[1、C和C++的区别 ?](https://link.zhihu.com/?target=http%3A//www.k6k4.com/simple_question/qshow/aaqtotpmw1537796355161)

1. C面向过程，结构化语言，考虑过程如何处理输入得到输出
2. C++面向对象，主要特征是（封装、继承、多态）:

封装：隐藏细节，模块化

继承：派生类可以继承父类数据和方法，扩展已经存在的模块，实现代码重用

多态：一个接口、多种实现。通过派生类重写父类虚函数，实现接口复用。基类指针指向派生类对象

1. 动态管理内存方法不同：C为malloc和free，C++为new和delete
2. C++支持函数重载，有引用的概念

[2、#include<file.h> #include "file.h" 有什么区别?](https://link.zhihu.com/?target=http%3A//www.k6k4.com/simple_question/qshow/aaqyszblj1538098274562)

1）<>从标准路径开始寻找，””从当前工作路径开始寻找

[3、C++文件编译与执行的四个阶段?](https://link.zhihu.com/?target=http%3A//www.k6k4.com/simple_question/qshow/aaqjjviar1537799714129)

1. 预处理：根据文件中的预处理指令来修改源文件内容
2. 编译：编译成汇编
3. 汇编：汇编代码翻译成目标机器指令
4. 链接：链接目标代码生成可执行文件

[4、堆和栈有什么区别?](https://link.zhihu.com/?target=http%3A//www.k6k4.com/simple_question/qshow/aaqvppxqu1538096999381)

1. 栈：存放函数的参数、局部变量，由编译器自动分配释放

堆：由new分配内存块，由程序控制，delete手动释放内存。否则程序结束后操作系统自动回收

1. 堆的内存分配需要频繁new delete，所以内存空间不连续，有大量碎片
2. 堆的生长空间向上，地址越大，栈的生长空间向下，地址越小

[5、深拷贝和浅拷贝的区别 ?](https://link.zhihu.com/?target=http%3A//www.k6k4.com/simple_question/qshow/aaqjjxlby1537800183617)

1）比如一个类拥有资源，当类发生复制时，如果资源重新分配就是深拷贝；没有重新分配资源就是浅拷贝

[6、哪些成员函数不能被继承?](https://link.zhihu.com/?target=http%3A//www.k6k4.com/simple_question/qshow/aaqcgzgcl1537799336958)

1. 构造函数

子类可以继承成员变量和方法。创建子类对象时，会调用父类构造方法初始化从父类继承来的数据成员。若无显式构造函数，系统会生成默认构造函数。

1. 析构函数

子类析构函数调用父类析构函数析构父类的变量

1. 赋值运算符重载函数

只是在子类的赋值运算法重载函数中会调用父类的赋值运算符重载函数

[7、基类的析构函数为什么要用virtual虚析构函数?](https://link.zhihu.com/?target=http%3A//www.k6k4.com/simple_question/qshow/aaqtbepkm1538099105453)

1. 防止内存泄露：delete p（基类）的时候，他会先执行派生类的析构函数，然后执行基类析构函数。若基类析构函数不是虚函数，当删除基类指针指向的派生类对象时就不会触发动态绑定，因此只会调用基类的析构函数，不会调用派生类的析构函数。
2. 动态绑定：基类指针指向派生类对象（实现了多态），虚函数是动态绑定的基础
3. 基类非虚析构函数也可能不会内存泄露，只需要派生类对象用派生类指针指向即可，但这就没有实现C++的多态

[8、哪些函数不能声明成虚函数?](https://link.zhihu.com/?target=http%3A//www.k6k4.com/simple_question/qshow/aaqonwiox1537798121846)

1. 非成员函数

只能重载（overload），不能继承（override），而虚函数主要作用是在继承中实现动态多态，非成员函数早在编译期间就已经绑定函数了，无法实现多态

1. 构造函数

调用虚函数需要通过“虚函数表”来进行，但虚函数需要对象实例化之后才能进行调用。而在构造函数运行期间，还没有为虚函数表分配空间，自然没法调用虚函数

1. 静态成员函数

每个类只有一份，所有对象共享，它是属于类。虚函数必须根据对象类型才能知道调用哪一个虚函数，故虚函数是一定要在对象基础上才可以，两者一个与实例相关，一个与类相关

1. 内联成员函数

内联函数是为了在代码中直接展开，减少函数调用话费的代价

虚函数是为了继承之后对象能够准确执行自己的动作，并在编译时被展开，虚函数在运行时才能动态绑定函数

1. 友元函数

C++不支持友元函数的继承，所以对于没有继承特性的函数没有虚函数的说法。友元函数不属于类的成员函数，不能被继承

[9、多态，虚函数，纯虚函数?](https://link.zhihu.com/?target=http%3A//www.k6k4.com/simple_question/qshow/aaqwsnzst1537796688271)

