1. <https://www.nowcoder.com/discuss/472070410226495488?sourceSSR=search>
2. **C++智能指针**

智能指针就是帮助程序员管理动态分配的内存，它会帮助我们自动释放new出来的内存，从而避免内存泄漏！

C++ 11引入了3个智能指针类型：

std::unique\_ptr<T>：独占资源所有权的指针，无法进行左值复制赋值操作，但允许临时右值赋值构造和赋值

std::shared\_ptr<T>：共享资源所有权的指针，就是记录对象被引用的次数，当引用次数为 0 的时候，也就是最后一个指向该对象的共享指针析构的时候，共享指针的析构函数就把指向的内存区域释放掉；多个shared\_ptr可以指向同一份资源，并在内部使用引用计数机制来实现这一点。

std::weak\_ptr<T>：共享资源的观察者，需要和 std::shared\_ptr 一起使用，不影响资源的生命周期。

1. auto\_ptr的作用

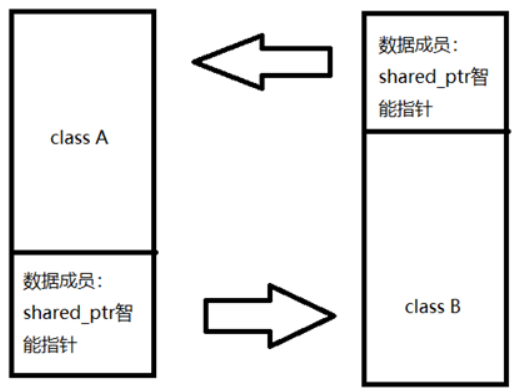
std::auto\_ptr已经废弃，被unique\_ptr代替，它是模板类，当auto\_ptr对象过期时，析构函数会使用delete来释放内存。

缺点：

**复制或者赋值都会改变资源的所有权**

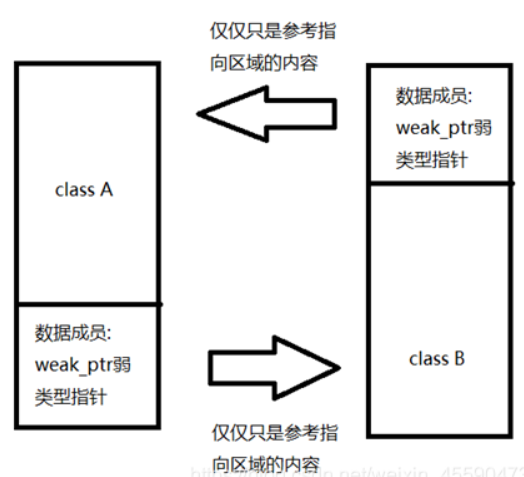
**不支持动态数组的内存管理**

1. **智能指针循环引用问题**



常常把weak\_ptr当作shared\_ptr的小弟，我们知道shared\_ptr不仅可以访问指向的内存区域而且还可以掌管指向区域的“生死大权”，也就是说伴随着shared\_ptr指针生命周期的结束，指针指向的区域也会被释放掉。但是weak\_ptr作为shared\_ptr的小弟，weak\_ptr不敢这样做，作为智能指针中最弱的一个，weak\_ptr只能访问所指向的内存区域，当weak\_ptr指针生命结束之时，其所指向的内存依旧完好无损，这就是weak\_ptr小弟和shared\_ptr大哥的本质区别。

由于在weak\_ptr指针生命结束之时，不会对指向内存产生任何影响，因此不会出现“上述shared\_ptr引发的环形引用的异常错误”。如果将上述例子中，shared\_ptr换做weak\_ptr结构将会发生变化：



1. **shared\_ptr和unique\_ptr在发生异常而导致的内存泄漏的时候能否替代auto\_ptr解决问题？**
2. **unordered\_map实现原理、底层结构**

