C++面试题

**那些函数不可以是虚函数：**

(1)类外的普通函数不能声明为虚函数；只能用virtual声明类的成员函数，将其作为虚函数，因为虚函数的作用是允许在派生类中对基类的虚函数重新定义

（2）一个函数已经被声明成虚函数之后，在同一族类中不能再定义一个非虚但与该虚函数具有相同参数和函数返回值类型的同名函数

（3）构造函数不能声明为虚函数，因为执行构造函数的时候对象还没有完全建立，谈不上动态绑定

**虚函数怎么定义：**

（1）在基类中用virtual声明成员函数之后，在类体外定义虚函数时，不必再加virtual；

（2）在派生类中重新定义此函数，函数名，函数类型，函数参数个数和类型必须与基类中的虚函数相同，根据派生类的需求设计新的函数体。

（3）当一个成员函数被定义成虚函数之后，其派生类的同名函数自动成为虚函数，在派生类中可以加virtual也可以不加，

（4）如果派生类中没有对基类的虚函数重新定义。则简单继承派生类的虚函数

（5）定义一个指向基类对象的指针变量，并使他指向同一族类中需要调用该函数的对象；

（6）通过指针变量调用该虚函数，此时调用就是指针变量指向的对象的同名函数。

**虚函数的实现：**

在类的最开始部分是一个虚函数表的指针；这个指针指向一个虚函数表，表中存放了虚函数的地址，实际的虚函数代码在代码段中。当子类继承父类的时候也会继承其虚函数表，当子类重写父类的虚函数时，会将其继承到虚函数表中的地址重新写的函数地址，使用虚函数表会增加访问内存开销，降低效率。

**静态绑定与动态绑定**：

（1）静态绑定：函数重载，或者通过对象名调用虚函数，在编译的时候就确定其调用的虚函数属于那一个类，在运行前关联，又称之为早期关联。

（2）通过基类指针与虚函数结合，编译系统早期是无法确定用哪一个类的对象的虚函数。因为编译只是静态的语法检查。此时，就在运行的时候确定关联关系。动态关联体现的是动态的多态性，运行阶段的多态性。又称为“滞后关联”。

**指针和数组的区别**：

（1）指针是存放的数据的地址，数组存放的是数据

（2）指针间接访问数据，数组直接访问数据

（3）常用于动态的数据结构，数组常用于固定数目且数据类型相同的结构

（4）通过malloc分配free分配和释放，数组分为隐式的分配和删除

（5）通常指向匿名数据，操作匿名函数，数组自身就是数据名

**C语言怎么实现函数调用:**

每一个函数都会分配函数栈，在栈内进行函数执行过程，调用前，先把返回地址压栈，然后把当前函数的esp指针压栈。

**C语言的压栈顺序（从右到左）**

**map和set区别：**

map和set底层实现都是红黑树。

（1）map中的元素是key-value对：关键字起到索引的作用，值则表示与索引相关联的数据；set与之相对就是关键字的简单集合，set每个元素只包含一个关键字

（2）set的迭代器是const的，不允许修改元素的值；map允许修改value，但是不允许修改key。原因是map和set是根据关键字排序来保证其有序的，如果允许修改key的话，那么首先要删除该key，然后调节平衡，如此就严重破坏了map和set的结构，导致迭代器失效。

（3）map支持下标操作，set不支持下标操作：map可以将key值作为下标，map的下标运算符[]将关键码作为下标去执行查找，如果不存在，则插入一个具有该关键码和mapped\_type类型的元素到map中，所以[]要慎用。

**请说明C++如何处理返回值**：

生成一个临时变量，把它的引用作为函数参数传进函数内

1、不要返回局部对象的引用或者指针

2、引用返回左值，其他返回类型得到右值

3、main函数返回值中，0表示成功，其他表示识别。

**STL的组成**：

（1）**容器迭代器** **仿真函数**，**算法分配器** **配接器**

分配器给容器分配存储空间，算法通过迭代器获取容器中的内容，仿函数可以协助算法完成操作，配接器用来套接适配仿函数

**STL中的resize与reserve**：

（1）resize（）

改变当前容器中含有元素的数量，如果原来的长度小于新增的长度，那么容器就新增几个元素，新增的值都是0；

（2）reserve（）

改变当前容器的最大容量，不会生成新的元素。

**STL中的map与unordered map**：

**Map**：所有的元素都是pair，同时拥有key和value。pair第一个元素视为键值，第二个元素视为实值，所有元素根据键值自动排序，不允许键值重复。

底层实现：红黑树

使用场景：有序键值对不重复映射

**unordered\_map**：底层实现：哈希表

**multimap：**

多重映射：所有元素都是pair，同时拥有实值和键值。pair的第一个元素是键值，第二个元素是实值。底层实现是红黑树

使用场景：有序键值对可重复映射。

**什么时候会发生段错误**：

通常发生在访问非法内存地址的时候：

（1）使用野指针

（2）试图修改字符串常量的内容

**C++的内存泄漏与内存溢出,内存泄漏怎么查找**：

内存泄漏（memory leak）：由于疏忽或者错误造成程序未能释放掉不在使用的内存的情况，对该段内存失去了控制，造成内存的浪费

**内存泄漏的分类**：

（1）堆内存泄漏（heap leak）：通过malloc等申请一块内存之后，没有对其释放

（2）系统资源泄漏：程序使用系统分配的资源比如：bitmap，handle，socket等没有释放，造成系统不稳定

（3）没有将基类的析构函数定义为虚函数。子类资源不能正确释放。

C++内存泄漏的处理与查找：使用varglind，mtrace检测，也可以在写代码的时候添加内存申请和释放的统计功能没统计申请和释放是否一致。

**内存溢出和内存越界**：

//内存溢出：申请内存的时候，没有足够的内存空间供其使用。

//内存越界：向系统申请一块内存之后，使用时超出了申请范围

//内存泄漏：申请了未释放

**malloc原理，brk系统调用和mmap系统调用的作用**：

malloc的作用是动态分配内存。为了减少内存碎片和系统调用的开销，malloc采用内存池的方式，先申请大块的内存作为堆区，然后将堆区分成多个内存块，以块作为内存管理的基本单位。当用户申请内存的时候，直接从堆区中分配一块合适的空闲块。

malloc采用隐式链表结构将堆分成连续的，大小不一的块，包含已经分配的块和未分配的块；同时malloc采用显示链表结构来管理所有的空闲块，使用一个双向链表将空闲块连接器来，每一个空闲块记录了一个连续的未分配的地址