1. 多态

不同对象接收相同消息时产生不同动作，体现在两个方面：

运行：通过继承和虚 体现

编译：函数和运算符重载

1. 虚函数

基类中 virtual成员函数。它提供一种接口与界面，允许在派生类中对基类的虚函数重载

1. 纯虚函数

基类中为派生类保留一个函数名字，以便派生类根据需要进行定义。作为接口，纯虚函数不具备函数功能，一般不能被直接调用

基类继承来的纯虚函数，在派生类中仍是虚基类。如果一个类至少有一个纯虚函数，则此类为抽象类

抽象类中可以包括纯虚函数和虚函数。抽象类必须用作派生其他类的基类，不能直接创建对象实例。但可使用指向抽象类的指针支持运行时的多态

[10、如何初始化const和static数据成员?](https://link.zhihu.com/?target=http%3A//www.k6k4.com/simple_question/qshow/aaqehuwab1538099378406)

1. static：类外，类的内部只是声明。类的静态成员函数和成员变量和类相关，不和对象相关

static const的整型（bool，char，int，long）可在类中，其他类型在类外（包括整型数组）

const：不能在类定义处初始化，只能通过构造函数初始化列表进行，必须有构造函数。const数据成员属于对象，对整个类是可变的。同一类的不同对象的const可以不一样。所以不能在类的声明中初始化，因为此时还没有对象。

要想建立整个类都恒定的常量，应该用类的枚举变量实现，或者static const

[11、关键字static有什么作用?](https://link.zhihu.com/?target=http%3A//www.k6k4.com/simple_question/qshow/aaqoluiwy1538097538950)

1. 函数体内：作用范围在此函数体，但是内存在堆上，所以下次调用次函数时还会维持上次的值
2. 模块类：修饰全局变量或者函数，使用范围限制在声明他的模块内，不能被模块外函数访问
3. 类中：修饰成员变量，表示变量属于整个类，对类的所有对象只有一份拷贝

修饰成员函数，表示函数为整个类所有，不接受this指针，指正访问类中的static成员变量

1. 与const区别：const强调值不能被修改，static强调对所类的所有对象只有唯一拷贝

[12、#define和const有什么区别?](https://link.zhihu.com/?target=http%3A//www.k6k4.com/simple_question/qshow/aaquqxqbg1537797080552)

1. #define定义的常量没有类型，给出的是一个立即数；const定义的常量有类型，存放在静态区
2. 处理阶段不同：#define定义的宏变量在预处理时进行替换，可能有多个拷贝；const定义的变量在编译时确定其值，只有一个拷贝
3. #define定义的常量不可以用指针去指向，const定义的常量可以用指针去指向该常量的地址
4. #define定义简单的函数，const不可以定义函数

[13、const 的作用?](https://link.zhihu.com/?target=http%3A//www.k6k4.com/simple_question/qshow/aaqtapdnc1537796042000)

1. 定义制度变量，即常量
2. 修函数的参数和返回值
3. 修饰函数的定义体，这里的函数为类的成员函数，被const修饰的成员函数代表不修改成员变量的值

[14、结构体struct和共同体union（联合）的区别 ?](https://link.zhihu.com/?target=http%3A//www.k6k4.com/simple_question/qshow/aaqderkcm1537796593391)

1. 结构体：不同类型数据组合成一个整体，是自定义类型。每个成员有独立地址，同时存在

Union：不同类型的几个变量共享一段内存。所有成员不同时存在

2） sizeof(struct)是内存对齐后所有成员长度的总和，sizeof(union)是内存对齐后最长数据成员的长度

[15、数组和指针有什么区别?](https://link.zhihu.com/?target=http%3A//www.k6k4.com/simple_question/qshow/aaqknxojs1538097221310)

1. 修改内容上差别：

char a[] = "hello";

a[0] = 'X';

char \*p = "world"; // 注意p 指向常量字符串

p[0] = 'X'; // 编译器不能发现该错误，运行时错误

1. sizeof可以计算数组的容量，sizeof(p)得到指针变量的字节数，C/C++无法知道指针所指内存容量，除非申请时记住。
2. 数组作为函数参数传递时，数组自动退化为同类型指针