基于hash表实现，在不发生冲突的情况下，插入、查找和删除的时间复杂度均为O(1)。具体实现原理如下：

1. 哈希函数：将键值映射到桶的索引上。
2. 冲突解决方法：拉链法或者开放寻址法
3. **unordered\_map发生哈希冲突怎么解决**

unordered\_map采用拉链法解决冲突，用链表连接存储在同一个桶中的数据。

1. **常见解决哈希冲突的办法。**

拉链法：用链表连接存储在同一个桶中的元素。

开放寻址法：发生冲突时，继续寻找空桶。

再哈希法：冲突就再次哈希，直至不发生冲突。

1. **vector,list,map,unordered\_map的区别，底层结构**

Vector：数组

List：双向链表

Unordered\_map：hash表

1. **栈和堆的区别**
2. 程序[内存](https://so.csdn.net/so/search?q=%E5%86%85%E5%AD%98&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/K346K346/article/details/_blank)布局场景下，堆与栈表示两种内存管理方式；

栈：栈由操作系统自动分配释放 ，用于存放函数的参数值、局部变量等，其操作方式类似于数据结构中的栈。

堆：堆由开发人员分配和释放， 若开发人员不释放，程序结束时由 OS 回收，分配方式类似于链表。

（2）[数据结构](https://so.csdn.net/so/search?q=%E6%95%B0%E6%8D%AE%E7%BB%93%E6%9E%84&spm=1001.2101.3001.7020)场景下，堆与栈表示两种常用的数据结构。

栈：先进后出的线性表

堆：树形结构，是一种特殊的完全二叉树

1. **new和malloc的区别**

**属性的区别**

new/delete：这两个是C++中的关键字，若要使用，需要编译器支持；

malloc/free：这两个是库函数，若要使用则需要引入相应的头文件才可以正常使用。

**使用上的区别**

malloc：申请空间需要显式填入申请内存的大小；

new：无需显式填入申请的内存大小，new会根据new的类型分配内存。

**内存位置的区别**

new：此操作符分配的内存空间是在自由存储区（自由存储区可以是堆、全局/静态存储区）；

malloc：申请的内存是在堆空间（堆是操作系统所维护的一块特殊内存，它提供了动态分配的功能）。

**返回类型的区别**

new操作符内存分配成功时，返回的是对象类型的指针

malloc内存分配成功则是返回void \* ，需要通过强制类型转换将void\*指针转换成需要的类型

**分配失败情况的区别**

malloc分配内存失败时返回NULL，我们可以通过判断返回值可以得知是否分配成功；

new内存分配失败时，会抛出bac\_alloc异常

**扩张内存大小的区别**

malloc：使用malloc分配内存后，可以通过realloc函数来扩张内存

new：new没有扩张内存的机制。

1. **C++ 常使用哪些 STL 容器**
2. **map 和 unordered\_map 区别和底层实现是什么**
3. **map 除了查询复杂度高还有什么缺点**

**map优点：**

1. map元素有序（这是map最大的优点，其元素的有序性在很多应用中都会简化很多的操作）；
2. 其红黑树的结构使得map的很多操作都可在O(logn)下完成；
3. map的各项性能较为稳定，与元素插入顺序无关；
4. map支持范围查找。

**缺点：**

1. 占用的空间大：红黑树的每一个节点需要保存其父节点位置、孩子节点位置及红/黑性质，因此每一个节点占用空间大。
2. 查询平均时间不如unordered\_map。

**适用场景：**

1. 元素需要有序；
2. 对于单次查询时间较为敏感，必须保持查询性能的稳定性，比如实时应用等等。

**unordered\_map优点：**

1. 查询速度快，平均性能接近于常数时间O(1)；

**缺点：**

1. 元素无序；
2. unordered\_map相对于map空间占用更大，且其利用率不高；
3. 查询性能不太稳定，最坏时间复杂度可达到O(n)。

**适用场景：**

1. 要求查找速率快，且对单次查询性能要求不敏感。
2. **vector 增删改查复杂度分别是什么**
3. **vector 支持越界检查吗**

[vector](https://so.csdn.net/so/search?q=vector&spm=1001.2101.3001.7020)以at的形式支持越界检查（**越界会抛出异常**），但也支持C语言风格的[]高效访问

