的声明提到头文件里去，否则cpp内部声明需加上static修饰；）

**2、static在C++中的作用**

a、static修饰变量：

普通、全局与C语言类似

类成员变量-----静态成员变量&普通成员变量的区别

在类中，静态成员可以实现多个对象之间的数据共享，并且使用静态数据成员还不会破坏隐藏的原则，即保证了安全性。因此，静态成员是类的所有对象中共享的成员，而不是某个对象的成员。

特性：

1>静态成员变量是类的属性，不属于某个具体的对象

2>存储位置：静态区---不会存在于对象中，不会影响sizeof的结果

3>访问：类名::静态成员变量名字||对象**.**静态成员变量名字

4>初始化：不能放在构造函数初始化列表中初始化，在类中只是声明，必须在类外进行初始化

b、static修饰函数：普通函数----与C语言中作用相似

类成员变量-----静态成员函数&普通成员函数的区别

特性：

1>没有this指针

2>不能直接访问非静态成员函数

3>不能直接调用非静态成员函数

4>不能用const修饰、virtual

5>调用方式：类名::静态成员函数名字||对象**.**静态成员函数名字

**●面向对象和面向过程的区别**

面向过程（自顶向下的编程）：就是分析出解决问题所需要的步骤，然后用函数把这些步骤一步步实现，使用时一个一个调用就可以了。

面向对象（高度事务抽象化）：是把构成问题事务分解成各个对象，建立对象的目的不是为了完成一个步骤，而是为了描述某个事务在整个解决问题的步骤中的行为。

**C++和C的区别**

设计思想上：

C++是面向对象的语言，而C是面向过程的结构化编程语言

语法上：

C++具有封装、继承和多态三种特性

C++相比C，增加多许多类型安全的功能，比如强制类型转换、

C++支持范式编程，比如模板类、函数模板等

## **● 请说一下C/C++ 中指针和引用的区别？**

1、**从概念**：别名----与其引用的实体共用同一块内存空间

2、 **从底层实现**：编译器在底层实际是将引用转换为指针类进行处理，因此引用在底层实际也有自己的内存空间（从底层实现上看：引用和指针是没有区别的）

**从应用或概念层面**：

>>引用在定义期间必须初始化，指针没要求

>>引用在初始化时引用一个实体后，不能在引用其他实体

>>在sizeof中含义不同：引用结果为引用类型的大小，但指针始终是在地址空间所占字节个数(4个字节)

>>引用自加即引用的实体增加1，指针自加即指针向后偏移一个类型的大小

>>有多级指针，但无多级引用

>>访问实体方式不同，指针需要显示解引用，引用编译器自己处理

>>引用比指针使用起来相对更安全

>>没有NULL引用，但有NULL指针

**● 请你说一下你理解的c++中的smart pointer四个智能指针： shared\_ptr,unique\_ptr,weak\_ptr,auto\_ptr**

1. 请你介绍一下C++中的智能指针？

智能指针主要用于管理在堆上分配的内存，它将普通的指针封装为一个栈对象。当栈对象的生存周期结束后，会在析构函数中释放掉申请的内存，从而防止内存泄漏。C++ 11中最常用的智能指针类型为shared\_ptr,它采用引用计数的方法，记录当前内存资源被多少个智能指针引用。该引用计数的内存在堆上分配。当新增一个时引用计数加1，当过期时引用计数减一。只有引用计数为0时，智能指针才会自动释放引用的内存资源。对shared\_ptr进行初始化时不能将一个普通指针直接赋值给智能指针，因为一个是指针，一个是类。可以通过make\_shared函数或者通过构造函数传入普通指针。并可以通过get函数获得普通指针。

1. 为什么需要有智能指针?