[16、指针和引用的区别?](https://link.zhihu.com/?target=http%3A//www.k6k4.com/simple_question/qshow/aaqcgbvxl1537796199204)

1. 引用是变量的别名，内部实现是只读指针
2. 只能在初始化时被赋值，其他时候值不能改变，指针的值可以在任何时候被改变
3. 引用不能为NULL，指针可以为NULL
4. 引用变量内存单元保存的是被引用变量的地址
5. Sizeof引用=指向变量的大小，sizeof指针=指针本身的大小
6. 引用可以取地址操作，返回的是被引用变量本身所在的内存单元地址
7. 引用使用在源代码级相当于普通的变量一样使用，做函数参数时，内部版传递的实际是变量地址

[17、delete与 delete []区别?](https://link.zhihu.com/?target=http%3A//www.k6k4.com/simple_question/qshow/aaqazczgn1537785890364)

1. delete只调用一次析构函数、delete[]调用每一个成员的析构函数。
2. delete-new，delete[]-new []

[18、new、delete、malloc、free关系?](https://link.zhihu.com/?target=http%3A//www.k6k4.com/simple_question/qshow/aaqfuvdah1537785722039)

1. malloc/free是C++/C语言的标准库函数，new/delete是C++的运算符
2. delete调用析构函数，new调用构造函数
3. 都可用于申请动态内存和释放内存
4. 对于非内部数据类型对象而言，malloc/free无法满足动态对象要求，他是库函数不是运算符，不在编译器控制权限之内，不能执行构造函数和析构函数。所以C++需要new/delete完成后动态内存分配和初始化工作的运算符new，以及清理释放内存工作的delete

[19、简单介绍vector内存分配方式?](https://link.zhihu.com/?target=http%3A//www.k6k4.com/simple_question/qshow/aaqgjbelf1539305201719)

1）先申请一定大小的数组。数组填满之后，另外申请一块原数组两倍大的新数组，然后将原数组数据拷贝到新数组，最后释放原数组

[20、STL里有哪些map?](https://link.zhihu.com/?target=http%3A//www.k6k4.com/simple_question/qshow/aaqrsmxws1539305067045)

1. map、unordered\_map、hash\_map
2. 空间复杂度:hash\_map<unordered\_map<map

查找和插入效率:unordered\_map>hash\_map>map

3) 使用unordered\_map代替hash\_map:

标准化的推进

速度差不多，但是unorderedd\_map支持string和复杂对象作为key

[21、STL是如何进行内存管理的?](https://link.zhihu.com/?target=http%3A//www.k6k4.com/simple_question/qshow/aaqrujdlj1539304787761)

[22、红黑树的特性与其在C++ STL中的应用 ?](https://link.zhihu.com/?target=http%3A//www.k6k4.com/simple_question/qshow/aaqgrjvzy1539304663725)

1. map、set、multimap、multiset的底层实现，epoll模型的底层数据结构，linux系统中CFS进程调度算法
2. 特性：根节点为黑色

不能有两个连续红节点

空指针是黑色

从任意一个节点触发，到后代中空指针的路径上，均包含相同数量的黑色节点

[23、STL中map和set的原理（关联式容器）?](https://link.zhihu.com/?target=http%3A//www.k6k4.com/simple_question/qshow/aaqwjqmdf1538097769282)

1）底层实现都是红黑树

[24、STL中unordered\_map和map的区别 ?](https://link.zhihu.com/?target=http%3A//www.k6k4.com/simple_question/qshow/aaqatjaok1537799983498)

1. map底层为红黑树，内部数据有序，查询、插入、删除时间复杂度为O(logN)
2. unordered\_map底层是防冗余的哈希表，内部无序

[25、STL中的vector的实现，是怎么扩容的?](https://link.zhihu.com/?target=http%3A//www.k6k4.com/simple_question/qshow/aaqaihqwa1537799907441)

[26、简单介绍一下STL库的容器和算法?](https://link.zhihu.com/?target=http%3A//www.k6k4.com/simple_question/qshow/aaqhstvdw1537797316583)

1. 容器：存储数据地方，比如array、vector，分为序列式容器和关联式容器

序列式容器：元素不一定有序，但可以被排序，birivector，list，deque，slist

关联式容器：内部为平衡二叉树，每个元素为key-value，比如mao，set，hashtable，hash\_set

1. 算法有排序，复制等，以及各个容器的特定算法
2. 迭代器是STL精髓，提供了一个类指针，能够按照顺序访问某个容器内的各个元素，但无需暴露容器的内部结构，将容器和算法分开，独立设计

