**[常见C++面试题及基本知识点总结（一）](http://www.cnblogs.com/LUO77/p/5771237.html)**

**1. 结构体和共同体的区别。**

定义：

结构体struct：把不同类型的数据组合成一个整体，自定义类型。

共同体union：使几个不同类型的变量共同占用一段内存。

地址：

struct和union都有内存对齐，结构体的内存布局依赖于CPU、操作系统、编译器及编译时的对齐选项。

关于内存对齐，先让我们看四个重要的基本概念：

1.数据类型自身的对齐值：

对于char型数据，其自身对齐值为1，对于short型为2，对于int,float,double类型，其自身对齐值为4，单位字节。

2.结构体或者类的自身对齐值：其成员中自身对齐值最大的那个值。

3.指定对齐值：#pragma pack(n)，n=1,2,4,8,16改变系统的对齐系数

4.数据成员、结构体和类的有效对齐值：自身对齐值和指定对齐值中小的那个值。

 常见数据类型及其长度：

注意long int和int一样是4byte，long double和double一样是8byte。

在标准c++中，int的定义长度要依靠你的机器的字长，也就是说，如果你的机器是32位的，int的长度为32位，如果你的机器是64位的，那么int的标准长度就是64位。



从上面的一段文字中，我们可以看出，首先根据结构体内部成员的自身对齐值得到结构体的自身对齐值（**内部成员最大的长度**），如果没有修改系统设定的默认补齐长度4的话，取较小的进行内存补齐。

结构体struct：不同之处，stuct里每个成员都有自己独立的地址。sizeof(struct)是内存对齐后所有成员长度的加和。

共同体union：当共同体中存入新的数据后，原有的成员就失去了作用，新的数据被写到union的地址中。sizeof(union)是最长的数据成员的长度。

总结： struct和union都是由多个不同的数据类型成员组成, 但在任何同一时刻, union中只存放了一个被选中的成员, 而struct的所有成员都存在。在struct中，各成员都占有自己的内存空间，它们是同时存在的。一个struct变量的总长度等于所有成员长度之和。在Union中，所有成员不能同时占用它的内存空间，它们不能同时存在。Union变量的长度等于最长的成员的长度。对于union的不同成员赋值, 将会对其它成员重写, 原来成员的值就不存在了, 而对于struct的不同成员赋值是互不影响的。

**2.static 和const分别怎么用，类里面static和const可以同时修饰成员函数吗。**

 static的作用：

对变量：

1.局部变量：

在局部变量之前加上关键字static，局部变量就被定义成为一个局部静态变量。

  1）内存中的位置：静态存储区

  2）初始化：未经初始化的全局静态变量会被程序自动初始化为0（自动对象的值是任意的，除非他被显示初始化）

  3）作用域：作用域仍为局部作用域，当定义它的函数或者语句块结束的时候，作用域随之结束。

 注：当static用来修饰局部变量的时候，它就**改变了局部变量的存储位置（从原来的栈中存放改为静态存储区）及其生命周期（局部静态变量在离开作用域之后，并没有被销毁，而是仍然驻留在内存当中，直到程序结束，只不过我们不能再对他进行访问），但未改变其作用域。**

2.全局变量

在全局变量之前加上关键字static，全局变量就被定义成为一个全局静态变量。

1）内存中的位置：静态存储区（静态存储区在整个程序运行期间都存在）

2）初始化：未经初始化的全局静态变量会被程序自动初始化为0（自动对象的值是任意的，除非他被显示初始化）

3）作用域：全局静态变量在声明他的文件之外是不可见的。准确地讲从定义之处开始到文件结尾。

注：static修饰全局变量，**并为改变其存储位置及生命周期，而是改变了其作用域，使当前文件外的源文件无法访问该变量**，好处如下：（1）不会被其他文件所访问，修改（2）其他文件中可以使用相同名字的变量，不会发生冲突。**对全局函数也是有隐藏作用。**

对类中的

　　　　1.成员变量

　　　　用static修饰类的数据成员实际使其成为类的全局变量，会被类的所有对象共享，包括派生类的对象。因此，**static成员必须在类外进行初始化(初始化格式： int base::var=10;)，而不能在构造函数内进行初始化，不过也可以用const修饰static数据成员在类内初始化 。**

　　　　特点：

* 1. 不要试图在头文件中定义(初始化)静态数据成员。在大多数的情况下，这样做会引起重复定义这样的错误。即使加上#ifndef #define #endif或者#pragma once也不行。
  2. 静态数据成员可以成为成员函数的可选参数，而普通数据成员则不可以。
  3. 静态数据成员的类型可以是所属类的类型，而普通数据成员则不可以。普通数据成员的只能声明为 所属类类型的指针或引用。

2.成员函数

* 1. 用static修饰成员函数，使这个类只存在这一份函数，所有对象共享该函数，不含this指针。
  2. 静态成员是可以独立访问的，也就是说，无须创建任何对象实例就可以访问。base::func(5,3);当static成员函数在类外定义时不需要加static修饰符。
  3. 在静态成员函数的实现中不能直接引用类中说明的非静态成员，可以引用类中说明的静态成员。因为静态成员函数不含this指针。

**不可以同时用const和static修饰成员函数。**

C++编译器在实现const的成员函数的时候为了确保该函数不能修改类的实例的状态，会在函数中添加一个隐式的参数const this\*。但当一个成员为static的时候，该函数是没有this指针的。也就是说此时const的用法和static是冲突的。

我们也可以这样理解：两者的语意是矛盾的。**static的作用是表示该函数只作用在类型的静态变量上，与类的实例没有关系；而const的作用是确保函数不能修改类的实例的状态**，与类型的静态变量没有关系。因此不能同时用它们。

const的作用：

 1.限定变量为不可修改。

2.限定成员函数不可以修改任何数据成员。

3.const与指针：

const char \*p 表示 指向的内容不能改变。

char \* const p，就是将P声明为常指针，它的地址不能改变，是固定的，但是它的内容可以改变。

**3.指针和引用的区别，引用可以用常指针实现吗。**

本质上的区别是，指针是一个新的变量，只是这个变量存储的是另一个变量的地址，我们通过访问这个地址来修改变量。

而引用只是一个别名，还是变量本身。对引用进行的任何操作就是对变量本身进行操作，因此以达到修改变量的目的。

[复制代码](javascript:void(0);)

(1)指针：指针是一个变量，只不过这个变量存储的是一个地址，指向内存的一个存储单元；而引用跟原来的变量实质上是同一个东西，只不过是原变量的一个别名而已。如：

int a=1;int \*p=&a;

int a=1;int &b=a;

上面定义了一个整形变量和一个指针变量p，该指针变量指向a的存储单元，即p的值是a存储单元的地址。

而下面2句定义了一个整形变量a和这个整形a的引用b，事实上a和b是同一个东西，在内存占有同一个存储单元。

(2)可以有const指针，但是没有const引用；

(3)指针可以有多级，但是引用只能是一级（int \*\*p；合法 而 int &&a是不合法的）

(4)指针的值可以为**空**，但是引用的值不能为**NULL，并且引用在定义的时候必须初始化**；

(5)指针的值在初始化后可以改变，即指向其它的存储单元，而引用在进行初始化后就不会再改变了。

(6)"sizeof引用"得到的是所指向的变量(对象)的大小，而"sizeof指针"得到的是指针本身的大小；

(7)指针和引用的自增(++)运算意义不一样；

[复制代码](javascript:void(0);)

指针传参的时候，还是值传递，试图修改传进来的指针的值是不可以的。只能修改地址所保存变量的值。

引用传参的时候，传进来的就是变量本身，因此可以被修改。

**4.什么是多态，多态有什么用途。**

1. 定义：“一个接口，多种方法”，程序在运行时才决定调用的函数。
2. 实现：C++多态性主要是通过虚函数实现的，虚函数允许子类重写override(注意和overload的区别，overload是重载，是允许同名函数的表现，这些函数参数列表/类型不同）。

多态与非多态的实质区别就是函数地址是**早绑定还是晚绑定**。如果函数的调用，在编译器编译期间就可以确定函数的调用地址，并生产代码，是静态的，就是说地址是早绑定的。而如果函数调用的地址不能在编译器期间确定，需要在运行时才确定，这就属于晚绑定。

3.目的：**接口重用。**封装可以使得代码模块化，继承可以扩展已存在的代码，他们的目的都是为了代码重用。而多态的目的则是为了接口重用。

4.用法：声明基类的指针，利用该指针指向任意一个子类对象，调用相应的虚函数，可以根据指向的子类的不同而实现不同的方法。

补充一下关于重载、重写、隐藏（总是不记得）的区别：

[复制代码](javascript:void(0);)

Overload(重载)：在C++程序中，可以将语义、功能相似的几个函数用同一个名字表示，但参数或返回值不同（包括类型、顺序不同），即函数重载。

（1）相同的范围（在同一个类中）；

（2）函数名字相同；

（3）参数不同；

（4）virtual 关键字可有可无。

Override(覆盖)：是指派生类函数覆盖基类函数，特征是：

（1）不同的范围（分别位于派生类与基类）；

（2）函数名字相同；

（3）参数相同；

（4）基类函数必须有virtual 关键字。  
**注：重写基类虚函数的时候，会自动转换这个函数为virtual函数，不管有没有加virtual，因此重写的时候不加virtual也是可以的，不过为了易读性，还是加上比较好。**

Overwrite(重写)：隐藏，是指派生类的函数屏蔽了与其同名的基类函数，规则如下：

（1）如果派生类的函数与基类的函数同名，但是参数不同。此时，不论有无virtual关键字，基类的函数将被隐藏（注意别与重载混淆）。

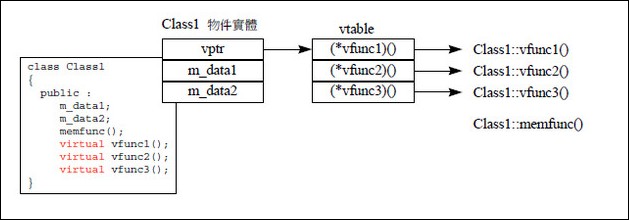
（2）如果派生类的函数与基类的函数同名，并且参数也相同，但是基类函数没有virtual关键字。此时，基类的函数被隐藏（注意别与覆盖混淆）。

[复制代码](javascript:void(0);)

补充一下虚函数表：

多态是由虚函数实现的，而虚函数主要是通过**虚函数表（V-Table）**来实现的。

如果一个类中包含虚函数（virtual修饰的函数），那么这个类就会包含一张虚函数表，虚函数表存储的每一项是一个虚函数的地址。如下图：

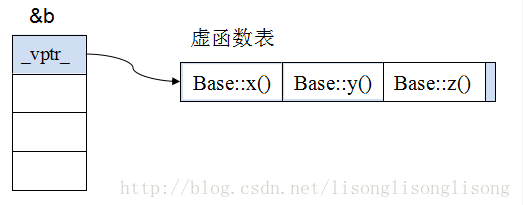


这个类的每一个对象都会包含一个**虚指针**（虚指针存在于对象实例地址的最前面，保证虚函数表有最高的性能），这个虚指针指向虚函数表。

**注：对象不包含虚函数表，只有虚指针，类才包含虚函数表，派生类会生成一个兼容基类的虚函数表。**

* 原始基类的虚函数表

　　下图是原始基类的对象，可以看到虚指针在地址的最前面，指向基类的虚函数表（假设基类定义了3个虚函数）



* 单继承时的虚函数（**无重写基类虚函数**）

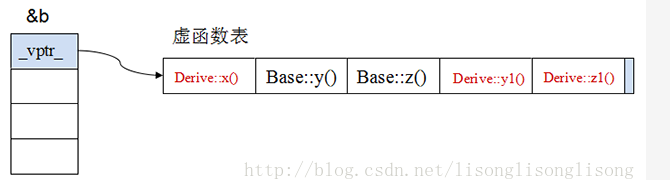
假设现在派生类继承基类，并且重新定义了3个虚函数，派生类会自己产生一个兼容基类虚函数表的**属于自己的虚函数表**。