1. **讲讲 vector 扩容**

底层数据结构是一个动态数组。 默认容量是0， 之后插入按照1 2 4 8 16二倍扩容。GCC是二倍扩容，VS13是1.5倍扩容。扩容后是一片新的内存，需要把旧内存空间中的所有元素都拷贝进新内存空间中去，之后再在新内存空间中的原数据的后面继续进行插入构造新元素，并且同时释放旧内存空间，并且，由于vector 空间的重新配置，导致旧vector的所有迭代器都失效了。

1. **push\_back 的复杂度一定是 O(1) 吗**

push\_back均摊后的[时间复杂度](https://so.csdn.net/so/search?q=%E6%97%B6%E9%97%B4%E5%A4%8D%E6%9D%82%E5%BA%A6&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/weixin_44178736/article/details/_blank)为O(1)。

vector只有当达到capacity之后才会扩容。

1. **push\_back和emplace\_back 区别**

push\_back调用构造函数构造这个临时对象，然后调用拷贝构造函数将这个临时对象复制到容器。

emplace\_back直接通过参数构造对象至容器中，不需要拷贝或者移动。如果传递的是一个已经存在的对象，那么将会调用该对象的复制构造函数。



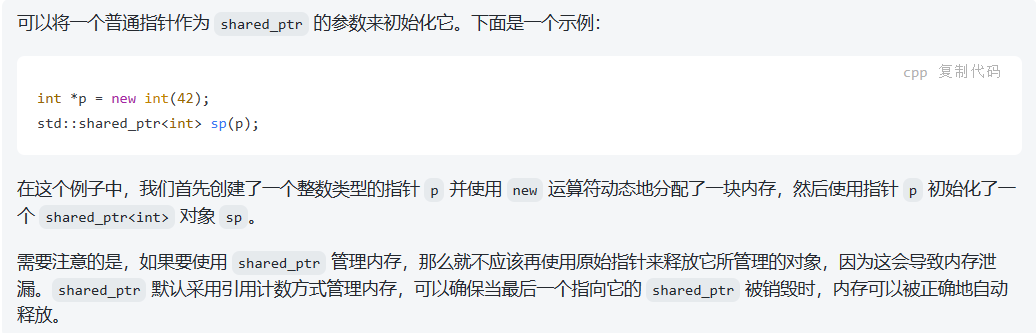
1. **emplace\_back可以传对象吗**
2. **移动构造函数和拷贝构造函数区别**