智能指针的作用是管理一个指针，因为存在以下这种情况：申请的空间在函数结束时忘记释放，造成内存泄漏。使用智能指针可以很大程度上的避免这个问题，因为智能指针就是一个类，当超出了类的作用域时，类会自动调用析构函数，析构函数会自动释放资源。所以智能指针的作用原理就是在函数结束时自动释放内存空间，不需要手动释放内存空间。

指针:灵活 缺陷:用户动态申请资源- - -通过指针接收 手动释放--- >容易被遗忘掉||代码丑陋

能否让动态自动去进行释放

2.什么是RAII?

资源获取即初始化---在C++中，创建对象或销毁对象时,编译器会自动调用构造函数完成对象初始化,会自动调用析构函数完成对象中资源的清理工作

可以巧妙借助构造和析构

3.如果让你设计智能指针，都需要实现哪些方法?

智能指针主要职责:帮助用户管理资源，并在合适的时机释放资源

智能指针:

1>.包含一个指针--->T\*(可以通过模板方式)

2>. RAII:构造中将用户的资源进行接收，在析构中将资源释放掉

3>.让智能指针的对象具有指针类似的行为---operator\*() / operator->()

4>.解决浅拷贝问题:用户必须显式实现拷贝构造函数以及赋值运算符重载

4.各个智能指针的实现原理以及区别

C++98: auto\_ptr

**<一>auto\_ ptr: RAII + operator\*0/operator->() +解决浅拷贝的方式**

解决浅拷贝方式:

1.资源的转移auto\_ptr<int> ap1(new int); auto\_ptr<int> ap2(ap1); ap1将其管理的资源直接转移ap2,然后ap1与资源完全断---->效果:可能让一个资源只释放一次， 那个对象最终拥有资源，那个对象进行释放；缺陷: ap1和ap2不能同时拥有资源

2.资源管理权的转移(只转移资源释放的权利，而需要对资源断开关联)

在类中增加bool类型的成员变量:

owner-->true: 表示当前对象在其声明周期结束时，必须要释放资源

false:表示当前对象不能释放资源

auto\_ptr<int> ap1(new int); auto\_ ptr<int> ap2(ap1);

ap1和ap2共享同一份资源，ap1没有与资源断开联系，解决方式1中的缺陷:

新的缺陷:可能会造成野指针（野指针就是指向一个已删除的对象或者未申请访问受限内存区域的指针）

**<二>unique\_ ptr**: 一个资源只能被一个unique\_ ptr类型的对象进行管理，禁止多个对象之间共享资源

实现原理: RAII + operator\*(/operator->() +防拷贝

防拷贝:

C++98:只需要将拷贝构造函数以及赋值运算符重载只声明不定义&必须将其访问限定符设置为private

C++11:只需要在拷贝构造函数和赋值运算符重载= delete (说明: =delete编译器不要生成默认的拷贝构造以及赋值运算符重载)

unique\_ ptr可以正常使用，唯一不足之处: 不能共享资源

**<三>shared\_ ptr**:因为unique\_ptr不能满足所有的场景，因此提供shared\_ptr

实现原理: RAII + operator\*()/operator->() +解决浅拷贝方式

解决浅拷贝方式:采用引用计数的方式解----引用计数:资源被共享的对象的个数

引用计数的方式:

1. 普通类型的整形成员变----不可以:计数需要被多个对象共享，每个对象中都包含一个计数， 如果一个对象将计数更改，并不会影响其他的对象

2.静态整形成员变----不可以:静态成员变量是所有类对象共享，引用计数并不是所有对象共享，引用计数记录的是使用资源的对象个数，引用计数应该是和资源挂钩，有多少分资源就应该有多少个计数,共享同一份资源的对象之间共享同一份计数

3.整形的指针

优点:大部分场景都可以处理

缺陷:可能会存在**循环引用**问题

1. 什么是循环引用

-1、node1和node2两个智能指针对象指向两个节点，引用计数变成1，我们不需要手动delete。

- 2、node1的\_next指向node2，node2的\_prev指向node1，引用计数变成2。

- 3、node1和node2析构，引用计数减到1，但是\_next还指向下一个节点。但是\_prev还指向上一个节点。

- 4、也就是说\_next析构了，node2就释放了。

- 5、也就是说\_prev析构了，node1就释放了。

- 6、但是\_next属于node的成员，node1释放了，\_next才会析构，而node1由\_prev管理，\_prev属于node2成员，所以这就叫循环引用，谁也不会释放。