当进行内存分配的时候，malloc会通过隐式链表遍历所有的空闲快，选择满足要求的块进行分配；当进行内存合并的时候，malloc采用边界标记的方法，根据每个快的前后是否已经分配来决定是否进行快合并。

malloc在申请内存之前，一般会通过brk或者mmap系统调用进行申请。其中当申请内存小于128k时，会使用系统brk在堆中分配；当申请内存大于128k，使用系统函数mmap在映射区分配。

**malloc与new区别**：

（1）new分配内存按照数据类型分配，malloc按照指定大小分配

（2）new返回的是指定对象的指针，malloc放回的是void\*

（3）new一般不分配一段内存，而且会动态调用析构函数，malloc不会

（4）new分配的内存用delete销毁，malloc用free销毁

（5）new是一个可重载的操作符，malloc是一个库函数

（6）malloc分配内存不够的时候，可以用realloc扩容，new不可以

（7）malloc分配失败会抛出bad\_malloc异常，new是返回NULL；

（8）申请数组：new[]多次分配所有内存，多次调用构造函数，delete[]多次调用析构函数。

**C++函数栈空间的最大值：1M**；

**C++与C的区别：**

设计思想上：

C++是面向对象的语言，C是面向过程的结构化编程语言

语法上：

C++具有封装、继承、多态三种特性

C++相比C，增加了安全性能，类型强制转换；

C++支持范式编程，比如模板类等

**隐式类型转换**：

对于内置的类型，低精度的给高精度的赋值会发生隐式转换，对于只存在单个参数的构造函数的对象构造来说，函数调用可以直接使用该参数传入，编译器会自动调用其构造函数生成临时的对象。

**C++定义引用数据成员：必须通过成员函数初始化列表进行初始化**

**如何采用单线程的方式处理高并发：**

可以采用I/O复用来提高单线程处理多个请求的能力，然后采用事件驱动模型，基于异步回调来处理事件。

**C++从文本到可执行文件经历的过程**：

四个过程：

（1）预处理阶段：对源代码文件中包含关系（头文件）、预编译语句（宏定义）进行分析和替换，生成编译文件

（2）编译阶段：将经过预处理后的预编译文件转换成特定的汇编代码，生成汇编文件

（3）汇编阶段：将编译阶段生成的汇编文件转换成机器码，生成可重定位目标文件

（4）链接阶段：将多个目标文件及所需要的库链接成最终的可执行的目标文件。

**类中实例对象和类对象区别**：

没有分配内存的只是对象（指针），分配了内存的是实例化的对象

**指针与引用**：

（1）指针有一块空间，引用只是别名

（2）sizeof看一个指针的大小为4，看引用的大小为引用对象的大小

（3）指针可以被初始化为NULL，但是引用必须初始化为一个实际的对象

（4）作为参数传递的时候，指针需要解引用才可以对对象进行操作，对引用的直接修改就会改变引用所指的对象

（5）指针在使用的过程中可以指向其他对象，引用只能指向初始化的对象，不能改变

（6）指针可以有多级指针，引用只有一级

（7）指针和引用的++运算符的意义不同，引用时对象自增

（8）如果返回动态分配内存的对象或者内存，必须使用指针，引用可能引起泄露。

**Const**：

（1）定义常量，

（2）const修饰函数形式参数，可以提高效率，避免值被修改

（3）const修饰函数返回值，则返回值不能被直接修改，其返回值只能被赋值给const同类指针

（4）const用于修饰类的成员函数，用于封装，保证内部不会被修改，防止非法修改。

（4）对于类的成员函数，若指定其为const类型，则表明其是一个常函数，不能修改类的 成员变量；

（5）对于类的成员函数，有时候必须指定其返回值为const类型，以使得其返回值不为“左值”。

（6）类的const成员，不能被修改，只能初始化列表赋值；

（7）void fun（）const;

（8）const定义的常量在运行时只有一份拷贝，编译的时候只是放在符号表中。

（9）const类成员函数不能修改成员变量，可以访问const变量，也可以访问其他成员。

（10）const成员只能被const函数访问。

**Static**：

（1）全局静态变量：在全局变量之前加上static，存储在静态存储区，整个程序运行期间存在。未经初始化自动赋值为0。作用域声明的该文件中，声明的文件之外是不可见的

（2）局部静态变量：静态存储区，未初始化自动为0；局部作用域，作用域之外不能访问，当作用域结束之后，不会消失，下一次引用上一次的值。

（3）静态函数：函数定义和声明默认是extern的，加上之后，这个函数就不能被其他文件所用。

（4）类的静态成员：静态成员可以实现多个对象之间的数据共享，不会破坏隐藏性的原则。静态数据成员存储在一处，所有对象成员都可以访问，其是属于类的，只能在类外初始化

（5）类的静态函数：属于类的，不属于特定成员。静态成员函数不能直接引用类中的非静态成员，非静态成员需要通过对象名来引用。

（6）非静态成员函数也可以访问静态成员，

（7）若其定义为私有的，必须通过接口访问。

**Volatile**：

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，因而编译器生成的汇编代码会重新从i的地址读取数据放在 b 中，对访问该变量的代码就不再进行优化。而优化做法是，由于编译器发现两次从 i读数据的代码之间的代码没有对 i 进行过操作，它会自动把上次读的数据放在 b 中。而不是重新从 i 里面读。

1) 中断服务程序中修改的供其它程序检测的变量需要加 volatile；

2) 多任务环境下各任务间共享的标志应该加 volatile；

3) 存储器映射的硬件寄存器通常也要加 volatile 说明，因为每次对它的读写都可能由不同意义。

**可变参数模板**：

对参数高度泛化，可以表示任意数目，任意类型的参数。class和typename后面加省略号

Template<class ... T>

void func(T ... args)