拷贝构造函数的参数是一个左值引用，但是移动构造函数的参数是一个右值引用。

1. **智能指针用过哪些，讲讲特点**
2. **shared\_ptr引用计数何时增加和减少**

每当一个新的指针指向资源时，计数器就会增加，而在share\_ptr被销毁时，计数器就会减少。

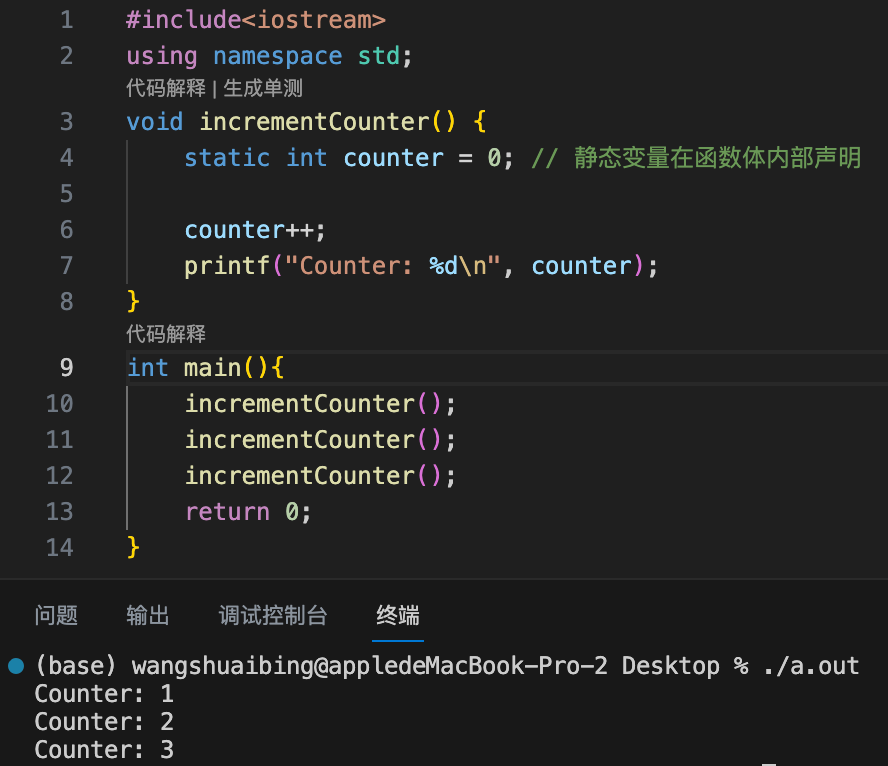
1. **如何用普通指针初始化shared\_ptr**



1. **用普通指针初始化 shared\_ptr 这种用法有什么坏处**
2. 多个 shared\_ptr 对象管理同一个内存资源时，需要保证引用计数正确，即所有 shared\_ptr 对象共享相同的计数器。如果使用普通指针创建多个shared\_ptr，则会导致引用计数不一致，从而产生未知行为。
3. 如果 shared\_ptr 管理的内存资源被多次释放，则会产生运行时错误，可能导致程序崩溃或出现难以排查的问题。
4. 普通指针不能自动判断内存资源是否还在使用中，也无法确定何时释放，容易造成内存泄漏或野指针的问题。

因此，在使用 shared\_ptr 时，最好避免直接使用普通指针来初始化它，可以使用 make\_shared 或者 shared\_ptr 的构造函数来分配和初始化内存。这样可以确保引用计数正确、内存可控、便于维护，在代码开发和优化中更加安全和稳定。

1. **static 变量可以用在哪些地方，分别有什么特点**
2. static最重要的作用是**隐藏**，当同时编译多个文件时，所有未加static的全局变量和函数都具有全局可见性，但加上static后的变量，只在当前文件可见；
3. 其次是保持变量内容的持久，也就是说保持局部变量在[函数调用](https://www.zhihu.com/search?q=%E5%87%BD%E6%95%B0%E8%B0%83%E7%94%A8&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7b" \t "/Users/wangshuaibing/Documents\\x/_blank)结束后不被销毁。



1. static变量默认初始化为0
2. 在类内，被static修饰的成员变量和函数被整个类所拥有。

注意：C语言中static修饰的变量在编译期间初始化，C++中由于引入了对象，在首次执行到定义语句被初始化。

1. **static 修饰的普通变量初始化在main 函数执行前还是后**

全局变量和[static](https://so.csdn.net/so/search?q=static&spm=1001.2101.3001.7020)的初始化就是 在main函数之前执行的