　　Derive class 继承了 Base class 中的三个虚函数，准确的说，是该函数实体的地址被拷贝到 Derive类的虚函数表，派生类新增的虚函数置于虚函数表的后面，并**按声明顺序存放**。

* 单继承时的虚函数（**重写基类虚函数**）

现在派生类重写基类的x函数，可以看到这个派生类构建自己的虚函数表的时候，修改了base::x()这一项，指向了自己的虚函数。



* 多重继承时的虚函数（Derived ::public Base1,public Base2）

这个派生类多重继承了两个基类base1，base2，因此它有两个虚函数表。



　　它的对象会有**多个虚指针（据说和编译器相关）**，指向不同的虚函数表。

　　多重继承时指针的调整：

Derive b;

Base1\* ptr1 = &b; // 指向 b 的初始地址

Base2\* ptr2 = &b; // 指向 b 的第二个子对象

因为 Base1 是第一个基类，所以 ptr1 指向的是 Derive 对象的起始地址，不需要调整指针（偏移）。

因为 Base2 是第二个基类，所以必须对指针进行调整，即加上一个 offset，让 ptr2 指向 Base2 子对象。

当然，上述过程是由编译器完成的。

Base1\* b1 = (Base1\*)ptr2;

b1->y(); // 输出 Base2::y()

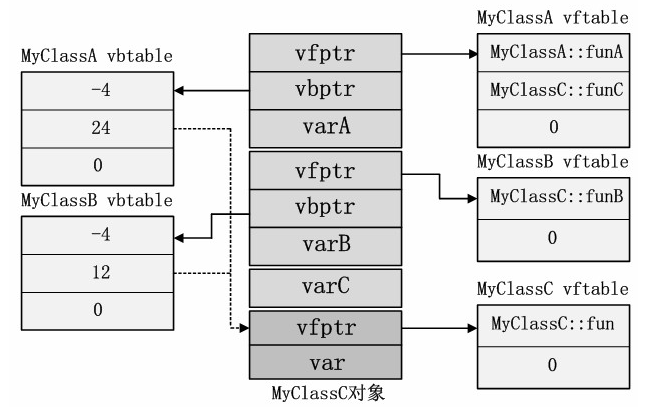
Base2\* b2 = (Base2\*)ptr1;

b2->y(); // 输出 Base1::y()

其实，**通过某个类型的指针访问某个成员时，编译器只是根据类型的定义查找这个成员所在偏移量，用这个偏移量获取成员**。由于 ptr2 本来就指向 Base2 子对象的起始地址，所以b1->y()调用到的是Base2::y()，而 ptr1 本来就指向 Base1 子对象的起始地址（即 Derive对象的起始地址），所以b2->y()调用到的是Base1::y()。

* 虚继承时的虚函数表

　　虚继承的引入把对象的模型变得十分复杂，除了每个基类（MyClassA和MyClassB）和公共基类（MyClass）的虚函数表指针需要记录外，每个虚拟继承了MyClass的父类还需要记录一个虚基类表vbtable的指针vbptr。MyClassC的对象模型如图4所示。



　　　虚基类表每项记录了被继承的虚基类子对象相对于**虚基类表指针的偏移量**。比如MyClassA的虚基类表第二项记录值为24，正是MyClass::vfptr相对于MyClassA::vbptr的偏移量，同理MyClassB的虚基类表第二项记录值12也正是MyClass::vfptr相对于MyClassA::vbptr的偏移量。（[虚函数与虚继承深入探讨](http://www.cnblogs.com/fanzhidongyzby/archive/2013/01/14/2859064.html)）

对象模型探讨：

1.没有继承情况,vptr存放在对象的开始位置,以下是Base1的内存布局

|  |
| --- |
| m\_iData :100 |

 2.单继承的情况下,对象只有一个vptr,它存放在对象的开始位置,派生类子对象在父类子对象的最后面,以下是D1的内存布局

|  |
| --- |
| B1:: m\_iData : 100 |
| B1::vptr : 4294800 |
| B2::vptr : 4294776 |
| D::m\_iData :300 |

4. 虚拟继承情况下,虚父类子对象会放在派生类子对象之后,派生类子对象的第一个位置存放着一个vptr,虚拟子类子对象也会保存一个vptr,以下是VD1的内存布局

|  |
| --- |
| Unknown : 4294888 |
| B1::vptr :4294864 |
| VD1::vptr :        4294944 |
| VD1::m\_iData :  200 |
| VD2::Unknown : 4294952 |
| VD::m\_iData : 500 |
| B1::m\_iData :  100 |

5. 棱形继承的情况下,非虚基类子对象在派生类子对象前面,并按照声明顺序排列,虚基类子对象在派生类子对象后面

|  |
| --- |
| VD1::Unknown : 4294968 |
| VD2::vptr :    4   294932 |
| VD2::m\_iData : 300 |
| B1::vptr :       4294920 |
| B1::m\_iData :  100 |

补充一下纯虚函数：

* 定义： 在很多情况下，基类本身生成对象是不合情理的。为了解决这个问题，方便使用类的多态性，引入了纯虚函数的概念，将函数定义为纯虚函数（方法：virtual ReturnType Function()=**0**;）纯虚函数不能再在基类中实现，编译器要求在派生类中必须予以重写以实现多态性。同时含有纯虚拟函数的类称为抽象类，它不能生成对象。
* 特点：

1，当想在基类中抽象出一个方法，且该基类只做能被继承，而不能被实例化；（避免类被实例化且在编译时候被发现，可以采用此方法）

2，这个方法必须在派生类(derived class)中被实现；

* 目的：使派生类仅仅只是继承函数的接口。

补充一下纯虚函数：

* 定义：称带有纯虚函数的类为抽象类。
* 作用：为一个继承体系提供一个公共的根，为派生类提供操作接口的通用语义。
* 特点：1.抽象类只能作为基类来使用，而继承了抽象类的派生类如果没有实现纯虚函数，而只是继承纯虚函数，那么该类仍旧是一个抽象类，如果实现了纯虚函数，就不再是抽象类。

　　　　　　2.抽象类不可以定义对象。

补充一下多重继承和虚继承：

多重继承：

定义：派生类继承多个基类，派生类为每个基类（显式或隐式地）指定了访问级别——public、protected 或 private。

class Panda : public Bear, public Endangered {

}

构造：

* 1. 派生类的对象包含每个基类的基类子对象。
  2. 派生类构造函数初始化所有基类（多重继承中若没有显式调用某个基类的构造函数，则编译器会调用该基类默认构造函数），派生类只能初始化自己的基类，并不需要考虑基类的基类怎么初始化。
  3. 多重继承时，基类构造函数按照基类构造函数在类派生列表中的出现次序调用。

析构：总是按构造函数运行的**逆序**调用析构函数。（基类的析构函数最好写成virtual，否则再子类对象销毁的时候，无法销毁子类对象部分资源。）假定所有根基类都将它们的析构函数适当定义为虚函数，那么，无论通过哪种指针类型删除对象，虚析构函数的处理都是一致的。

拷贝构造/赋值：如果要为派生类编写拷贝构造函数，则需要为调用基类相应拷贝构造函数并为其传递参数，否则只会拷贝派生类部分。

深拷贝与浅拷贝：

浅拷贝：默认的复制构造函数只是完成了对象之间的位拷贝，也就是把对象里的值完全复制给另一个对象，如A=B。这时，如果B中有一个成员变量指针已经申请了内存，那A中的那个成员变量也指向同一块内存。  
　　　　这就出现了问题：当B把内存释放了（如：析构），这时A内的指针就是野指针了，出现运行错误。

深拷贝：自定义复制构造函数需要注意，对象之间发生复制，资源重新分配，即A有5个空间，B也应该有5个空间，而不是指向A的5个空间。

虚继承与虚基类：

定义：在多重继承下，一个基类可以在派生层次中出现多次。（派生类对象中可能出现多个基类对象）在 C++ 中，通过使用**虚继承**解决这类问题。虚继承是一种机制，类通过虚继承指出它希望共享其虚基类的状态。在虚继承下，对给定虚基类，无论该类在派生层次中作为虚基类出现多少次，只继承一个共享的基类子对象。共享的基类子对象称为**虚基类**。

用法：istream 和 ostream 类对它们的基类进行虚继承。通过使基类成为虚基类，istream 和 ostream 指定，如果其他类（如 iostream 同时继承它们两个，则派生类中只出现它们的公共基类ios的一个副本。通过在派生列表中包含关键字 virtual 设置虚基类：

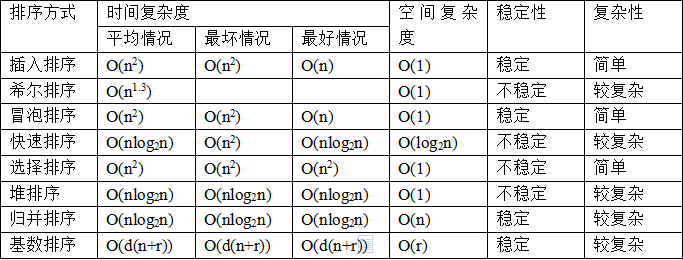
**class istream : public virtual ios { ... };**

**class ostream : virtual public ios { ... };**

**// iostream inherits only one copy of its ios base class**

**class iostream: public istream, public ostream { ... };**

**5.各个排序算法的时间复杂度和稳定性，快排的原理。**



[排序深入探讨](http://blog.csdn.net/morewindows/article/details/7961256)

* 插入排序

　　每次将一个待排序的数据，跟前面已经有序的序列的数字一一比较找到自己合适的位置，插入到序列中，直到全部数据插入完成。

* 希尔排序

　　先将整个待排元素序列分割成若干个子序列（由相隔某个“增量”的元素组成的）分别进行直接插入排序，然后依次缩减增量再进行排序，待整个序列中的元素基本有序（增量足够小）时，再对全体元素进行一次直接插入排序。由于希尔排序是对**相隔若干距离**的数据进行**直接插入排序**，因此可以形象的称希尔排序为“**跳着插**”