{

cout<<”num is”<<sizeof ...(args)<<endl;

}

func();//args不含任何参数

func(1);//args包含一个int类型的实参

func(1,2.0)//args包含一个int一个double类型的实参

其中T叫做模板参数包，args叫做函数参数包

省略号的作用：

（1）声明一个包含0到任一个参数的参数包

（2）在模板定义的右边，可以将参数包展成一个个独立的参数

**STL为什么有指针还需要迭代器**：

（1）iterator是运用聚合对象的一种模式，通过运用该模式，使得我们可以在不知道对象内部表示的情况下，按照一定的顺序访问聚合对象的各个元素

（2）迭代器不是指针，是类的模板，表现的像指针。他只是模拟了指针的一些功能，通过重载了指针的一些操作符，->,\*,++,--等。迭代器封装了指针，是一个可以遍历的STL容器内部全部和部分元素的对象。

本质上是封装了原生指针，是指针概念的提升，提供了比指针高级的行为，相当于一种智能指针，可以根据数据类型的不同实现不同的++，--操作。

迭代器返回对象引用，而不是对象的值，所以cout只能输出迭代器使用\*的取值后的值，不能使用迭代器直接输出。

迭代器的作用：把不同集合类的访问逻辑抽象出来，使得不暴露集合内部的结构从而达到循环遍历的效果

**右值引用与左值引用**：

目的：

（1）消除两个对象交互时不必要的对象拷贝，节省运算存储资源，提高效率

（2）能够简洁明了的定义泛型函数

左值：能对表达式取地址，或者具名对象/变量。一般表达式结束后依然存在的持久变量

右值：不能对表达式取地址，或匿名对象，一般指表达式结束了就不再存在的临时对象

区别

（1）左值可以寻址，右值不可以

（2）左值可以被赋值，右值不可以被 赋值，但是可以给左值赋值

（3）左值可变，右值不可变。

**list与vector区别**：

（1）vector

连续存储的容器，动态数组，在堆上分配空间；

底层实现：数组，两倍容量扩充

如果容量不够了，则会重新配置所有元素个数的两倍空间，然后将元素通过复制的方式初始化新的空间，再向新空间中增加元素，最后析构释放源空间，之前的迭代器会失效

访问：O(1)；

插入：最后插入，空间够的时候很快；空间不够的时候，需要拷贝和释放内存，

在中间插入的时候，空间够，内存拷贝；空间不够的时候，需要申请和释放，对源数据拷贝

删除：在最后删除，快；在中间删除，内存拷贝

使用场景：经常随机访问，且不经常对非尾部节点进行插入删除的操作

（2）list

动态链表，在堆上分配空间，每插入一个元素分配一个空间，每删除一个元素释放一个空间

访问：随机访问性差，只能快速访问头尾节点

插入：很快，常数开销

删除：很快，常数开销

使用场景：经常需要删除大量数据

（1）vector底层是数组，list底层是双向链表

（2）vector支持随机访问，list不支持

（3）vector是顺序内存，list不是

（4）vector中间节点插入和删除导致内存拷贝，list不会

（5）vector一次性分配好内存，不够时才进行2倍扩容，list每次插入新节点的时候进行内存申请

（6）vector随机访问性能好，插入删除性能差；list相反

**C++内存布局**：

C++中，虚拟内存分为代码段，数据段，BSS段，堆区，文件映射区以及栈区六大部分。

**代码段**（text segment）：包括只读存储和文本区，其中只读存储区存储字符串常量，文本区存储程序的机器代码

**数据段**（data segment）：存储程序中已经初始化的全局变量和静态变量

**bss段**：存储未初始化的全局变量和静态变量（局部+全局），以及所有被初始化为0的全局变量和静态变量

**堆区**：调用new/malloc函数在堆区动态分配内存

**映射区**：存储动态链接库以及调用mmap函数进行的文件映射

**栈**：使用栈空间存储函数的返回地址，参数，局部变量，返回值

**C++中struct与class的区别**：

C++中的struct与class都可以继承

区别在于：

struct默认继承是public，class默认继承是pravite；另外class可以定义模板类形参

**C++的重载与重写**：

//重载：两个函数名相同，但是参数个数，参数类型或者参数顺序有所不同，返回值没有要求，在同一个作用域中

//重写：指的是派生类继承父类的时候，对父类的虚函数进行重新的定义，要求所有的都一样。

**C++中四种cast转换**：

（1）const\_cast

用于将const变量转为非const变量

（2）static\_cast：用于各种隐式的转换，比如非const转const。能用于多态向上转化，如果向下准话成功但是不安全，结果未知

（3）dynamic\_cast：用于动态类型转换，只能用于含有虚函数的类，用于类层次间的向上和向下转化，只能转指针或引用。

向下转换：指的是子类向基类的转换，如果是非法的，指针返回NULL，引用则抛出异常

向上转换：指的是基类向子类转换。

（4）reinterpret\_cast：什么都可以转，可能会出现问题，少用

C的强制转换不会进行错误检查。

**函数指针：**

函数指针，指向函数的指针，可以利用该指针变量调用一个函数：int（\*p）（）；

**如何定义一个无法继承的类**：

C++子类的构造函数会调用父类的构造含糊，只要构造函数和析构函数都声明为私有的，当一个类企图继承的时候，会导致编译错误，通过定义静态来创建和释放类的实例。其能实例化，但是不能被继承。

**join(),wait(),sleep()函数：**

（1）join：使得调用该函数的线程在此之前执行完毕，也就是等待该函数的线程执行完毕后再继续往下执行

（2）wait：当一个线程执行wait函数，就进入到一个和该对象相关的等待池中（等待阻塞），同时释放对象锁，并且让出CPU资源，待指定时间结束后返回得到对象锁。用于协调多线程堆共享数据的存取，只能在同步方法或者同步快中使用

（3）sleep：线程类的静态方法，让调用的线程进入指定时间的睡眠状态，使得当前线程进入阻塞状态，告诉线程至少在指定时间内不需要为线程调度器为该线程分配指向时间片，给机会给其他线程，但是监控状态依然保持，到时候自动回复。