[27、vector和list的区别?](https://link.zhihu.com/?target=http%3A//www.k6k4.com/simple_question/qshow/aaqcenzuk1539305401334)

1. vector和数组类似，拥有一段连续内存空间，其实地址不变，能够高效随机存取。但插入删除会造成内存块拷贝，时间复杂度O(n)。数组内存不够时，还要重新申请二倍空间进行内存拷贝
2. list：双向链表，内存空间不连续。查询效率O(n)，插入删除O(1)

[28、解释静态绑定和动态绑定?](https://link.zhihu.com/?target=http%3A//www.k6k4.com/simple_question/qshow/aaqujgcod1537800110594)

展现了C++的多态

1. 对象的静态类型和动态类型

静态类型：对象在声明时采用的类型，编译时确定

动态类型：当前对象所指的类型，运行期间确定

对象的动态类型可变，静态类型无法改变

1. 静态绑定和动态绑定

静态绑定：绑定的是对象的静态类型，函数依赖于对象的静态类型，在编译期确定

动态绑定：绑定的是对象的动态类型，函数依赖于对象的动态类型，在运行期确定

只有虚函数才使用动态绑定，其他全是静态绑定

[29、进程间通信方式和线程间通信方式 ?](https://link.zhihu.com/?target=http%3A//www.k6k4.com/simple_question/qshow/aaqccbpng1537799083452)

**（1）进程间通信方式：**

# 管道( pipe )：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程间使用。进程的亲缘关系通常是指父子进程关系。

# 信号量( semophore ) ： 信号量是一个计数器，可以用来控制多个进程对共享资源的访问。它常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源。因此，主要作为进程间以及同一进程内不同线程之间的同步手段。

# 消息队列( message queue ) ： 消息队列是由消息的链表，存放在内核中并由消息队列标识符标识。消息队列克服了信号传递信息少、管道只能承载无格式字节流以及缓冲区大小受限等缺点。

# 共享内存( shared memory ) ：共享内存就是映射一段能被其他进程所访问的内存，这段共享内存由一个进程创建，但多个进程都可以访问。共享内存是最快的 IPC 方式，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号两，配合使用，来实现进程间的同步和通信。

# 套接字( socket ) ： 套解口也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同及其间的进程通信。

**（2）线程间通信方式：**

#全局变量；

#Messages消息机制；

#CEvent对象（MFC中的一种线程通信对象，通过其触发状态的改变实现同步与通信）。

[30、描述内存分配方式以及它们的区别?](https://link.zhihu.com/?target=http%3A//www.k6k4.com/simple_question/qshow/aaqsgnkmu1537797464930)

1） 从静态存储区域分配。内存在程序编译的时候就已经分配好，这块内存在程序的整个运行期间都存在。例如全局变量，static 变量。

2） 在栈上创建。在执行函数时，函数内局部变量的存储单元都可以在栈上创建，函数执行结束时这些存储单元自动被释放。栈内存分配运算内置于处理器的指令集。

3） 从堆上分配，亦称动态内存分配。程序在运行的时候用malloc 或new 申请任意多少的内存，程序员自己负责在何时用free 或delete 释放内存。动态内存的生存期由程序员决定，使用非常灵活，但问题也最多。

31.虚函数、纯虚函数（为了实现多态）

1）虚函数是为了允许用基类的指针来调用子类的这个函数。函数可以被实现

2）纯虚函数是为了实现一个接口，起到一个规范的作用，规范继承这个类的程序员必须实现这个函数。含纯虚函数的为抽象类，不能生成对象。

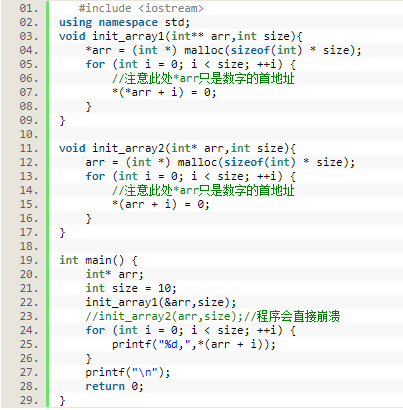
[32.智能指针（shared\_ptr,unique\_ptr,weak\_ptr）](http://c.biancheng.net/view/7898.html)

1. 适当时机释放分配的内存：底层引用计数。智能指针在申请对内存空间的同时，会配备诶个整形值（初始值为1），每当有新对象引用此堆内存时，整形值+1；反之，每当使用此堆内存的对象被释放时，该整形值-1.当堆空间对应的整形值为0时，即表明不再有对象使用它，该堆空间就会被释放。
2. Shared\_ptr:引用计数