* 冒泡排序

通过交换使相邻的两个数变成小数在前大数在后，这样每次遍历后，最大的数就“沉”到最后面了。重复N次即可以使数组有序。

冒泡排序改进1：在某次遍历中如果没有数据交换，说明整个数组已经有序。因此通过设置标志位来记录此次遍历有无数据交换就可以判断是否要继续循环。

冒泡排序改进2：记录某次遍历时最后发生数据交换的位置，这个位置之后的数据显然已经有序了。因此通过记录最后发生数据交换的位置就可以确定下次循环的范围了。

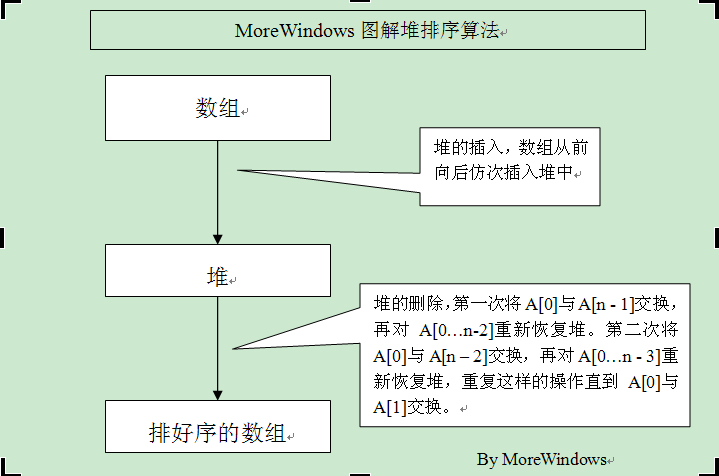
* 快速排序

**“挖坑填数+分治法”**，首先令i =L; j = R; 将a[i]挖出形成第一个坑，称a[i]为基准数。然后j--由后向前找比基准数小的数，找到后挖出此数填入前一个坑a[i]中，再i++由前向后找比基准数大的数，找到后也挖出此数填到前一个坑a[j]中。重复进行这种“挖坑填数”直到i==j。再将基准数填入a[i]中，这样i之前的数都比基准数小，i之后的数都比基准数大。因此将数组分成二部分再分别重复上述步骤就完成了排序。

* 选择排序

数组分成有序区和无序区，初始时整个数组都是无序区，然后每次从无序区选一个最小的元素直接放到有序区的最后，直到整个数组变有序区。

* 堆排序



堆的插入就是——每次插入都是将新数据放在数组最后，而从这个新数据的父结点到根结点必定是一个有序的数列，因此只要将这个新数据插入到这个有序数列中即可。

堆的删除就是——堆的删除就是将最后一个数据的值赋给根结点，然后再从根结点开始进行一次从上向下的调整。调整时先在左右儿子结点中找最小的，如果父结点比这个最小的子结点还小说明不需要调整了，反之将父结点和它交换后再考虑后面的结点。相当于从根结点开始将一个数据在有序数列中进行“下沉”。

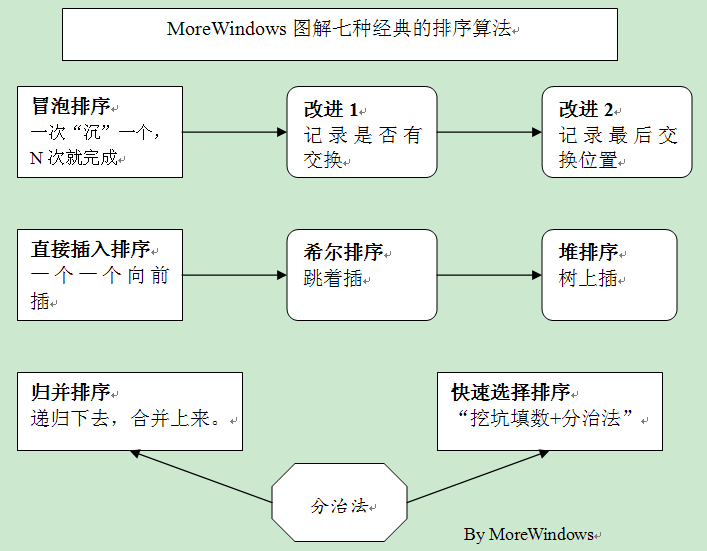
因此，堆的插入和删除非常类似**直接插入排序**，只不是在**二叉树**上进行插入过程。所以可以将堆排序形容为“**树上插**”

* 归并排序

归并排序主要分为两步：分数列（divide），每次把数列一分为二，然后分到只有两个元素的小数列；合数列（Merge），合并两个已经内部有序的子序列，直至所有数字有序。用递归可以实现。

* 基数排序（桶排序）

基数排序，第一步根据数字的个位分配到每个桶里，在桶内部排序，然后将数字再输出（串起来）；然后根据十位分桶，继续排序，再串起来。直至所有位被比较完，所有数字已经有序。

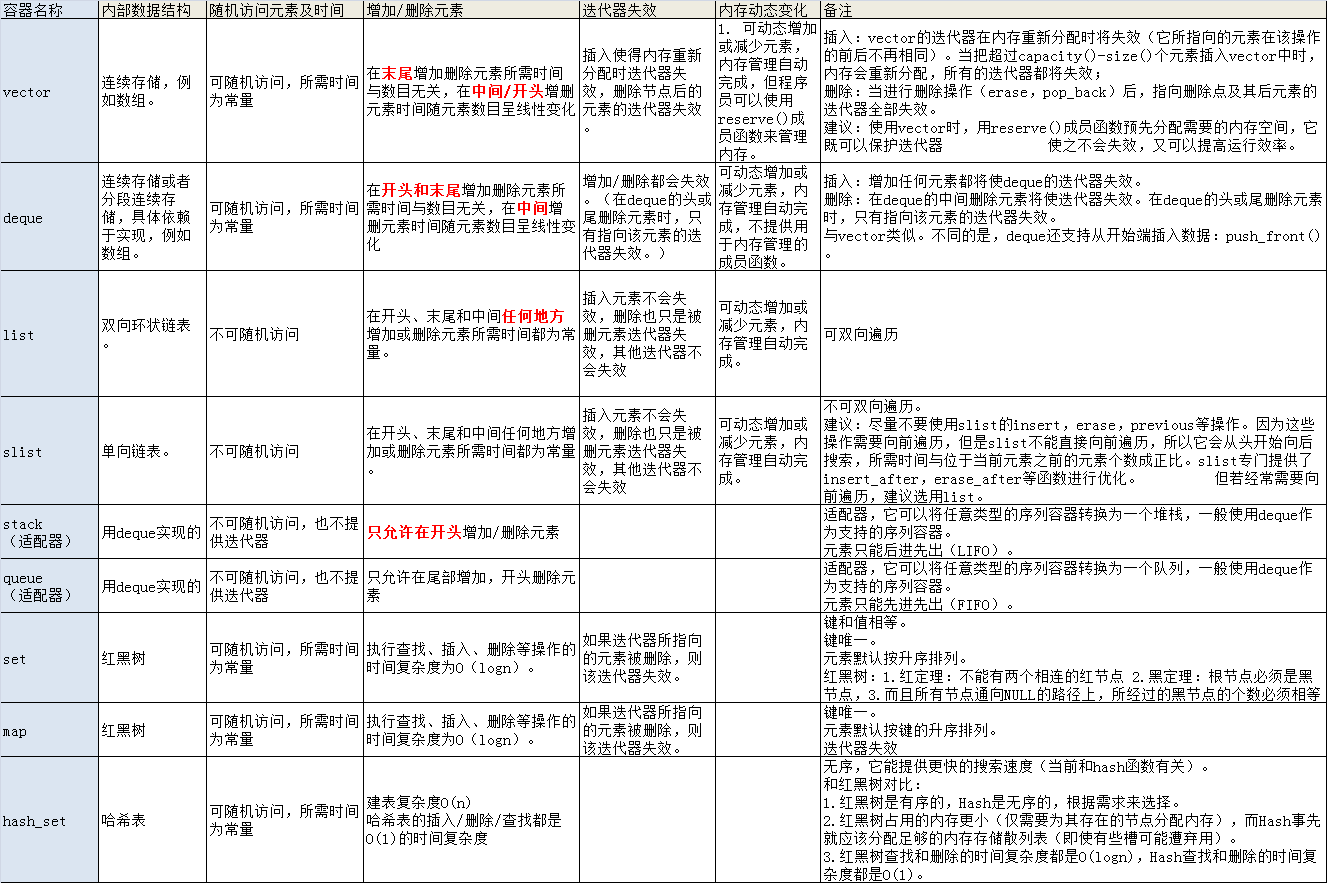


**6.vector中size()和capacity()的区别。**

size()指容器当前拥有的元素个数（对应的resize(size\_type)会在容器尾添加或删除一些元素，来调整容器中实际的内容，使容器达到指定的大小。）；capacity（）指容器在必须分配存储空间之前可以存储的元素总数。

size表示的这个vector里容纳了多少个元素，capacity表示vector能够容纳多少元素，它们的不同是在于vector的size是2倍增长的。如果vector的大小不够了，比如现在的capacity是4，插入到第五个元素的时候，发现不够了，此时会给他重新分配8个空间，把原来的数据及新的数据复制到这个新分配的空间里。（会有迭代器失效的问题）

各容器的特点：



**7.map和set的原理。**

([map和set的四个问题](http://www.cnblogs.com/LUO77/p/5774476.html))

map和set的底层实现主要是由红黑树实现的。

红黑树：

性质1 节点是**红色**或**黑色**。

性质2 根节点是**黑色**。

性质3 每个叶节点（NIL节点，空节点）是**黑色**的。

性质4 每个**红色**节点的两个子节点都是黑色。(从每个叶子到根的所有路径上不能有两个连续的红色节点)

性质5 从任一节点到其每个叶子的所有路径都包含相同数目的**黑色**节点。

这些约束的好处是：保持了树的相对平衡，同时又比AVL的插入删除操作的复杂性要低许多。

（[深入探讨红黑树](http://daoluan.net/%E6%95%B0%E6%8D%AE%E7%BB%93%E6%9E%84/%E7%AE%97%E6%B3%95/2013/09/25/rbtree-is-not-difficult.html)）

**8.tcp为什么要三次握手，tcp为什么可靠。**

为什么不能两次握手：（防止已失效的连接请求又传送到服务器端，因而产生错误）

假设改为两次握手，client端发送的一个连接请求在服务器滞留了，这个连接请求是无效的，client已经是closed的状态了，而服务器认为client想要建立

一个新的连接，于是向client发送确认报文段，而client端是closed状态，无论收到什么报文都会丢弃。而如果是两次握手的话，此时就已经建立连接了。

服务器此时会一直等到client端发来数据，这样就浪费掉很多server端的资源。

（校注：此时因为client没有发起建立连接请求，所以client处于CLOSED状态，接受到任何包都会丢弃，谢希仁举的例子就是这种场景。但是如果服务器发送对这个延误的旧连接报文的确认的同时，客户端调用connect函数发起了连接，就会使客户端进入SYN\_SEND状态，当服务器那个对延误旧连接报文的确认传到客户端时，因为客户端已经处于SYN\_SEND状态，所以就会使客户端进入ESTABLISHED状态，此时服务器端反而丢弃了这个重复的通过connect函数发送的SYN包，见第三个图。而连接建立之后，发送包由于SEQ是以被丢弃的SYN包的序号为准，而服务器接收序号是以那个延误旧连接SYN报文序号为准，导致服务器丢弃后续发送的数据包）

**三次握手的最主要目的是保证连接是双工的，可靠更多的是通过重传机制来保证的。**

TCP可靠传输的实现：

TCP 连接的每一端都必须设有两个窗口——一个发送窗口和一个接收窗口。TCP 的可靠传输机制用字节的序号进行控制。TCP 所有的确认都是基于序号而不是基于报文段。  
发送过的数据未收到确认之前必须保留，以便超时重传时使用。发送窗口没收到确认不动，和收到新的确认后前移。