**fork函数**：

创建一个和当前进程映像一样的进程，可以通过fork()系统调用

#include <sys/types.h>

#include <unistd.h>

pid\_t fork(void);

成功调用fork函数会创建一个新的进程，几乎与调用fork的进程一样。在子进程中，成功的fork调用会返回0.在父进程中fork返回子进程的pid。如果出现错误，返回一个负值。

最常见的是fork函数创建一个新的进程，然后使用exec载入二进制映像，替换打当前进程的映像。这种情况下，派生了新的进程，而这个子进程会执行一个新的二进制可执行文件的映像。

**C++STL中的allocator**：

new运算分两个阶段：(1)调用::operator new配置内存;(2)调用对象构造函数构造对象内容

delete运算分两个阶段：(1)调用对象希构函数；(2)掉员工::operator delete释放内存

为了精密分工，STL allocator将两个阶段操作区分开来：内存配置有alloc::allocate()负责，内存释放由alloc::deallocate()负责；对象构造由::construct()负责，对象析构由::destroy()负责。

同时为了提升内存管理的效率，减少申请小内存造成的内存碎片问题，SGI STL采用了两级配置器，当分配的空间大小超过128B时，会使用第一级空间配置器；当分配的空间大小小于128B时，将使用第二级空间配置器。第一级空间配置器直接使用malloc()、realloc()、free()函数进行内存空间的分配和释放，而第二级空间配置器采用了内存池技术，通过空闲链表来管理内存。

**reactor模型的作用**：

reactor模型要求主线程只负责监听文件描述是否有事件发生，有就立即将该时间通知工作线程，除此之外，主线程不做任何其他实质性的工作。读写数据，接收新的连接及处理客户请求都在工作线程中完成。

组成：

1）Handle：即操作系统中的句柄，是对资源在操作系统层面上的一种抽象，它可以是打开的文件、一个连接(Socket)、Timer等。由于Reactor模式一般使用在网络编程中，因而这里一般指Socket Handle，即一个网络连接。

2）Synchronous Event Demultiplexer（同步事件复用器）：阻塞等待一系列的Handle中的事件到来，如果阻塞等待返回，即表示在返回的Handle中可以不阻塞的执行返回的事件类型。这个模块一般使用操作系统的select来实现。

3）Initiation Dispatcher：用于管理Event Handler，即EventHandler的容器，用以注册、移除EventHandler等；另外，它还作为Reactor模式的入口调用Synchronous Event Demultiplexer的select方法以阻塞等待事件返回，当阻塞等待返回时，根据事件发生的Handle将其分发给对应的Event Handler处理，即回调EventHandler中的handle\_event()方法。

4）Event Handler：定义事件处理方法：handle\_event()，以供InitiationDispatcher回调使用。

5）Concrete Event Handler：事件EventHandler接口，实现特定事件处理逻辑。

**C++怎么创建一个线程**：

thread

**（1）用一个初始函数创建一个线程**

C++在运行一个可执行的程序的时候，就创建了一个进程，会自动创建一个主线程，这个主线程和进程同生共死

thread mythread1(print1); //创建一个线程mythread1,print1()是该线程的初始函数（执行函数）

mythread1.join(); //阻塞主线程，等待mythread1这个线程执行完毕在继续执行，推荐这种做法。

**（2）用类的对象创建一个线程**

thread mythread2(t); //调用了拷贝构造函数

**（3）用lambda表达式创建一个线程**

auto mylamthread = []

{//这是一个lambda表达式

cout << "我的线程开始执行了" << endl;

cout << "我的线程执行结束了" << endl;

};thread mythread3(mylamthread); //创建线程

**C++中四种智能指针：**

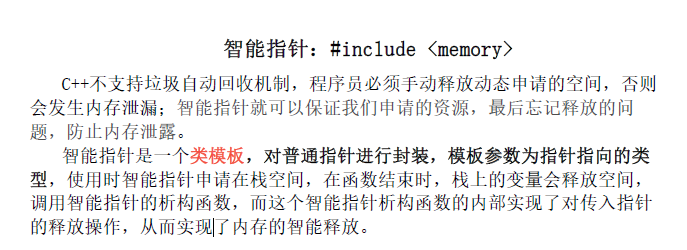
管理在堆上分配内存，智能指针作用是管理一个指针，如果申请的时候忘记了释放，造成内存泄漏。使用智能指针可以避免这个问题，智能指针是一个类，当超出了类的作用域时，类会自动调用析构函数，析构函数会自动释放资源，所以智能指针的作用原理就是在函数结束的时候自动释放内存空间，不需要手动释放空间。

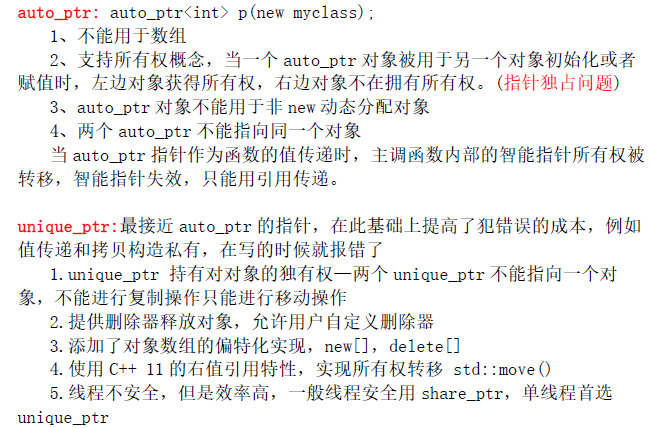
（1）auto\_ptr：采用所有权模式：存在潜在的内存崩溃问题，已被弃用

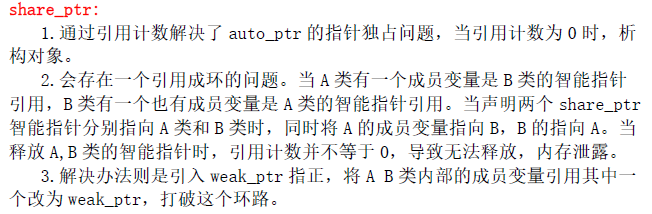
（2）unique\_ptr:独占式拥有或者严格拥有。保证同一时间只有一个智能指针可以指向该对象，对于避免内存泄漏特别有用。