Unique\_ptr：只有一个智能指针可以拥有它，可以用move转移所有权

33.strcpy函数

**34. Ｃ语言**[**malloc**](https://blog.csdn.net/qq91752728/article/details/79988210)**和指针**[**传递**](https://blog.csdn.net/qq91752728/article/details/79988210)**陷阱**



malloc为指针分配内存，但是init\_array2传递的是指针，所以函数会生成一个指针的副本，指针及其副本都指向同一个地址。但是malloc会重新对地址进行赋值，所以指针副本指向的地址就不再是原指针指向的地址

35. const char \* 和 [char](https://blog.csdn.net/u013271921/article/details/45030633) const \* 和 char \* const 区别

char\* const不允许修改指针本身，不能再指向其他地方,把指针当常量。另外两个不允许修改指针指向的地址的值，把值当常量

36. C++内存分配方式详解(堆、栈、自由存储区、全局/静态存储区和常量存储区)

37.C++运算符重载限制

1）重载后的运算符至少有一个是自定义的类对象，这可以防止用户为标准类型重载

2）使用运算符不能违背原来的使用规则，不能修改运算符优先级，不能创建新运算符（operator\*\*求幂是错误的）

3）不能重载的运算符; sizeof（长度运算符，p++相当于p+sizeof(A)，如果重载了就不能保证字符+的长度） . .\*（成员指针访问符，保证成员访问的功能不变） typeid :: ?:（重载后无法保证执行两个） const\_cast dynamic\_cast static\_cast reinterpret\_cast

4)只能用成员函数重载: [] () -> =，防止出现 1[x] 1(x) 1->x 1=x

38.#ifndef #define #endif作用

1）为了防止头文件的重复包含和编译

如果两个c++文件都include了同一个头文件，当这两个C++文件编译成一个可执行文件时，就会出现声明冲突

39.extern “C”作用

C++调用c语言编写的函数时都需要加extern “C”。实现c++正确调用c语言代码。加了extern “C”后，会指示编译器按照c语言进行编译，而不是c++。因为c++支持函数重载，所以编译函数的过程中会将函数的参数类型加到编译后的代码中，而不仅仅是函数名；而c语言不支持函数重载，所以编译的函数代码不会带上函数的参数类型。

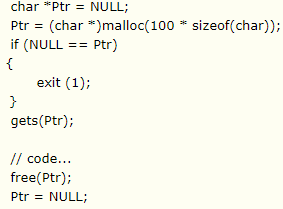
1. 单一语句：extern “C” double sqrt(double)
2. 复合语句：extern “C”{fun1;func2..}
3. 包含头文件，相当于头文件中的声明都加了extern “C” {#include <cmath>}
4. 不可将extern “C”添加在函数内部
5. 如果函数有多个声明，可以都加extern “C”，也可只在第一次声明中加，后面的生命会接收第一个链接指示符的规则

40.malloc和free详解

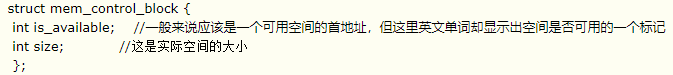
1） void \*malloc(long NumBytes)：该函数分配了NumBytes个字节，并返回了指向这块内存的指针。如果分配失败（空间不足），则返回一个空指针（NULL）。

void free(void \*FirstByte)： 该函数是将之前用malloc分配的空间还给程序或者是操作系统，也就是释放了这块内存，让它重新得到自由。

函数用法

2）malloc之后丢给free  


1. malloc申请之后必须检查是否分配成功；不需要使用申请内存的时候记得释放，并将指向这块内存的指针指向NULL；malloc和free要配对使用，不能多次free，不过多次free(NULL)无所谓
2. malloc的内存从堆里获得，函数返回的指针指向堆里面的一块内存。操作系统中有一个记录空闲内存地址的链表。当操作系统收到程序申请时，就遍历此链表，找到第一个大于所申请空间的堆节，然后将该节点从空闲节点链表中删除，并将该节点的空间分配给程序。
3. free释放的是指针指向的内存空间
4. free函数只有申请空间的指针，这是因为malloc申请时实际占用的内存要比申请的大，而超出的空间就是用来记录对这块内存的管理信息（如分配块的长度，指向下一个分配块的指针等）。C语言中记录信息的结构体为



free()源码