**发送缓存**用来暂时存放： 发送应用程序传送给发送方 TCP 准备发送的数据；TCP 已发送出但尚未收到确认的数据。

**接收缓存**用来暂时存放：按序到达的、但尚未被接收应用程序读取的数据； 不按序到达的数据。

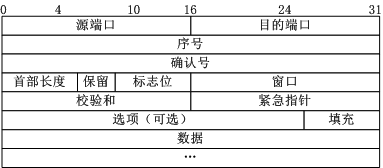
必须强调三点：

1> A 的发送窗口并不总是和 B 的接收窗口一样大（因为有一定的时间滞后）。

2> TCP 标准没有规定对不按序到达的数据应如何处理。通常是先临时存放在接收窗口中，等到字节流中所缺少的字节收到后，再按序交付上层的应用进程。

3> TCP 要求接收方必须有累积确认的功能，这样可以减小传输开销

* TCP报文格式

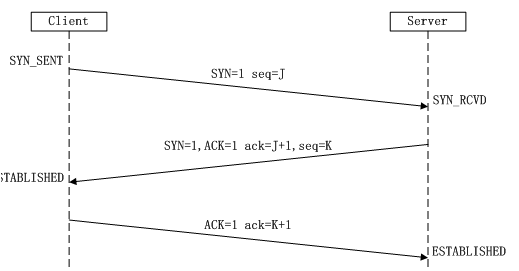


  （1）序号：Seq序号，占32位，用来标识从TCP源端向目的端发送的字节流，发起方发送数据时对此进行标记。  
  （2）确认序号：Ack序号，占32位，只有ACK标志位为1时，确认序号字段才有效，Ack=Seq+1。  
  （3）标志位：共6个，即URG、ACK、PSH、RST、SYN、FIN等，具体含义如下：  
  　　（A）URG：紧急指针（urgent pointer）有效。  
  　　（B）ACK：确认序号有效。  
  　　（C）PSH：接收方应该尽快将这个报文交给应用层。  
  　　（D）RST：重置连接。  
  　　（E）SYN：发起一个新连接。  
  　　（F）FIN：释放一个连接。

 需要注意的是：  
  （A）不要将确认序号Ack与标志位中的ACK搞混了。  
  （B）确认方Ack=发起方Req+1，两端配对。

* 三次握手

TCP三次即建立TCP连接，指建立一个TCP连接时，需要客户端服务端总共发送3 个包以确认连接的建立。在socket编程中，这一过程中由客户端执行connect来触发，流程如下：



（1）第一次握手：Client将标志位SYN置为1（表示要发起一个连接），随机产生一个值seq=J，并将该数据包发送给Server，Client进入**SYN\_SENT**状态，等待Server确认。  
（2）第二次握手：Server收到数据包后由标志位SYN=1知道Client请求建立连接，Server将标志位SYN和ACK都置为1，ack=J+1，随机产生一个值seq=K，并将该数据包发送给Client以确认连接请求，Server进入**SYN\_RCVD**状态。  
（3）第三次握手：Client收到确认后，**检查ack是否为J+1，ACK是否为1**，如果正确则将标志位ACK置为1，ack=K+1，并将该数据包发送给Server，Server检查ack是否为K+1，ACK是否为1，如果正确则连接建立成功，Client和Server进入ESTABLISHED状态，完成三次握手，随后Client与Server之间可以开始传输数据了。

[复制代码](javascript:void(0);)

SYN攻击：

在三次握手过程中，Server发送SYN-ACK之后，收到Client的ACK之前的TCP连接称为半连接（half-open connect），此时Server处于SYN\_RCVD状态，当收到ACK后，Server转入ESTABLISHED状态。SYN攻击就是Client在短时间内伪造大量不存在的IP地址，并向Server不断地发送SYN包，Server回复确认包，并等待Client的确认，由于源地址是不存在的，因此，Server需要不断重发直至超时，这些伪造的SYN包将产时间占用未连接队列，导致正常的SYN请求因为队列满而被丢弃，从而引起网络堵塞甚至系统瘫痪。SYN攻击时一种典型的DDOS攻击，检测SYN攻击的方式非常简单，即当Server上有大量半连接状态且源IP地址是随机的，则可以断定遭到SYN攻击了，使用如下命令可以让之现行：

#netstat -nap | grep SYN\_RECV

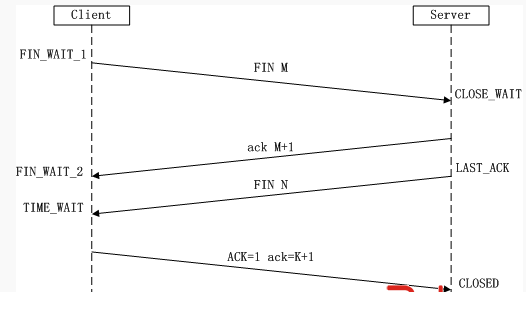
[复制代码](javascript:void(0);)

ddos攻击：

分布式拒绝服务(DDoS:Distributed Denial of Service)攻击指借助于客户/服务器技术，将多个计算机联合起来作为攻击平台，对一个或多个目标发动DDoS攻击，从而成倍地提高拒绝服务攻击的威力。通常，攻击者使用一个偷窃帐号将DDoS主控程序安装在一个计算机上，在一个设定的时间主控程序将与大量代理程序通讯，代理程序已经被安装在网络上的许多计算机上。代理程序收到指令时就发动攻击。利用客户/服务器技术，主控程序能在几秒钟内激活成百上千次代理程序的运行。

* 四次挥手

所谓四次挥手（Four-Way Wavehand）即终止TCP连接，就是指断开一个TCP连接时，需要客户端和服务端总共发送4个包以确认连接的断开。在socket编程中，这一过程由客户端或服务端任一方执行close来触发，整个流程如下图所示：



由于TCP连接时全双工的，因此，每个方向都必须要单独进行关闭，这一原则是当一方完成数据发送任务后，发送一个FIN来终止这一方向的连接，收到一个FIN只是意味着这一方向上没有数据流动了，即不会再收到数据了，但是在这个TCP连接上仍然能够发送数据，直到这一方向也发送了FIN。首先进行关闭的一方将执行主动关闭，而另一方则执行被动关闭，上图描述的即是如此。  
 （1）第一次挥手：Client发送一个FIN，用来关闭Client到Server的数据传送，Client进入FIN\_WAIT\_1状态。  
 （2）第二次挥手：Server收到FIN后，发送一个ACK给Client，确认序号为收到序号+1（与SYN相同，一个FIN占用一个序号），Server进入CLOSE\_WAIT状态。  
 （3）第三次挥手：Server发送一个FIN，用来关闭Server到Client的数据传送，Server进入LAST\_ACK状态。  
 （4）第四次挥手：Client收到FIN后，Client进入TIME\_WAIT状态，接着发送一个ACK给Server，确认序号为收到序号+1，Server进入CLOSED状态，完成四次挥手。

 为什么需要TIME\_WAIT

TIMEWAIT状态也称为**2MSL等待状态**。

 1）为实现TCP这种全双工（full-duplex）连接的可靠释放

这样可让TCP再次发送最后的ACK以防这个**ACK丢失**(另一端超时并重发最后的FIN)。这种2MSL等待的另一个结果是这个TCP连接在2MSL等待期间，定义这个连接的插口(客户的IP地址和端口号，服务器的IP地址和端口号)不能再被使用。这个连接只能在2MSL结束后才能再被使用。

2）为使旧的数据包在网络因过期而消失

每个具体TCP实现必须选择一个报文段最大生存时间MSL(Maximum Segment Lifetime)。它是任何报文段被丢弃前在网络内的最长时间。

为什么建立连接是三次握手，而关闭连接却是四次挥手呢？

 这是因为服务端在LISTEN状态下，收到建立连接请求的SYN报文后，把ACK和SYN放在一个报文里发送给客户端。而关闭连接时，当收到对方的FIN报文时，仅仅表示对方不再发送数据了但是还能接收数据，我们也未必全部数据都发送给对方了，所以我们不可以立即close，也可以发送一些数据给对方后，再发送FIN报文给对方来表示同意现在关闭连接，因此，我们的ACK和FIN一般都会分开发送。

**9.函数调用和系统调用的区别。**

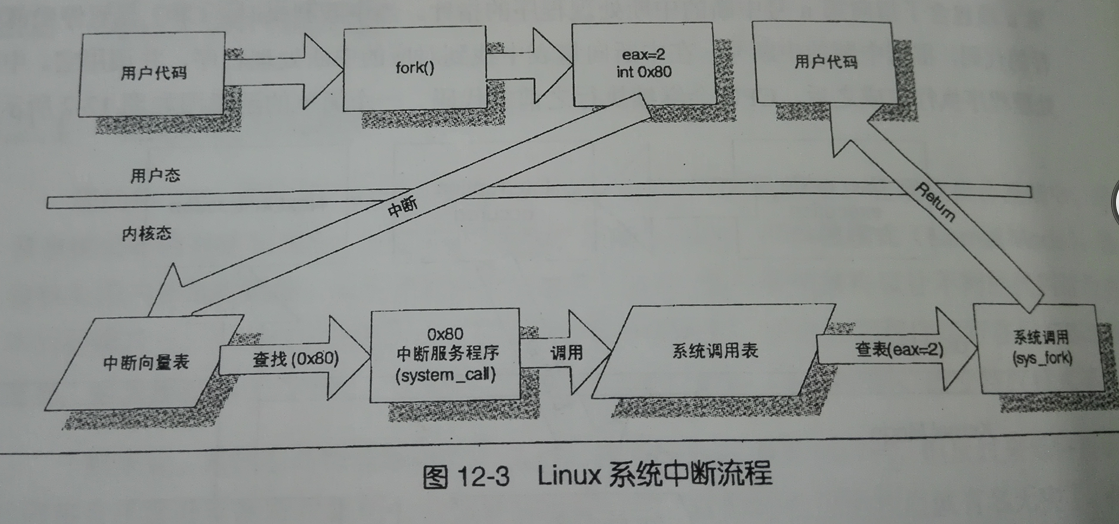
什么是系统调用？（[常见Linux及其分类表](http://www.cnblogs.com/LUO77/p/5823241.html)）

所谓系统调用就是用户在程序中调用操作系统所提供的一个子功能，也就是系统API，系统调用可以被看做特殊的公共子程序。系统中的各种共享资源都由操作系统统一掌管，因此在用户程序中，凡是与资源有关的操作（如存储分配、进行I/O传输及管理文件等），都必须通过系统调用方式向操作系统提出服务请求，并由操作系统代为完成。通常，一个操作系统提供的系统调用命令有几十个乃至上百个之多。这些系统调用按照功能大致可以分为以下几类：

* 设备管理：完成设备的请求或释放，以及设备启动等功能。
* 文件管理：完成文件的读、写、创建及删除等功能
* 进程控制：完成进程的创建、撤销、阻塞、及唤醒的功能
* 进程通信：完成进程之间的消息传递或信号的传递
* 内存管理：完成内存的分配、回收以及获取作业占用内存区大小及始址等功能。

显然，系统调用运行在系统的核心态。通过系统调用的方式来使用系统功能，可以保证系统的稳定性和安全性，防止用户随意更改或访问系统的数据或命令。系统调用命令式由操作系统提供的一个或多个子程序模块来实现的。

下图详细阐述了，Linux系统中系统调用的过程：（int 0x80中断向量是dos系统返回，int 3中断向量是断点指令——可以查中断向量表）



库是可重用的模块，处于用户态。  
系统调用是操作系统提供的服务，处于内核态，不能直接调用，而要使用类似int 0x80的软中断陷入内核，所以库函数中有很大部分是对系统调用的封装。

既然如此，如何调用系统调用？

用户是处于用户态，具有的权限是非常有限，肯定是不能直接使用内核态的服务，只能**间接通过有访问权限的API函数内嵌的系统调用函数来调用。**

介绍下系统调用的过程：  
首先将API函数参数压到**栈**上，然后将函数内调用**系统调用的代码放入寄存器**，**通过陷入中断**，**进入内核将控制权交给操作系统**，**操作系统获得控制后**，**将系统调用代码拿出来**，**跟操作系统一直维护的一张系统调用表做比较**，**已找到该系统调用程序体的内存地址**，**接着访问该地址**，**执行系统调用**。执行完毕后，返回用户程序