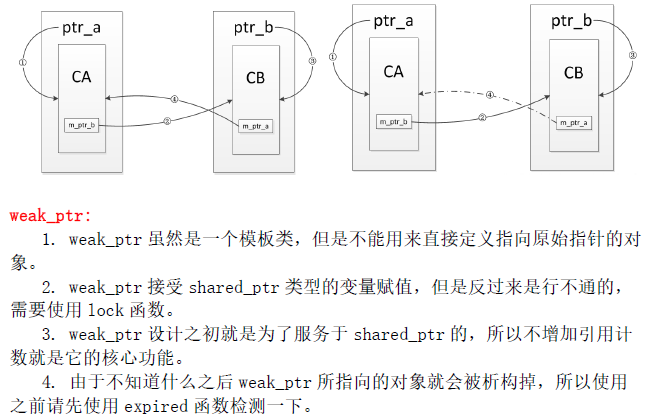
（3）shared\_ptr:为了解决auto\_ptr在对象所有权上的局限性，在使用引用计数的机制上提供了可以共享所有权的智能指针。

（4）weak\_ptr是一种不控制对象生命周期的智能指针，指向一个share\_ptr管理对象，进行管理的是强引用的share\_ptr，引用一个智能指针来辅助。他的构造和析构都不会引起计数的增加或者减少。其是用来解决shared\_ptr相互引用时死锁的问题。如果两个shared\_ptr指针相互引用，那么两个指针的引用计数永远不可能降到0，资源永远不会被释放，会造成内存泄漏。









**深拷贝和浅拷贝**：

//拷贝构造函数调用情形：

（1）一个对象以值传递的方式进入函数体

（2）一个对象以值传递的方式从函数体返回

（3）一个对象需要通过另外一个对象初始化

深拷贝：如果拷贝某个对象的时候为新的对象制作了外部对象的独立拷贝

浅拷贝：如果拷贝的对象中引用了某个外部的内容（例如在堆上分配了数据），那么在复制这个对象的时候，让新旧两个对象指向同一个外部的内容。（新旧两个对象指向外部同一个内容）（有可能造成多次调用析构函数清除某个对象的额空间造成程序崩溃）

//继承类要通过调用基类的拷贝构造函数对基类的私有数据访问，

//拷贝构造函数与赋值函数的区别

（1）拷贝构造函数是用一个对象来初始化一个内存区域，这块内存就是新对象的内存区，赋值函数是对一个已经初始化过的对象的赋值操作

（2）拷贝构造函数是一个构造函数。赋值函数则是把一个对象给一个原有的对象，首先需要吧原有的内存释放，还要检查两者是否是一个对象。

**野指针**：野指针是指向不明的指针，指针创建未初始化，指向“垃圾”内存，未初始化随机指向内存。或者是删除之后没有置为NULL；再次用的时候就是野指针了。一般指的是未初始化。  
**悬垂指针**：指向曾今存在的对象，但是该对象已经不存在了。

**哑指针**：哑指针指传统的C/C++指针，它只是一个指向，除此以外它不会有其他任何动作，所有的细节必须程序员来处理，比如指针初始化,释放等等。

**访问控制权限的区别**：

C++通过pulic，protected，private三个关键字控制成员变量和成员函数的访问权限

在类的内部，无论成员是声明为何种类型，都是可以互相访问的，没有权限限制。在类外定义的，只能通过对象访问成员，并且通过对象只能访问public属性的成员。

**STL迭代器删除元素**：

（1）对于序列容器vector，deque来说，使用erase后，后边的每个元素的迭代器都会失效，但是后边的每个元素都会往前移动一个位置，但是erase会返回下一个有效的迭代器

（2）对于关联容器map和set来说，使用erase后，当前元素的迭代器失效，但是其结构是红黑树，删除当前元素，不会影响到下一个元素的迭代器

（3）对于list来说，其使用了一个不连续的内存分配，并且erase会返回一个有效的迭代器。

**抽象类或者抽象基类**：

包含纯虚函数的类叫做抽象类，其不能被调用，不能实例化对象。抽象类是为一个类族提供一个公共接口。

虽然抽象类不能定义对象，但是可以定义指针，当派生类成为具体的类之后，用指针指向派生类的对象，实现多态性。

**虚函数与多态**:

（1）多态：主要分为静态多态和动态多态

//静态多态：主要是重载，在编译的时候就确定了，是静态绑定

//动态多态：虚函数机制实现的，运行的时候绑定的。

子类如果重写父类的虚函数，在虚函数表中，该函数的地址会被替换，对于存在虚函数的类的对象，在VS中，对象的对象模型的头部存放指向虚函数表的指针，通过该机制实现多态

**虚函数的作用**:

在同一个类中不能定义两个相同名字、参数个数和类型名都相同的函数，否则就是重复定义。但是不同类中可以定义相同的，就需要利用对象去调用类名加限定符再加函数名调用a.B：：get()；

（1）以上做法很不方便，虚函数就是解决动态多态性问题的，

（2）在基类中声明函数是虚拟的，在派生类中正式定义该函；

（3）允许派生类中重新定义与基类同名的函数，并且通过基类指针或者引用来访问基类和派生类中的同名函数。

**纯虚函数的存在必要性**:

（1）纯虚函数没有函数体；

（2）最后是=0；=0只是表示该函数是纯虚函数。

（3）只是一个声明语句，以分号结束。

作用：纯虚函数只有函数名字，没有功能，不能被调用。作用是在基类中为其派生类保留一个函数的名字，以便派生类根据需要对其进行定义，如果基类没有保留函数名字，就无法实现多态性。