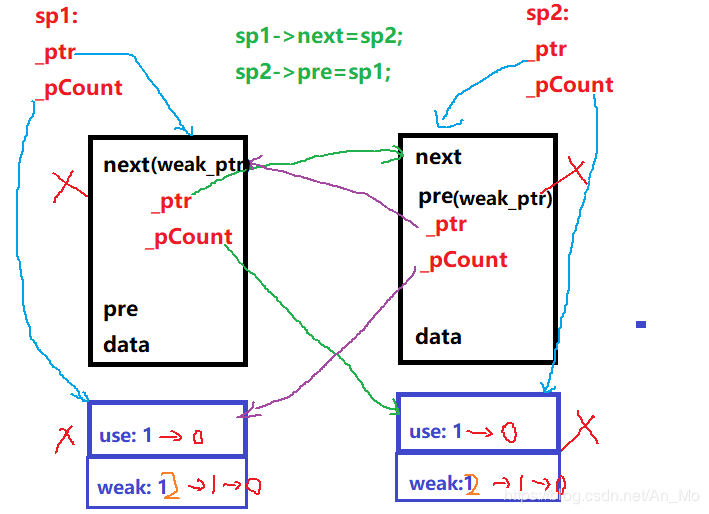
2.循环引用会造成什么后果

资源没有释放--到引起资源泄漏

1. 循环引用如何解决

/////////////////////////////////////////////////////////////////////

****解决方式：在引用计数的场景下，把节点中的\_pre和\_next改成weak\_ptr就可以了****  
****weak\_ptr****：  
****1）实现原理****：RAII+具有指针类似操作+引用计数  
****2）作用****：配合shared\_ptr，解决其循环引用问题  
****3）注意****：****weak\_ptr对象不能独立管理资源****



/////////////////////////////////////////////////////////////////////

weak\_ ptr:作用-配合shared \_ptr使用，帮助shared\_ptr解决循环引用的问题

weak\_ ptr不能独立的管理资源

实现原理:与shared\_ptr的实现原理一致，都采用引用计数shared\_ptr和weak\_ptr采用的不是同一个计数

**定制删除器:**

为什么要定制删除器:资源的种类比较多，不同类型的资源应该选择合适的方式进行释放，因此智能指针的析构函数中不能将资源的释放方式写死，不同类型的资源就会采用同一种方式进行释放而引起代码崩溃

定制方式:在实现智能指针时，给用户预留选择释放方式的接口

>> 1.多给一个模板类型的参数---上课讲

>> 2.可以通过参数

具体实现: 1. 函数指针2. 仿函数3. lambda表达式

5.智能指针和STL中容器的区别

C++11提供的智能指针只能管理单个独享的空间，不能管理一段连续的空间

为什么: STL中已经存在vector

智能指针:管理的资源是外部用户申请的

STL容器:资源是容器自己维护

6.面试官可能会让学生模拟实现一个智能指针

建议:

1.模拟实现unique\_ptr,因为:实现简单

2.最好可以定制删除器---参考: shared\_ptr定制删除器方式

## **● 请回答一下数组和指针的区别**

|  |  |
| --- | --- |
| 指针 | 数组 |
| 保存数据的地址 | 保存数据 |
| 间接访问数据，首先获得指针的内容，然后将其作为地址，从该地址中提取数据 | 直接访问数据， |
| 通常用于动态的数据结构 | 通常用于固定数目且数据类型相同的元素 |
| 通过Malloc分配内存，free释放内存 | 隐式的分配和删除 |
| 通常指向匿名数据，操作匿名函数 | 自身即为数据名 |