1. **类的 static 方法可以访问非 static 对象吗**

答案是不可以

1. 静态成员属于类而不是属于类对象，因此静态成员没有this指针；而非静态成员属于对象，因此非静态成员拥有this指针，指向所属对象。

b) 调用静态成员函数时，并不知道其中的 non-static member从哪来，属于哪个对象，甚至可能都不存在。因此静态成员函数无法访问自己类的非静态成员。

1. **构造和析构函数调用顺序？**

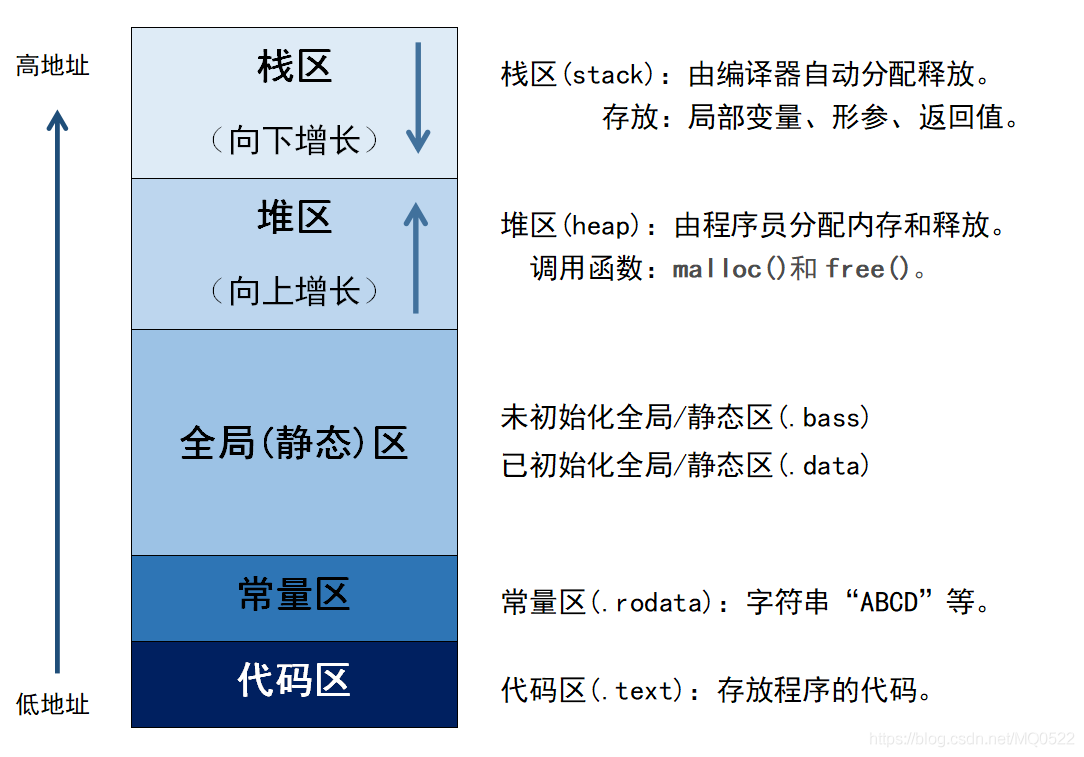
基类构造函数 -> 成员对象构造函数 -> 类自身构造函数。

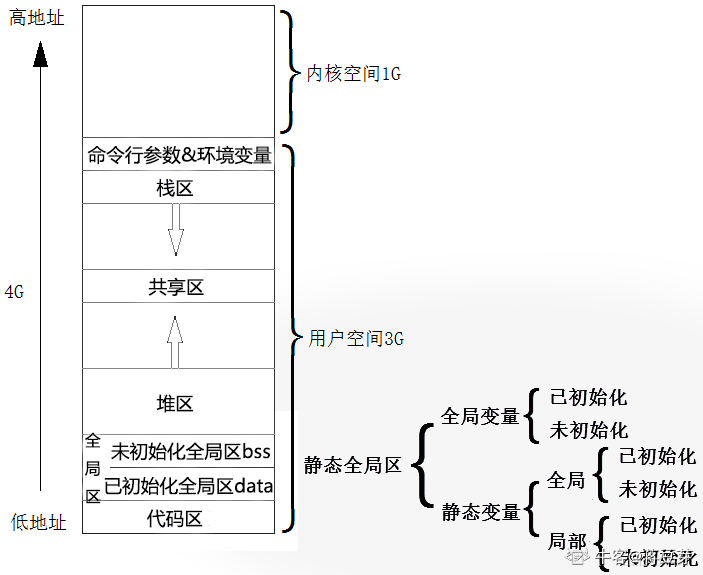
类自身析构函数 -> 成员对象析构函数（逆序）-> 基类析构函数（逆序）。

1. **c++中i++ 和++i的区别？++i快还是i++更快呢？**
2. 赋值顺序不同：++ i 是先加后赋值；i ++ 是先赋值后加；++i和i++都是分两步完成的。
3. 效率不同：后置++执行速度比前置的慢。