如果一个函数在基类中是纯虚函数，派生类中没有对其定义，该纯虚函数仍然是派生类的纯虚函数。

**C++的锁**:

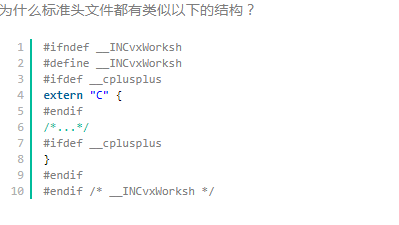
（1）互斥锁

（2）条件变量

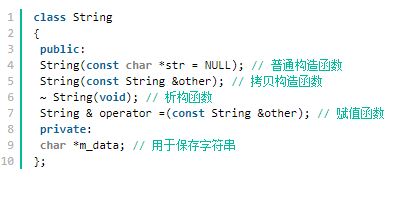
（3）自旋锁

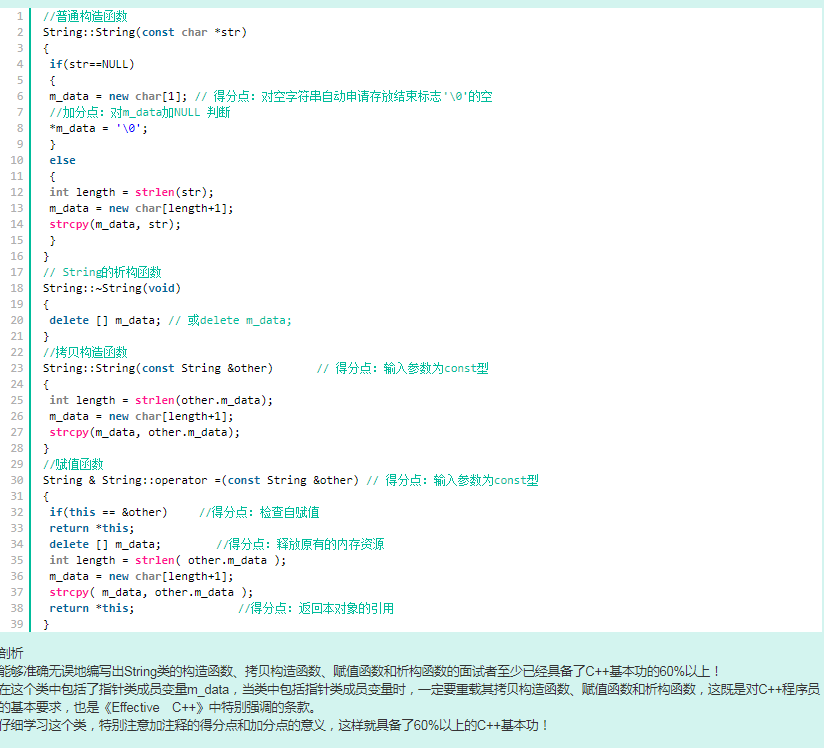
（4）读写锁

**C++中标准的头文件**:

  
file://c:\users\admini~1\appdata\local\temp\tmp24jqpv\2.png  
C++支持函数重载，但是c语言不支持。函数在C++编译之后与C语言的函数名不一样。为了实现混合编程，提供连接符extern "C"，解决名字匹配问题，加上名字之后，就按照C语言的方式进行编译。

**编写string函数的构造函数，析构函数和赋值函数**:





**静态变量的初始化:**

静态变量存储在虚拟地址空间的数据段和bss段，C语言中其在代码执行之前初始化，属于编译期初始化。而C++中由于引入对象，对象生成必须调用构造函数，因此C++规定全局或局部静态对象当且仅当对象首次用到时进行构造

**常量存放的地方**:

常量在C++里的定义就是一个top-level const加上对象类型，常量定义必须初始化。对于局部对象，常量存放在栈区，对于全局对象，常量存放在全局/静态存储区。对于字面值常量，常量存放在常量存储区。

**STL删除操作**:

这个主要考察的是迭代器失效的问题。1.对于序列容器vector,deque来说，使用erase(itertor)后，后边的每个元素的迭代器都会失效，但是后边每个元素都会往前移动一个位置，但是erase会返回下一个有效的迭代器；2.对于关联容器map set来说，使用了erase(iterator)后，当前元素的迭代器失效，但是其结构是红黑树，删除当前元素的，不会影响到下一个元素的迭代器，所以在调用erase之前，记录下一个元素的迭代器即可。3.对于list来说，它使用了不连续分配的内存，并且它的erase方法也会返回下一个有效的iterator，因此上面两种正确的方法都可以使用。

**函数的形参和实参**：

**形参**:定义函数时候的参数,不占存储单元,在调用的时候会分配存储单元,该函数结束释放该内存空间.

**实参变量:**是对形参变量的数据传递是“值传递”,单向传递,只能够由实参传递给形参,不能反向传递,调用的时候系统会给形参分配另外的存储单元,函数结束的时候释放该单元.

//函数返回值与函数类型不同,对于数据型的数据会直接转换成函数类型.

//函数声明:把函数的名字,类型,参数,形参类型,形参个数与顺序.

//递归函数的实现:第一阶段是回溯:由最顶向下回溯到递归边界;然后由边界递推到顶点.

//调用函数需要一定的时间和空间的开销:需要在进入被调函数之前记录当前执行的指令的地址,还要记录相关信息,以便函数调用结束后继续运行.如果函数频繁调用会增加时间开销.因此引入了内联函数。

**Bool型中TRUE和false：ture返回1，false返回0；**

**表达式的真假：0就是假，非0就是真**

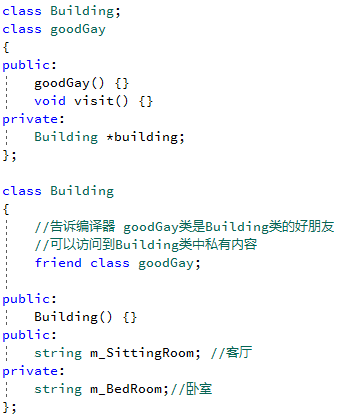
**友元函数**：

让一个函数或者类访问另一个类中的私有成员，友元关键字为friend。

（1）全局函数作为友元函数



（2）类作为友元类



（3）成员函数作为



**宏定义：**

容易产生错误，二义性，也可以定义表达式。

**内联函数**：对应于宏定义

内联函数:在函数声明(定义)时,函数类型前面加上inline;系统在编译的时候遇到调用该函数就会用该函数的函数体去替换掉函数的调用;会节省时间,但是增加了空间,而且内联函数不能包括循环语句和选择switch.