例子：

int main()

{

int fd = create("filename",0666);

exit(0);

}

在执行main函数时，是在user mode下执行，当遇到create函数时，继续在user mode下执行，然后将filename和0666两个参数压入栈中寄存器，接着调用库函数create，系统仍然处于user mode。这里的库函数create实际上**调用了内核的系统调用create**，执行到这里后，系统将create系统调用的unique number压入寄存器，然后**执行指令trap使系统进入kernel mode(执行int $0x80产生中断)**。这时系统意识到要进行系统调用的invoke，于是从刚才的寄存器中取出create系统调用的unique number，从系统调用表中得知要invoke的系统调用是create，然后执行。执行完毕返回库函数create的调用，库函数负责检查系统调用的执行情况(检查某些寄存器的值)，然后库函数create根据检查的结果返回响应的值。

这里**trap指令类似于一个系统中断并且是软中断**，而系统调用create类似于一个中断处理函数所有的系统调用都与上边的情况类似，靠**中断机制切换到内核模式实现**。

系统调用通常比库函数要慢，因为要把上下文环境切换到内核模式。

 补充一下系统调用和库函数的区别：

[复制代码](javascript:void(0);)

系统调用：是操作系统为用户态运行的进程和硬件设备(如CPU、磁盘、打印机等)进行交互提供的一组接口，即就是设置在应用程序和硬件设备之间的一个接口层。可以说是操作系统留给用户程序的一个接口。再来说一下，linux内核是单内核，结构紧凑，执行速度快，各个模块之间是直接调用的关系。放眼望整个linux系统，从上到下依次是用户进程->linux内核->硬件。其中系统调用接口是位于Linux内核中的，如果再稍微细分一下的话，整个linux系统从上到下可以是：用户进程->系统调用接口->linux内核子系统->硬件，也就是说Linux内核包括了系统调用接口和内核子系统两部分；或者从下到上可以是：物理硬件->OS内核->OS服务->应用程序，其中操作系统起到“承上启下”的关键作用，向下管理物理硬件，向上为操作系服务和应用程序提供接口，这里的接口就是系统调用了。

一般地，操作系统为了考虑实现的难度和管理的方便，它只提供一少部分的系统调用，这些系统调用一般都是由C和汇编混合编写实现的，其接口用C来定义，而具体的实现则是汇编，这样的好处就是执行效率高，而且，极大的方便了上层调用。

库函数：顾名思义是把函数放到库里。是把一些常用到的函数编完放到一个文件里，供别人用。别人用的时候把它所在的文件名用#include<>加到里面就可以了。一般是放到lib文件里的。一般是指编译器提供的可在c源程序中调用的函数。可分为两类，一类是c语言标准规定的库函数，一类是编译器特定的库函数。(由于版权原因，库函数的源代码一般是不可见的，但在头文件中你可以看到它对外的接口)

libc中就是一个C标准库，里面存放一些基本函数，这些基本函数都是被标准化了的，而且这些函数通常都是用汇编直接实现的。

库函数一般可以概括的分为两类，一类是随着操作系统提供的，另一类是由第三方提供的。随着系统提供的这些库函数把系统调用进行封装或者组合，可以实现更多的功能，这样的库函数能够实现一些对内核来说比较复杂的操作。比如，read()函数根据参数，直接就能读文件，而背后隐藏的比如文件在硬盘的哪个磁道，哪个扇区，加载到内存的哪个位置等等这些操作，程序员是不必关心的，这些操作里面自然也包含了系统调用。而对于第三方的库，它其实和系统库一样，只是它直接利用系统调用的可能性要小一些，而是利用系统提供的API接口来实现功能(API的接口是开放的)。部分Libc库中的函数的功能的实现还是借助了系统掉调用，比如printf的实现最终还是调用了write这样的系统调用；而另一些则不会使用系统调用，比如strlen, strcat, memcpy等。

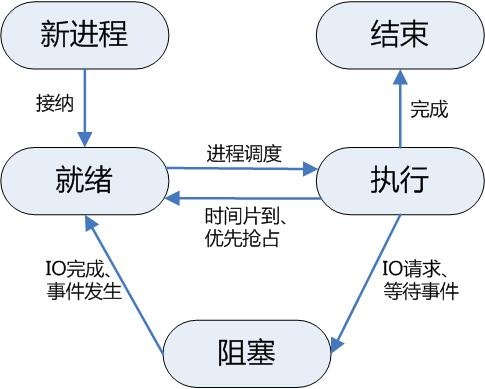
实时上，系统调用所提供给用户的是直接而纯粹的高级服务，如果想要更人性化，具有更符合特定情况的功能，那么就要我们用户自己来定义，因此就衍生了库函数，它把部分系统调用包装起来，一方面把系统调用抽象了，一方面方便了用户级的调用。系统调用和库函数在执行的效果上很相似（当然库函数会更符合需求），但是系统调用是运行于内核状态；而库函数由用户调用，运行于用户态。

系统调用是为了方便使用操作系统的接口，而库函数则是为了人们编程的方便。

[复制代码](javascript:void(0);)

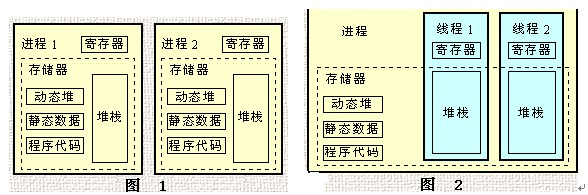
**10.线程和进程，线程可以共享进程里的哪些东西。 知道协程是什么吗**

**进程**，是并发执行的程序在执行过程中分配和管理资源的基本单位，每一个进程都有一个自己的地址空间，即进程空间或（虚空间）。进程空间的大小 只与处理机的位数有关，一个 16 位长处理机的进程空间大小为 216 ，而 32 位处理机的进程空间大小为 232 。进程至少有 5 种基本状态，它们是：初始态，执行态，等待状态，就绪状态，终止状态。



**线程**，在网络或多用户环境下，一个服务器通常需要接收大量且不确定数量用户的并发请求，为每一个请求都创建一个进程显然是行不通的，——无论是从系统资源开销方面或是响应用户请求的效率方面来看。因此，操作系统中线程的概念便被引进了。线程，是进程的一部分，一个没有线程的进程可以被看作是单线程的。线程有时又被称为轻权进程或轻量级进程，也是 CPU 调度的一个基本单位。

共享进程的地址空间，全局变量（数据和堆）。在一个进程中，各个线程共享堆区，而进程中的线程各自维持自己的栈。



Each thread has its own:

* 栈区和栈指针（Stack area and stack pointer）
* 寄存器（Registers）
* 调度优先级Scheduling properties (such as policy or priority)
* 信号（阻塞和悬挂）Signals (pending and blocked signals)
* 普通变量Thread specific data ( automatic variables )

[复制代码](javascript:void(0);)

线程是指进程内的一个执行单元,也是进程内的可调度实体.

与进程的区别:

(1)地址空间:进程内的一个执行单元;进程至少有一个线程;它们共享进程的地址空间;而进程有自己独立的地址空间;

(2)资源拥有:进程是资源分配和拥有的单位,同一个进程内的线程共享进程的资源

(3)线程是处理器调度的基本单位,但进程不是.

4)二者均可并发执行.

进程和线程都是由操作系统所体会的程序运行的基本单元，系统利用该基本单元实现系统对应用的并发性。进程和线程的区别在于：

简而言之,一个程序至少有一个进程,一个进程至少有一个线程.

线程的划分尺度小于进程，使得多线程程序的并发性高。

另外，进程在执行过程中拥有独立的内存单元，而多个线程共享内存，从而极大地提高了程序的运行效率。

线程在执行过程中与进程还是有区别的。每个独立的线程有一个程序运行的入口、顺序执行序列和程序的出口。但是线程不能够独立执行，必须依存在应用程序中，由应用程序提供多个线程执行控制。

从逻辑角度来看，多线程的意义在于一个应用程序中，有多个执行部分可以同时执行。但操作系统并没有将多个线程看做多个独立的应用，来实现进程的调度和管理以及资源分配。这就是进程和线程的重要区别。

进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动,进程是系统进行资源分配和调度的一个独立单位.

线程是进程的一个实体,是CPU调度和分派的基本单位,它是比进程更小的能独立运行的基本单位.线程自己基本上不拥有系统资源,只拥有一点在运行中必不可少的资源(如程序计数器,一组寄存器和栈),但是它可与同属一个进程的其他的线程共享进程所拥有的全部资源.

一个线程可以创建和撤销另一个线程;同一个进程中的多个线程之间可以并发执行.

[复制代码](javascript:void(0);)

协程：

定义：协程其实可以认为是比线程更小的执行单元。为啥说他是一个执行单元，因为他自带CPU上下文。

协程切换：协程拥有自己的寄存器上下文和栈。协程调度切换时，将寄存器上下文和栈保存到其他地方，在切回来的时候，恢复先前保存的寄存器上下文和栈。

　　　　　（我们在自己在进程里面完成逻辑流调度，碰着i\o我就用非阻塞式的。那么我们即可以利用到异步优势，又可以避免反复系统调用，还有进程切换造成的开销，分分钟给你上几千个　　　　逻辑流不费力。这就是协程。）

协程的调度完全由用户控制，一个线程可以有多个协程，用户创建了几个线程，然后每个线程都是循环按照指定的任务清单顺序完成不同的任务，当任务被堵塞的时候执行下一个任务，当恢复的时候再回来执行这个任务，任务之间的切换只需要保存每个任务的上下文内容，就像直接操作栈一样的，这样就完全没有内核切换的开销，可以不加锁的访问全局变量，所以上下文的切换非常快；另外协程还需要保证是非堵塞的且没有相互依赖，协程基本上不能同步通讯，多采用一步的消息通讯，效率比较高。

多线程和多进程的优劣：

多线程还是多进程的争执由来已久，这种争执最常见到在**B/S通讯中服务端并发技术**的选型上，比如WEB服务器技术中，Apache是采用多进程的（perfork模式，每客户连接对应一个进程，每进程中只存在唯一一个执行线程），Java的Web容器Tomcat、Websphere等都是多线程的（每客户连接对应一个线程，所有线程都在一个进程中）。

多进程：fork

多线程：pthread\_create



**11.mysql的数据库引擎有哪些，他们的区别**

### ISAM

　　ISAM是一个定义明确且历经时间考验的数据表格管理方法，它在设计之时就考虑到数据库被查询的次数要远大于更新的次数。因此，ISAM执行读取操作的速度很快，而且不占用大量的内存和存储资源。ISAM的两个主要不足之处在于，它不支持事务处理，也不能够容错：如果你的硬盘崩溃了，那么数据文件就无法恢复了。如果你正在把ISAM用在关键任务应用程序里，那就必须经常备份你所有的实时数据，通过其复制特性，MYSQL能够支持这样的备份应用程序。

### MYISAM

　　MYISAM是MYSQL的ISAM扩展格式和缺省的数据库引擎。除了提供ISAM里所没有的索引和字段管理的大量功能，MYISAM还使用一种表格锁定的机制，来优化多个并发的读写操作。其代价是你需要经常运行OPTIMIZE TABLE命令，来恢复被更新机制所浪费的空间。MYISAM还有一些有用的扩展，例如用来修复数据库文件的MYISAMCHK工具和用来恢复浪费空间的MYISAMPACK工具。