4. i++不能作为左值，而++i可以，它返回的是一个临时结果，而不是变量本身。
5. 两者都不是原子操作。
6. **两个float类型如何判断相等？**

浮点数在计算机内部是以二进制形式存储的，而在这种存储方式下，可能会存在精度误差的问题。这意味着两个看似完全相等的浮点数，在计算机内部表示时可能存在微小的差异。

使用差值判断：判断两个浮点数之间的差值是否小于一个给定的误差范围，如果小于误差范围则认为两个浮点数相等。

**13）c++内存分配了解么？**



如上图，从低地址到高地址，一个程序由代码段、数据段、BSS段、堆栈段组成。

代码段：存放程序执行代码的一块内存区域。只读，不允许修改，代码段的头部还会包含一些只读的常量，如字符串常量字面值（注意：const变量虽然属于常量，但是本质还是变量，不存储于代码段）。

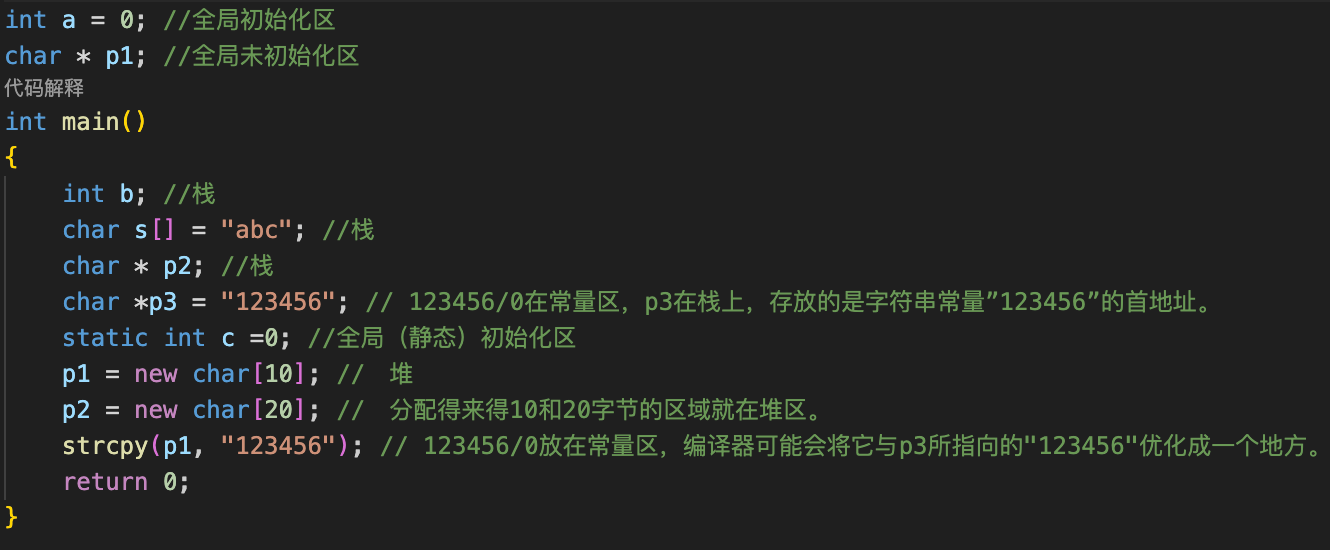
常量区：字符串、数字等常量存放在常量区；const修饰的全局变量存放在常量区；程序运行期间，常量区的内容不可以被修改。