内联函数和宏的区别：

（1）都是在编译的时候展开

（2）内敛函数是嵌入目标代码中，宏只是一个简单的文本替换

（3）内敛函数具有检查，编译的功能，宏没有

（4）宏不是函数，inline势函数

（5）宏定义的括号容易出现二义性，内联函数不会出现二义性。

内联函数只有加在函数定义才有用，加在函数声明不算：

内联函数的作用：代替宏定义解决程序中函数调用的效率问题。

定义在类声明之中的成员函数将自动称为内联函数：

内联函数的优点：

（1）函数代码被放入符号表中，使用的时候直接替换，没有调用的开销，效率高

（2）内联函数也是一个真正的函数，编译器在调用一个内联函数的时候，会先检查它的参数类型，保证调用的正确。

（3）inline可以作为某个类的成员函数，当然就可以在其中使用所在类的保护成员及私有成员。

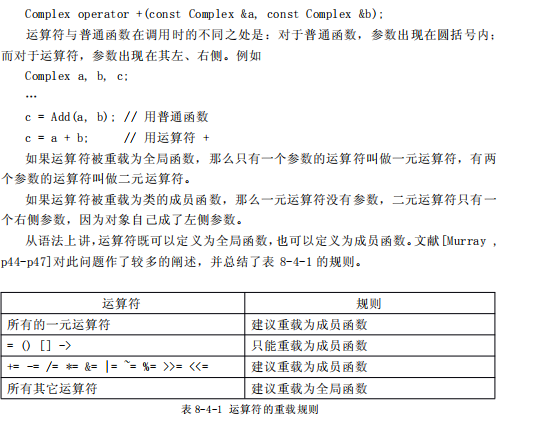
（4）内联函数能够提高执行效率，但是仅仅是省去了函数调用的开销，其实相当于复制代码到各个地方，会使得程序很长，空间消耗大。下列情况不适合用内联函数：

1、如果函数体内代码较长；

2、如果函数体内有循环。

3、构造函数和析构函数一般不能内联，放在类体外定义。

**运算符重载**：



**句柄与指针**：

//句柄指的是指向指针的指针

（1）句柄的指向可以是个很复杂的结构，与系统相关，

（2）指针也可以指向复杂结构，但大部分是用户定义的。

**常量：**

(1)数值常量

(2)布尔常量

(3)字符常量:单引号,转义字符

(4)字符串常量:双引号

(5)#define

(6)const

**存储类**：

(1)auto:声明变量的时候,根据初始化表自动推断该变量的类型,声明函数的时候,函数返回占位符

(2)register:存储在寄存器中,而不是RAM中的局部变量.变量的最大尺寸等于寄存器的带下,不能对其使用一元运算符'&',因为他没有内存的位置.

(3)static:

(4)extern:全局变量的引用,对所有程序文件课件.

(5)mutable:说明符仅适用于类的对象,允许对象成员代替常量,mutable成员可以通过const成员函数修改

(6)thread\_local:说明符声明的变量仅可在他其上创建的线程上访问.变量在创建线程时创建,并且在销毁的时候销毁,每个线程都有自己的副本.

**C++空类会产生那些成员函数**：

（1）默认的构造函数

（2）默认的析构函数

（3）默认的拷贝构造函数

（4）默认的赋值函数：引用，不产生另外的空间

（5）取址运算：对类的对象取地址

**构造函数执行顺序**：

（1）任何虚拟基类的构造函数按照他们继承的顺序构造

（2）任何非虚拟基类的构造函数按照他们被构造的顺序构造

（3）任何成员对象构造按照他们的声明顺序调用

（4）类自身的构造函数。

**虚函数与纯虚函数的区别**：

（1）类里面如果声明了虚函数，这个类是实现的，哪怕是空实现，作用是为了让这个函数被其子类所覆盖，编译器就可以后期绑定来达到多态，纯虚函数知识一个接口，是个函数声明，留给子类实现。

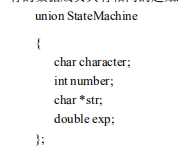
（2）虚函数在子类里面也可以不重载；但是纯虚函数必须在子类中实现。

（3）虚函数的类用于“实作继承”，继承接口同时也继承了父类的实现。纯虚函数的类用于“介面继承”纯虚函数关注的是接口的统一性，实现由子类完成。

（4）带纯虚函数的类叫做虚基类，只能被继承。也叫抽象类。

**Union关键字**：

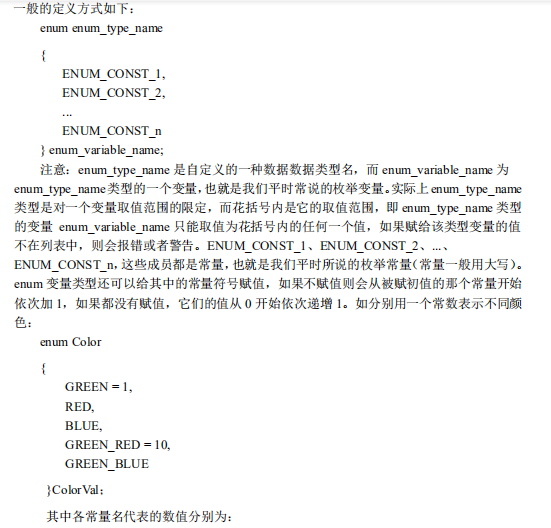
union：空间只能放置多个数据成员的“一种”，而不是为每个数据成员配置空间，在union中所有的数据成员共用一个空间，同一时间值存储其中一个数据成员，所有的数据成员具有相同的起始地址。



一个union只配置一个足够大的空间来容纳最大长度的数据成员，上述中最大的数据成员为double型，所以union的空间大小为double数据类型的大小。union默认的属性为public，union主要用来压缩空间。