    MYISAM强调了快速读取操作，这可能就是为什么MYSQL受到了WEB开发如此青睐的主要原因：在WEB开发中你所进行的大量数据操作都是读取操作。所以，大多数虚拟主机提供商和INTERNET平台提供商只允许使用MYISAM格式。

### HEAP

　　HEAP允许只驻留在内存里的临时表格。驻留在内存使得HEAP比ISAM和MYISAM的速度都快，但是它所管理的数据是不稳定的，而且如果在关机之前没有进行保存，那么所有的数据都会丢失。在数据行被删除的时候，HEAP也不会浪费大量的空间，HEAP表格在你需要使用SELECT表达式来选择和操控数据的时候非常有用。要记住，用完表格后要删除表格。

### INNODB和BERKLEYDB

　　INNODB和BERKLEYDB（BDB）数据库引擎都是造就MYSQL灵活性的技术的直接产品，这项技术就是MySql++ API。在使用MySql的时候，你所面对的每一个挑战几乎都源于ISAM和MYIASM数据库引擎不支持事务处理也不支持外来键。尽管要比ISAM和MYISAM引擎慢很多，但是INNODB和BDB包括了对事务处理和外来键的支持，这两点都是前两个引擎所没有的。如前所述，如果你的设计需要这些特性中的一者或者两者，那你就要被迫使用后两个引擎中的一个了。

**12.makefile吗，一个文件依赖库a，库a依赖库b，写makefile的时候，a要放在b的前面还是后面**

* Makefile概述：

什么是makefile？或许很多Winodws的程序员都不知道这个东西，因为那些Windows的IDE都为你做了这个工作，但我觉得要作一个好的和professional的程序员，makefile还是要懂。这就好像现在有这么多的HTML的编辑器，但如果你想成为一个专业人士，你还是要了解HTML的标识的含义。特别在Unix下的软件编译，你就不能不自己写makefile了，会不会写makefile，从一个侧面说明了一个人是否具备完成大型工程的能力。

因为，makefile关系到了整个工程的**编译规则**。一个工程中的源文件不计数，其按类型、功能、模块分别放在若干个目录中，**makefile定义了一系列的规则来指定，哪些文件需要先编译，哪些文件需要后编译，哪些文件需要重新编译，甚至于进行更复杂的功能操作，**因为makefile就像一个Shell脚本一样，其中也可以执行操作系统的命令。

makefile带来的好处就是——**“自动化编译”**，一旦写好，只需要一个make命令，整个工程完全自动编译，极大的提高了软件开发的效率。**make是一个命令工具，是一个解释makefile中指令的命令工具，**一般来说，大多数的IDE都有这个命令，比如：Delphi的make，Visual C++的nmake，Linux下GNU的make。可见，makefile都成为了一种在工程方面的编译方法。

现在讲述如何写makefile的文章比较少，这是我想写这篇文章的原因。当然，不同产商的make各不相同，也有不同的语法，但其本质都是在“文件依赖性”上做文章，这里，我仅对GNU的make进行讲述，我的环境是RedHat Linux 8.0，make的版本是3.80。必竟，这个make是应用最为广泛的，也是用得最多的。而且其还是最遵循于IEEE 1003.2-1992 标准的（POSIX.2）。

在这篇文档中，将以C/C++的源码作为我们基础，所以必然涉及一些关于C/C++的编译的知识，相关于这方面的内容，还请各位查看相关的编译器的文档。这里所默认的编译器是UNIX下的GCC和CC。

* 编译和连接：

编译：

定义：一般来说，无论是C、C++、还是pas，首先要把源文件编译成中间代码文件，在Windows下也就是 .obj 文件，UNIX下是 .o 文件，即 Object File，这个动作叫做编译（compile）。

描述：编译时，编译器需要的是语法的正确，函数与变量的声明的正确。只要所有的语法正确，编译器就可以编译出中间目标文件。一般来说，每个源文件都应该对应于一个中间目标文件（O文件或是OBJ文件）。

连接：

定义：然后再把大量的Object File合成执行文件，这个动作叫作链接（link）。

描述：通常是你需要告诉编译器头文件的所在位置（头文件中应该只是声明，而定义应该放在C/C++文件中），链接时，主要是链接函数和全局变量，所以，我们可以使用这些中间目标文件（O文件或是OBJ文件）来链接我们的应用程序。链接器并不管函数所在的源文件，只管函数的中间目标文件（Object File），在大多数时候，由于源文件太多，编译生成的中间目标文件太多，而在链接时需要明显地指出中间目标文件名，这对于编译很不方便，所以，我们要给中间目标文件打个包，在Windows下这种包叫“库文件”（Library File)，也就是 .lib 文件，在UNIX下，是Archive File，也就是 .a 文件。

总结一下，源文件首先会生成中间目标文件，再由中间目标文件生成执行文件。在编译时，编译器只检测程序语法，和函数、变量是否被声明。如果函数未被声明，编译器会给出一个警告，但可以生成Object File。而在链接程序时，链接器会在所有的Object File中找寻函数的实现，如果找不到，那到就会报链接错误码（Linker Error），在VC下，这种错误一般是：Link 2001错误，意思说是说，链接器未能找到函数的实现。你需要指定函数的Object File.

* Makefile

make命令执行时，需要一个 Makefile 文件，以告诉make命令需要怎么样的去编译和链接程序。  
  
首先，我们用一个示例来说明Makefile的书写规则。我们的规则是：  
1）如果这个工程没有编译过，那么我们的所有C文件都要编译并被链接。  
2）如果这个工程的某几个C文件被修改，那么我们只编译被修改的C文件，并链接目标程序。  
3）如果这个工程的头文件被改变了，那么我们需要编译引用了这几个头文件的C文件，并链接目标程序。  
  
只要我们的Makefile写得够好，所有的这一切，我们只用一个make命令就可以完成，make命令会自动智能地根据当前的文件修改的情况来确定哪些文件需要重编译，从而自己编译所需要的文件和链接目标程序。

　　Makefile的规则：

　　target…：dependecies…

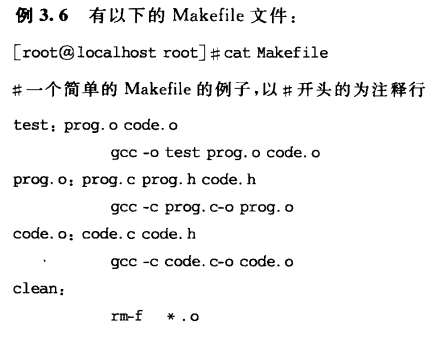
　　　　command

target也就是一个目标文件，可以是Object File，也可以是执行文件。还可以是一个标签（Label），对于标签这种特性，在后续的“伪目标”章节中会有叙述。  
dependicies就是，要生成那个target所需要的文件或是目标。  
command也就是make需要执行的命令。（任意的Shell命令）  
这是一个文件的依赖关系，也就是说，target这一个或多个的目标文件依赖于dependicies中的文件，其生成规则定义在command中。说白一点就是说，dependicies中如果有一个以上的文件比target文件要新的话，command所定义的命令就会被执行。这就是Makefile的规则。也就是Makefile中最核心的内容。（[深入探讨makefile](http://blog.csdn.net/liang13664759/article/details/1771246)）

　　注意事项：

　　1.命令要以[Tab]为开始

　　2.有clean



 1.    面向对象的程序设计思想是什么?

答：把数据结构和对数据结构进行操作的方法封装形成一个个的对象。

2.    什么是类?  
答：把一些具有共性的对象归类后形成一个集合，也就是所谓的类。

3.    对象都具有的两方面特征是什么?分别是什么含义?  
答：对象都具有的特征是：静态特征和动态特征。  
静态特征是指能描述对象的一些属性（成员变量），动态特征是指对象表现出来的行为（成员函数）

4.    在头文件中进行类的声明，在对应的实现文件中进行类的定义有什么意义?  
答：这样可以提高编译效率，因为分开的话只需要编译一次生成对应的.obj文件后，再次应用该类的地方，这个类就不会被再次编译，从而大大的提高了编译效率。

5.    在类的内部定义成员函数的函数体，这种函数会具备那种属性?  
答：这种函数会自动为内联函数，这种函数在函数调用的地方在编译阶段都会进行代码替换。

6.    成员函数通过什么来区分不同对象的成员数据?为什么它能够区分?  
答：通过this指针指向对象的首地址来区分的。

7.    C++编译器自动为类产生的四个缺省函数是什么?  
答：默认构造函数，拷贝构造函数，析构函数，赋值函数。

8.    拷贝构造函数在哪几种情况下会被调用?  
答：  
1.当类的一个对象去初始化该类的另一个对象时；  
2.如果函数的形参是类的对象，调用函数进行形参和实参结合时；  
3.如果函数的返回值是类对象，函数调用完成返回时。

9.    构造函数与普通函数相比在形式上有什么不同?（构造函数的作用，它的声明形式来分析）  
答：构造函数是类的一种特殊成员函数，一般情况下，它是专门用来初始化对象成员变量的。  
构造函数的名字必须与类名相同，它不具有任何类型，不返回任何值。

10.  什么时候必须重写拷贝构造函数?  
答：当构造函数涉及到动态存储分配空间时，要自己写拷贝构造函数，并且要深拷贝。

11.  构造函数的调用顺序是什么?  
答： 1.先调用基类构造函数  
2.按声明顺序初始化数据成员  
3.最后调用自己的构造函数。

12.  哪几种情况必须用到初始化成员列表?  
答： 1.类的成员是常量成员初始化；  
2.类的成员是对象成员初始化，而该对象没有无参构造函数。  
3.类的成员为引用时。

13.  什么是常对象?  
答：常对象是指在任何场合都不能对其成员的值进行修改的对象。

14.  静态函数存在的意义?  
答：静态私有成员在类外不能被访问，可通过类的静态成员函数来访问；  
当类的构造函数是私有的时，不像普通类那样实例化自己，只能通过静态成员函数来调用构造函数。

15.  在类外有什么办法可以访问类的非公有成员?  
答：友元，继承，公有成员函数。

16.  什么叫抽象类?  
答：不用来定义对象而只作为一种基本类型用作继承的类。

17.  运算符重载的意义?  
答：为了对用户自定义数据类型的数据的操作与内定义的数据类型的数据的操作形式一致。

18.  不允许重载的5个运算符是哪些?  
答：1.    .\*（成员指针访问运算符号）  
2.    ：：域运算符  
3.    Sizeof 长度运算符号  
4.    ?：条件运算符号  
5.  .（成员访问符）

19.  运算符重载的三种方式?  
答：普通函数，友元函数，类成员函数。

20.  流运算符为什么不能通过类的成员函数重载?一般怎么解决?  
答：因为通过类的成员函数重载必须是运算符的第一个是自己，而对流运算的重载要求第一个参数是流对象。所以一般通过友元来解决。  
21.  赋值运算符和拷贝构造函数的区别与联系?  
答： 相同点：都是将一个对象copy到另一个中去。  
不同点：拷贝构造函数涉及到要新建立一个对象。