数据段data：存放程序中已初始化的全局变量和静态变量的一块内存区域。

BSS 段：存放程序中未初始化的全局变量和静态变量的一块内存区域。

可执行程序在运行时又会多出两个区域：堆区和栈区。\*\*堆区：\*\*动态申请内存用。堆从低地址向高地址增长。栈区：存储局部变量、函数参数值。栈从高地址向低地址增长。是一块连续的空间。

最后还有一个文件映射区（共享区），位于堆和栈之间。



<https://www.nowcoder.com/feed/main/detail/04034133cd9d42a38ac2f9b1058e152e?sourceSSR=search>

1. class 和 struct的区别

使用 class 时，类中的成员默认都是 [private](https://haicoder.net/c/c-keywords.html) 属性的，而使用 struct 时，结构体中的成员默认都是 public 属性的。

class 继承默认是 private 继承，而 struct 继承默认是 public 继承。

template <class T>和template <typename T>等价，template <struct T>编译不会通过。  
2. 一个空的class里面默认有什么函数

简称为空类，系统会自动生成六个默认的成员函数，构造函数、析构函数、拷贝构造函数、赋值运算符、取地址运算符、const取地址运算符

3. 右值引用

右值（Rvalue）指的是不能出现在赋值语句左边的表达式，即不能被赋值的表达式，例如常量、临时对象等。与之对应的是左值（Lvalue），指的是可以出现在赋值语句左边的表达式，即可以被赋值的表达式，例如变量、数组元素等。

4. vector和list的底层实现

5. inline内联函数

inline关键字，在编译阶段，编译器使用函数定义的代码直接替换调用部分，然后对整个代码进行编译。这将大大减少上述运行时的开销，提高程序的整体效率。

优点

（1）避免函数调用提高程序运行效率

（2）减少函数调用时栈操作的开销

（3）优化内存访问的局部性

缺点

（1）当内联函数定义改变时，需要重新编译整个程序。

（2）因为内联函数本质是代码的扩展，它增加了执行程序的大小。

（3）可能因为物理内存不足而导致缺页异常。

**编译器在什么情况下会禁止inline：**

1. 大型函数，包含大量的代码行数。
2. 递归函数，内联展开需要在编译时展开函数调用，而递归函数的调用是自身调用，无法提前确定调用的次数。
3. 虚函数，它是通过运行时动态绑定来实现多态，因此编译器无法在编译阶段将其内联。
4. 通过函数指针调用inline函数，函数指针确定被调用的代码是运行时完成的，这一点与虚函数相似。

6. Null和nullptr的区别

在C语言中，NULL通常被定义为：#define NULL ((void \*)0)

在C++中，NULL实际上是0。

但是实际上，用NULL代替0表示空指针在函数重载时会出现问题。

为解决NULL代指空指针存在的二义性问题，在C++11版本中特意引入了nullptr这一新的关键字来代指空指针。

void func(int p){}

void func(int\* p){}

func(NULL); // 因为二义性报错

<https://www.nowcoder.com/discuss/473609195074387968?sourceSSR=search>

1. 指针相关，问最后的输出

char str1[] = "ab";

char str2[] = "ab";

char\* str3 = "ab";

char\* str4 = "ab";

cout<<(str1 == str2) << endl; //0

cout<<(str3 == str4) << endl; //1

str1和str2两个字符数组都存储在栈空间上，但两者地址值不相等。而str3和str4并非字符数组而是字符指针，并不分配存储区，其后的“abc”以常量形式存储在常量区，str3和str4是他们指向的地址的首地址，而他们仅是指向该区首地址的指针，多一相等（&str3和&str4是指指针自己的地址，所以两者地址是不相等的）