重点：union为空的时候，和结构体一样，sizeof（union a）=1；其余的时候以最大的为主。

**Enum枚举类型**：



（1）#define是在编译的时候简单替换，枚举常量是在编译的时候确定其值

（2）一般可以调试枚举常量，但是不能调宏常量

（3）枚举可以一次定义大量相关的常量，宏只能定义一个。

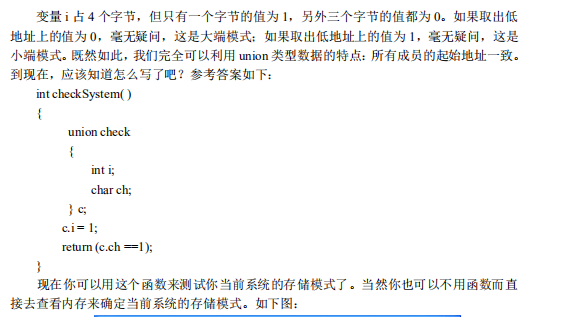
sizeof（ColorVal）=4，相当于一个int型的常量。无论这个枚举类型是不是空都是4.枚举变量是当做整型常量来处理的。

**大端与小端**：

大端模式：字数据的高字节存储在低地址中，字数据的低字节存放在高地址中。

小端模式：字数据的高字节存储在高地址中，字数据的低字节存放在低地址中。

可以利用union来判断。



int是4个字节，而char是一个字节，都是从存储低地址开始读取，另c.i=1，相当于在4个字节的地址中存放了1，然后从低地址取出，若果取出为1，则是小端存储。

**Malloc分配空间**：

//malloc调用形式：（类型\*）malloc（size）：分配空间大小返回首地址，内存未被初始化。

//calloc：调用形式：（类型\*）calloc（n，size）：分配n快长度为size的空间连续空间，返回首地址，内存初始化为0

//realloc：调用形式：（类型\*）realloc（\*ptr，size）：增大ptr内存增大到size的大小，新增的内存没有初始化

**This指针**：

（1）this指针只能在成员函数中用，全局函数和静态函数都不能用。

（2）this指针成员函数的开始前构造，在成员结束后清除

（3）可能存在堆区，栈区，全局变量等

**成员的初始化**：不管是初始化还是继承的初始化，一定是先定义的先初始化，与参数列表中的顺序无关。

**只能使用初始化列表的情况**：

(1)对于const和refreence的变量，只能进行初始化操作，不能赋值，因此只能采用初始化列表

（2）类的构造函数需要调用基类的构造函数，初始化基类构造函数的时候。

**多重继承**：

C++允许一个派生类有多个基类，容易出现二义性。

（1）加上全局作用符明确调用

（2）虚拟继承解决二义性，将共有的基类都定义成虚拟继承

多重继承的优点：

（1）对象可以调用多个基类的接口

（2）容易出现继承的二义性。

**Strlen():函数得到长度不包括‘\0’**；

**Sizeof()：**

是个关键字：得到的是空间大小，包括了‘\0’；

//联合体的大小取决于成员中所占内存最大的字节，但是也必须与最大的空间进行对齐

如：对齐是为了提高读取效率。

union u3

{

char a[13];

int b;

}

//size的大小为16，因为char a为13个字节，但是要和内部最大的类型对齐，就是16.

#pargma pack(2)//设置对齐字节，就是最大的对齐字节设置

int \*p=NULL;

int a[100];

int b[100];

cout<<sizeof(p)<<endl;//指针变量定义就是4个字节

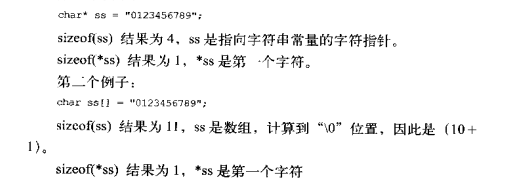
cout<<sizeof(\*p)<<endl;//int型变量4个字节

cout<<sizeof(a)<<endl;//表示整个数组的字节数400

cout<<sizeof(a[100])<<endl;//表示第下标为100的数组的字节数为4

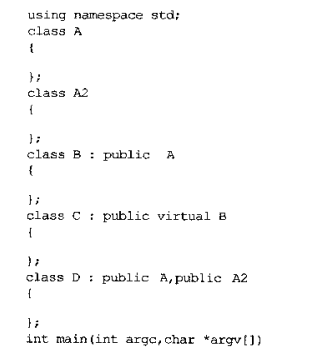
cout<<sizeof(&a)<<endl;//a的引用，创建的一个int型

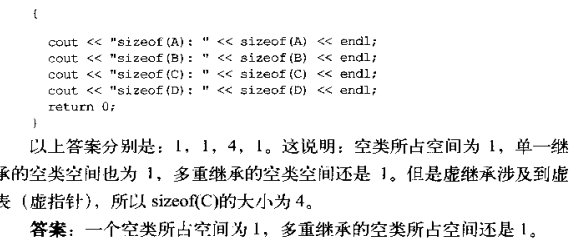
cout<<sizeof(&a[0])<<endl;//类似于上



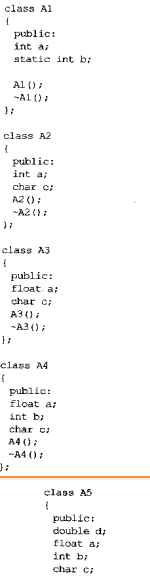
//如果是数组名，就是整个数组的空间大小；如果是指针就是指针变量的空间大小

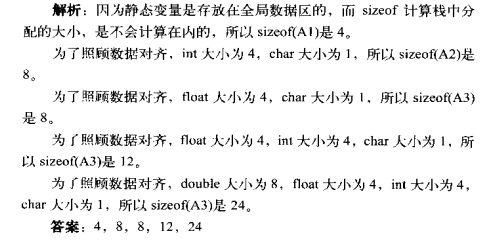
空类和多重继承的大小:





几个虚继承就多几个指针，静态成员不占类的空间大小。





**指针的一些定义：**

指针在后，表示指向某个东西的指针

指针常量：指针的常量，重点在于常量，int const \*p；

常量指针：指向常量的指针，重点在于指针，const int \*p；

指针数组：重点是数组，数组中存放的都是同类型的指针 int \*a[11]

数组指针：重点在于指针，指向数组的指针，int \*b=new int[]；

指针函数：函数的返回值是指针。int \*fun（）；

函数指针：指向函数的指针，int（\*fun）（）；