22.  在哪种情况下要调用该类的析构函数?  
答：对象生命周期结束时。

23.  对象间是怎样实现数据的共享的?  
答：通过类的静态成员变量来实现对象间的数据共享。静态成员变量占有自己独立的空间不为某个对象所私有。

24.  友元关系有什么特性?  
答：单向的，非传递的，不能继承的。

25.  对对象成员进行初始化的次序是什么?  
答：它的次序完全不受它们在初始化表中次序的影响，只有成员对象在类中声明的次序来决定的。

26.  类和对象之间的关系是什么?  
答：类是对象的抽象，对象是类的实例。

27.  对类的成员的访问属性有什么?  
答：public，protected，private。

28．const char \*p和char \*  const p; 的区别  
答： 如果const位于星号的左侧，则const就是用来修饰指针所指向的变量，即指针指向为常量；  
如果const位于星号的右侧，const就是修饰指针本身，即指针本身是常量。

29.  是不是一个父类写了一个virtual 函数，如果子类覆盖它的函数不加virtual ,也能实现多态?  
答： virtual修饰符会被隐形继承的。  
virtual可加可不加,子类覆盖它的函数不加virtual ,也能实现多态。

30.  函数重载是什么意思?它与虚函数的概念有什么区别?  
答：函数重载是一个同名函数完成不同的功能，编译系统在编译阶段通过函数参数个数、参数类型不同，函数的返回值来区分该调用哪一个函数，即实现的是静态的多态性。但是记住：不能仅仅通过函数返回值不同来实现函数重载。而虚函数实现的是在基类中通过使用关键字virtual来申明一个函数为虚函数，含义就是该函数的功能可能在将来的派生类中定义或者在基类的基础之上进行扩展，系统只能在运行阶段才能动态决定该调用哪一个函数，所以实现的是动态的多态性。它体现的是一个纵向的概念，也即在基类和派生类间实现。

31.  构造函数和析构函数是否可以被重载,为什么?  
答：构造函数可以被重载，析构函数不可以被重载。因为构造函数可以有多个且可以带参数，而析构函数只能有一个，且不能带参数。

32.  如何定义和实现一个类的成员函数为回调函数?  
答：所谓的回调函数，就是预先在系统的对函数进行注册，让系统知道这个函数的存在，以后，当某个事件发生时，再调用这个函数对事件进行响应。  
定义一个类的成员函数时在该函数前加CALLBACK即将其定义为回调函数，函数的实现和普通成员函数没有区别。

33.  虚函数是怎么实现的?  
答：简单说来使用了虚函数表.

34.  抽象类不会产生实例，所以不需要有构造函数。 错

35.  从一个模板类可以派生新的模板类，也可以派生非模板类。 对

36.  main 函数执行以前，还会执行什么代码?  
答案：全局对象的构造函数会在main 函数之前执行。

37.  当一个类A 中没有生命任何成员变量与成员函数,这时sizeof(A)的值是多少，如果不是零，请解释一下编译器为什么没有让它为零。（Autodesk）  
答案：肯定不是零。举个反例，如果是零的话，声明一个class A[10]对象数组，而每一个对象占用的空间是零，这时就没办法区分A[0],A[1]…了。

38.  delete与 delete []区别：  
答：delete只会调用一次析构函数，而delete[]会调用每一个成员的析构函数。

39．子类析构时要调用父类的析构函数吗?  
答：会调用。析构函数调用的次序是先派生类的析构后基类的析构，也就是说在基类的的析构调用的时候,派生类的信息已经全部销毁了

40.  继承的优缺点。

答：  
1、类继承是在编译时刻静态定义的，且可直接使用，  
2、类继承可以较方便地改变父类的实现。  
缺点：  
1、因为继承在编译时刻就定义了，所以无法在运行时刻改变从父类继承的实现  
2、父类通常至少定义了子类的部分行为，父类的任何改变都可能影响子类的行为  
3、如果继承下来的实现不适合解决新的问题，则父类必须重写或被其他更适合的类替换。这种依赖关系限制了灵活性并最终限制了复用性。  
41.  解释堆和栈的区别。   
答：栈区（stack）— 由编译器自动分配释放 ，存放函数的参数值，局部变量的值等。  
堆（heap）一般由程序员分配释放， 若程序员不释放，程序结束时可能由OS回收 。

42.  一个类的构造函数和析构函数什么时候被调用,是否需要手工调用?  
答：构造函数在创建类对象的时候被自动调用，析构函数在类对象生命期结束时，由系统自动调用。

43.  何时需要预编译：  
答：总是使用不经常改动的大型代码体。  
程序由多个模块组成，所有模块都使用一组标准的包含文件和相同的编译选项。在这种情况下，可以将所有包含文件预编译为一个预编译头。

44.  多态的作用?  
答：主要是两个：  
1. 隐藏实现细节，使得代码能够模块化；扩展代码模块，实现代码重用；  
2. 接口重用：为了类在继承和派生的时候，保证使用家族中任一类的实例的某一属性时的正确调用

45.  虚拟函数与普通成员函数的区别?内联函数和构造函数能否为虚拟函数?  
答案：区别：虚拟函数有virtual关键字，有虚拟指针和虚函数表，虚拟指针就是虚拟函数的接口，而普通成员函数没有。内联函数和构造函数不能为虚拟函数。

46.  构造函数和析构函数的调用顺序? 析构函数为什么要虚拟?  
答案：构造函数的调用顺序：基类构造函数—对象成员构造函数—派生类构造函数；析构函数的调用顺序与构造函数相反。析构函数虚拟是为了防止析构不彻底，造成内存的泄漏。  
47. C++中类型为private的成员变量可以由哪些函数访问?  
答：只可以由本类中的成员函数和友元函数访问  
48.  请说出类中private，protect，public三种访问限制类型的区别  
答：private是私有类型，只有本类中的成员函数访问;protect是保护型的，本类和继承类可以访问;public是公有类型，任何类都可以访问.  
49.  类中成员变量怎么进行初始化?  
答：可以通过构造函数的初始化列表或构造函数的函数体实现。  
50.  在什么时候需要使用“常引用”?   
答：如果既要利用引用提高程序的效率，又要保护传递给函数的数据不在函数中被改变，就应使用常引用。  
51.  引用与指针有什么区别?  
答 、1) 引用必须被初始化，指针不必。  
2) 引用初始化以后不能被改变，指针可以改变所指的对象。  
3) 不存在指向空值的引用，但是存在指向空值的指针。  
52.  描述实时系统的基本特性  
答 、在特定时间内完成特定的任务，实时性与可靠性。  
54.  全局变量和局部变量在内存中是否有区别?如果有，是什么区别?  
答 、全局变量储存在静态数据区，局部变量在堆栈中。  
55.  堆栈溢出一般是由什么原因导致的?  
答 、没有回收垃圾资源  
56.  什么函数不能声明为虚函数?  
答：构造函数（constructor）  
57.  IP地址的编码分为哪俩部分?  
答 IP地址由两部分组成，网络号和主机号。  
58.  不能做switch()的参数类型是：  
答 、switch的参数不能为实型。  
59.  如何引用一个已经定义过的全局变量?  
答 、可以用引用头文件的方式，也可以用extern关键字，如果用引用头文件方式来引用某个在头文件中声明的全局变理，假定你将那个变写错了，那么在编译期间会报错，如果你用extern方式引用时，假定你犯了同样的错误，那么在编译期间不会报错，而在连接期间报错  
60.  对于一个频繁使用的短小函数,在C语言中应用什么实现,在C++中应用什么实现?  
答 、c用宏定义，c++用inline  
61.  C++是不是类型安全的?  
答案：不是。两个不同类型的指针之间可以强制转换（用reinterpret cast)  
62.  当一个类A 中没有生命任何成员变量与成员函数,这时sizeof(A)的值是多少，请解释一下编译器为什么没有让它为零。  
答案：为1。举个反例，如果是零的话，声明一个class A[10]对象数组，而每一个对象占用的空间是零，这时就没办法区分A[0],A[1]…了。  
63.  简述数组与指针的区别?  
答：数组要么在静态存储区被创建（如全局数组），要么在栈上被创建。指针可以随时指向任意类型的内存块。  
(1)修改内容上的区别  
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  
(2) 用运算符sizeof 可以计算出数组的容量（字节数）。sizeof(p),p 为指针得到的是一个指针变量的字节数，而不是p 所指的内存容量。  
64.  C++函数中值的传递方式  
答：有三种方式：值传递、指针传递、引用传递  
65.  内存的分配方式  
答：分配方式有三种，  
1、 静态存储区，是在程序编译时就已经分配好的，在整个运行期间都存在，如全局变量、常量。  
2、 栈上分配，函数内的局部变量就是从这分配的，但分配的内存容易有限。  
3、 堆上分配，也称动态分配，如我们用new,malloc分配内存，用delete,free来释放的内存。  
66.  extern“C”有什么作用?  
答：Extern “C”是由Ｃ＋＋提供的一个连接交换指定符号，用于告诉Ｃ＋＋这段代码是Ｃ函数。这是因为C++编译后库中函数名会变得很长，与C生成的不一致，造成Ｃ＋＋不能直接调用C函数，加上extren “c”后，C++就能直接调用C函数了。  
Extern “C”主要使用正规DLL函数的引用和导出 和 在C++包含C函数或C头文件时使用。使用时在前面加上extern “c” 关键字即可。可以用一句话概括extern “C”这个声明的真实目的：实现C++与C及其它语言的混合编程。  
67.  用什么函数开启新进程、线程。  
答案：  
线程：CreateThread/AfxBeginThread等  
进程：CreateProcess等  
68.  SendMessage和PostMessage有什么区别  
答案：SendMessage是阻塞的，等消息被处理后，代码才能走到SendMessage的下一行。PostMessage是非阻塞的，不管消息是否已被处理，代码马上走到PostMessage的下一行。  
69.  CMemoryState主要功能是什么  
答案：查看内存使用情况，解决内存泄露问题。  
70.  #include <filename.h>和 #include “filename.h” 有什么区别?  
答：对于#include <filename.h> ，编译器从标准库路径开始搜索 filename.h  
对于#include “filename.h” ，编译器从用户的工作路径开始搜索 filename.h  
71.  处理器标识#error的目的是什么?  
答：编译时输出一条错误信息，并中止继续编译。  
72.  #if!defined(AFX\_…\_HADE\_H)  
 #define(AFX\_…\_HADE\_H)  
…  
#endif作用?  
答：防止该头文件被重复引用。  
73. 在定义一个宏的时候要注意什么?  
答：定义部分的每个形参和整个表达式都必须用括号括起来，以避免不可预料的错误发生  
74.  数组在做函数实参的时候会转变为什么类型?  
答：数组在做实参时会变成指针类型。  
75.  系统会自动打开和关闭的3个标准的文件是?  
(1)   标准输入—-键盘—stdin  
(2)   标准输出—-显示器—stdout  
(3)   标准出错输出—-显示器—stderr  
76.  .在Win32下 char, int, float, double各占多少位?  
(1)   Char       占用8位  
(2)   Int      占用32位  
(3)   Float    占用32位  
(4)   Double   占用64位  
77.  strcpy()和memcpy()的区别?  
答：strcpy()和memcpy()都可以用来拷贝字符串，strcpy()拷贝以’\0’结束，但memcpy()必须指定拷贝的长度。  
78.  说明define和const在语法和含义上有什么不同?  
答：(1)   #define是C语法中定义符号变量的方法，符号常量只是用来表达一个值，在编译阶段符号就被值替换了，它没有类型；  
(2)   Const是C++语法中定义常变量的方法，常变量具有变量特性，它具有类型，内存中存在以它命名的存储单元，可以用sizeof测出长度。  
79.  说出字符常量和字符串常量的区别，并使用运算符sizeof计算有什么不用?  
答：字符常量是指单个字符，字符串常量以‘\0’结束，使用运算符sizeof计算多占一字节的存储空间。  
80.  简述全局变量的优缺点?  
答：全局变量也称为外部变量，它是在函数外部定义的变量，它属于一个源程序文件，它保存上一次被修改后的值，便于数据共享，但不方便管理，易引起意想不到的错误。  
81.  总结static的应用和作用?  
答：（1）函数体内static变量的作用范围为该函数体，不同于auto变量，该变量的内存只被分配一次，因此其值在下次调用时仍维持上次的值；  
（2）在模块内的static全局变量可以被模块内所用函数访问，但不能被模块外其它函数访问；  
（3）在模块内的static函数只可被这一模块内的其它函数调用，这个函数的使用范围被限制在声明它的模块内；  
（4）在类中的static成员变量属于整个类所拥有，对类的所有对象只有一份拷贝；  
（5）在类中的static成员函数属于整个类所拥有，这个函数不接收this指针，因而只能访问类的static成员变量。  
82.  总结const的应用和作用?  
答： （1）欲阻止一个变量被改变，可以使用const关键字。在定义该const变量时，通常需要对它进行初始化，因为以后就没有机会再去改变它了；  
（2）对指针来说，可以指定指针本身为const，也可以指定指针所指的数据为const，或二者同时指定为const；  
（3）在一个函数声明中，const可以修饰形参，表明它是一个输入参数，在函数内部不能改变其值；  
（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”  
83.  什么是指针?谈谈你对指针的理解?  
答：指针是一个变量，该变量专门存放内存地址；  
指针变量的类型取决于其指向的数据类型，在所指数据类型前加\*  
指针变量的特点是它可以访问所指向的内存。  
84.  什么是常指针，什么是指向常变量的指针?  
答：常指针的含义是该指针所指向的地址不能变，但该地址所指向的内容可以变化，使用常指针可以保证我们的指针不能指向其它的变量，  
指向常变量的指针是指该指针的变量本身的地址可以变化，可以指向其它的变量，但是它所指的内容不可以被修改。指向长变量的指针定义，  
85.  函数指针和指针函数的区别?  
答：函数指针是指向一个函数入口的指针；指针函数是函数的返回值是一个指针类型。  
87.  简述Debug版本和Release版本的区别?  
答：Debug版本是调试版本，Release版本是发布给用户的最终非调试的版本，  
88.  指针的几种典型应用情况?  
答：  
int \*p[n];—–指针数组，每个元素均为指向整型数据的指针。  
int (\*)p[n];—p为指向一维数组的指针，这个一维数组有n个整型数据。  
int \*p();——函数带回指针，指针指向返回的值。  
int (\*)p();—-p为指向函数的指针。  
89.  static函数与普通函数有什么区别?  
答：static函数在内存中只有一份，普通函数在每个被调用中维持一份拷贝  
90.  struct(结构) 和 union(联合)的区别?  
答：1. 结构和联合都是由多个不同的数据类型成员组成, 但在任何同一时刻, 联合中只存放了一个被选中的成员（所有成员共用一块地址空间）, 而结构的所有成员都存在（不同成员的存放地址不同）。  
2. 对于联合的不同成员赋值, 将会对其它成员重写,  原来成员的值就不存在了, 而对于结构的不同成员赋值是互不影响的。  
91.  class 和 struct 的区别?  
答：struct 的成员默认是公有的，而类的成员默认是私有的。  
92.  简述枚举类型?  
答：枚举方便一次定义一组常量，使用起来很方便；  
93.  assert()的作用?  
答：ASSERT()是一个调试程序时经常使用的宏，在程序运行时它计算括号内的表达式，如果表达式为FALSE (0), 程序将报告错误，并终止执行。如果表达式不为0，则继续执行后面的语句。这个宏通常原来判断程序中是否出现了明显非法的数据，如果出现了终止程序以免导致严重后果，同时也便于查找错误。  
94.  局部变量和全局变量是否可以同名?  
答：能。局部会屏蔽全局。要用全局变量，需要使用”::”(域运算符)。  
95.  程序的局部变量存在于（堆栈）中，全局变量存在于（静态区 ）中，动态申请数据存在于（ 堆）中。  
96.  在什么时候使用常引用?  
答：如果既要利用引用提高程序的效率，又要保护传递给函数的数据不在函数中被改变，就应使用常引用。  
97.  类的声明和实现的分开的好处?  
答：1.    起保护作用；  
2.    提高编译的效率。  
98.  windows消息系统由哪几部分构成?  
答：由一下3部分组成：  
1.    消息队列：操作系统负责为进程维护一个消息队列，程序运行时不断从该消息队列中获取消息、处理消息；  
2.    消息循环：应用程序通过消息循环不断获取消息、处理消息。  
3.    消息处理：消息循环负责将消息派发到相关的窗口上使用关联的窗口过程函数进行处理。  
99.  什么是消息映射?  
答：消息映射就是让程序员指定MFC类（有消息处理能力的类）处理某个消息。然后由程序员完成对该处理函数的编写，以实现消息处理功能。  
100. 什么是UDP和TCP的区别是什么?  
答：TCP的全称为传输控制协议。这种协议可以提供面向连接的、可靠的、点到点的通信。  
UDP全称为用户报文协议，它可以提供非连接的不可靠的点到多点的通信。用TCP还是UDP，那要看你的程序注重哪一个方面?可靠还是快速?