1. vector和数组

C++中数组是一种内置的数据类型。

数组是存放类型相同的对象的容器，数组的大小确定不变，不能随意向数组中增加元素。

vector是类型相同的对象的容器，vector的大小可以变化，可以向数组中增加元素。

1. 延伸着问vector是怎么动态分配的

底层数据结构是一个动态数组。 默认容量是0， 之后插入按照1 2 4 8 16二倍扩容。GCC是二倍扩容，VS13是1.5倍扩容。扩容后是一片新的内存，需要把旧内存空间中的所有元素都拷贝进新内存空间中去，之后再在新内存空间中的原数据的后面继续进行插入构造新元素，并且同时释放旧内存空间，并且，由于vector 空间的重新配置，导致旧vector的所有迭代器都失效了。

1. 多态

C++ 多态意味着调用成员函数时，会根据调用函数的对象的类型来执行不同的函数。

静态多态就是重载，因为在编译期决议确定，所以称为静态多态。在编译时就可以确定函数地址。动态多态就是通过继承重写基类的虚函数实现的多态。运行时在虚函数表中寻找调用函数的地址。

总结：如果这个类**声明了至少一个虚函数**，那么它会**拥有一个虚函数表**。

如果这个类**没有声明虚函数**，它将**不会拥有虚函数表**。

当一个类继承自另一个类时，它会**继承父类的虚函数表指针**。这意味着派生类将共享其基类的虚函数表，但可能会在其中添加新的虚函数或重写已有的虚函数。

每个**对象实例并不拥有自己的虚函数表，而是共享它们所属类的虚函数表**。这样做是为了节省内存，并且可以实现多态性，允许在运行时动态绑定适当的虚函数。

在多态性和继承关系中，每个类都有一个虚函数表指针（vptr），指向其对应类的虚函数表。当调用虚函数时，实际上是通过这个虚函数表指针找到对应的虚函数地址进行调用。在多继承情况下，有多少个基类就有多少个虚函数表指针，前提是基类要有虚函数才算上这个基类。

7. extern的作用，extern “C”的作用

extern 是一个关键字，用于声明一个变量或函数是在其他源文件中定义的。

extern "C" 来指示编译器按照 C 语言的方式进行链接。  
15. c++右值引用，完美转发

移动语义允许我们将资源的所有权从一个对象转移到另一个对象，而不是进行拷贝或复制。

完美转发是指在不改变参数类型的情况下，将参数传递给另一个函数。

<https://www.nowcoder.com/discuss/472725095295766528?sourceSSR=search>

1. C++ volatile关键字的作用

防止变量被编译器优化。

volatile 指出 i 是随时可能发生变化的，每次使用它的时候必须从 i的地址中读取。

"volatile"是一个关键字，用于指示变量是易变的（volatile）。它告诉编译器不要对该变量进行优化，因为它可以在意料之外的时间被修改。这在多线程或并发编程中特别有用。当一个变量被声明为volatile时，编译器会生成特殊的代码来确保对该变量的访问是直接的，而不是从**缓存**中读取。这样做可以避免编译器优化导致的错误结果。

2. C++ 虚函数的作用

3. C++ 指针和引用的区别，什么时候用指针，什么时候用引用

指针是一个变量，它存储了一个内存地址，而引用本质上是一个别名，它与某个变量绑定在一起。

指针可以被赋值为NULL，表示它不指向任何有效的内存地址，而引用必须始终引用某个变量。

在选择使用指针或引用时，应该考虑以下几个方面：

当需要在函数中修改函数外部的变量时，通常会使用引用。这是因为引用使得参数按照引用传递，可以直接对原始变量进行修改，而不需要通过指针间接地访问变量。

当需要动态分配内存或者操作复杂的数据结构时，通常会使用指针。这是因为指针可以通过new和delete运算符动态分配和释放内存，而引用不能做到这一点。

当需要迭代数组或容器中的元素时，通常会使用指针。这是因为指针可以进行算术运算，从而实现对数组或容器中的元素进行遍历，而引用则没有这样的能力。