**1. C 和 C++ 区别**

**2. const 有什么用途**

主要有三点：

      1：定义只读变量，即常量

      2：修饰函数的参数和函数的返回值

      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

**3. 指针和引用的区别**

   1：引用是变量的一个别名，内部实现是只读指针

    2：引用只能在初始化时被赋值，其他时候值不能被改变，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，内部传递的实际是变量地址

**4. C++中有了malloc / free , 为什么还需要 new / delete**

1,malloc与free是C++/C语言的标准库函数，new/delete是C++的运算符。它们都可用于申请动态内存和释放内存。

2,对于非内部数据类型的对象而言，光用maloc/free无法满足动态对象的要求。

对象在创建的同时要自动执行构造函数，对象在消亡之前要自动执行析构函数。

由于malloc/free是库函数而不是运算符，不在编译器控制权限之内，不能够把执行构造函数和析构函数的任务强加于malloc/free。

3,因此C++语言需要一个能完成动态内存分配和初始化工作的运算符new，以一个能完成清理与释放内存工作的运算符delete。注意new/delete不是库函数。

**5. 编写类String 的构造函数，析构函数，拷贝构造函数和赋值函数**

**6. 多态的实现**

**7. 单链表的逆置**

**8. 堆和栈的区别**

一个由c/C++编译的程序占用的内存分为以下几个部分

1、栈区（stack）― 由编译器自动分配释放 ，存放函数的参数值，局部变量的值等。其操作方式类似于数据结构中的栈。

2、堆区（heap） ― 一般由程序员分配释放， 若程序员不释放，程序结束时可能由OS回收 。

注意它与数据结构中的堆是两回事，分配方式倒是类似于链表，呵呵。

3、全局区（静态区）（static）―，全局变量和静态变量的存储是放在一块的，

初始化的全局变量和静态变量在一块区域， 未初始化的全局变量和未初始化的静态变量在相邻的另一块区域。 - 程序结束后有系统释放

4、文字常量区 ―常量字符串就是放在这里的。 程序结束后由系统释放

5、程序代码区―存放函数体的二进制代码。

**10. 不调用C/C++ 的字符串库函数，编写strcpy**

char \* strcpy(char \* strDest,const char \* strSrc)

{

if ((strDest==NULL)||strSrc==NULL))

return NULL;

char \* strDestCopy=strDest;

while ((\*strDest++=\*strSrc++)!='\0');

\*strDest = '\0';

return strDestCopy;

}

**11. 关键字static的作用**

   1.  函数体内 static 变量的作用范围为该函数体，不同于 auto 变量， 该变量的内存只被分配一次，因此其值在下次调用时仍维持上次的值

    2.  在模块内的 static 全局变量可以被模块内所有函数访问，但不能被模块外其他函数访问

    3.  在模块内的static 函数只可被这一模块内的其他函数调用，这个函数的使用范围被限制在声明它的模块内

    4.  在类的static 成员变量属于整个类所拥有，对类的所以对象只有一份拷贝

    5.  在类中的 static 成员函数属于整个类所拥有，这个函数不接收 this 指针，因而只能访问类的 static 成员变量

     介绍它最重要的一条：隐藏。（static函数，static变量均可） --> 对应上面的2、3项  
        当同时编译多个文件时，所有未加static前缀的全局变量和函数都具有全局可见性。  
        举例来说明。同时编译两个源文件，一个是a.c，另一个是main.c。

//a.c

char a = 'A'; // global variable

void msg()

{

printf("Hello\n");

}

[复制代码](javascript:void(0);)

//main.c

int main()

{

extern char a; // extern variable must be declared before use

printf("%c ", a);

(void)msg();

return 0;

}

[复制代码](javascript:void(0);)

       程序的运行结果是：

|  |
| --- |
| A Hello |

      为什么在a.c中定义的全局变量a和函数msg能在main.c中使用？

      前面说过，所有未加static前缀的全局变量和函数都具有全局可见性，其它的源文件也能访问。此例中，a是全局变量，msg是函数，并且都没有加static前缀，

        因此对于另外的源文件main.c是可见的。

     如果加了static，就会对其它源文件隐藏。例如在a和msg的定义前加上static，main.c就看不到它们了。

     利用这一特性可以在不同的文件中定义同名函数和同名变量，而不必担心命名冲突。static可以用作函数和变量的前缀，对于函数来讲，static的作用仅限于隐藏

**12. 在c++程序中调用被C编译器编译后的函数，为什么要加extern“C”**

C++语言支持函数重载，[C语言](http://lib.csdn.net/base/c)不支持函数重载，函数被C++编译器编译后在库中的名字与C语言的不同，

      假设某个函数原型为：

1. void foo(int x, inty);

    该函数被C编译器编译后在库中的名字为:  \_foo

    而C++编译器则会产生像: \_foo\_int\_int   之类的名字。  
    为了解决此类名字匹配的问题，C++提供了C链接交换指定符号 extern "C"。

**13. 头文件种的ifndef/define/endif 是干什么用的**

   防止头文件被重复包含

**14. 线程和进程的联系和区别**

**http://blog.csdn**[**.NET**](http://lib.csdn.net/base/dotnet)**/wolenski/article/details/7969908**

**15. 线程有哪几种状态**

**http://blog.csdn**[**.Net**](http://lib.csdn.net/base/dotnet)**/wolenski/article/details/7969908**

**16. 进程间的通信方式**

管道、有名管道、信号、共享内存、消息队列、信号量、套接字、文件.

**17. 线程同步和线程互斥的区别**

**http://blog.csdn.net/wolenski/article/details/7969908**

**18. 线程同步的方式**

[**Linux**](http://lib.csdn.net/base/linux)**:**互斥锁、条件变量和信号量

**http://blog.csdn.net/zsf8701/article/details/7844316**

**19. 网络七层**

**20. TCP和UDP有什么区别**

TCP---传输控制协议,提供的是面向连接、可靠的字节流服务。

                 当客户和服务器彼此交换数据前，必须先在双方之间建立一个TCP连接，之后才能传输数据。

                 TCP提供超时重发，丢弃重复数据，检验数据，流量控制等功能，保证数据能从一端传到另一端。

     UDP---用户数据报协议，是一个简单的面向数据报的运输层协议。

                 UDP不提供可靠性，它只是把应用程序传给IP层的数据报发送出去，但是并不能保证它们能到达目的地。

                 由于UDP在传输数据报前不用在客户和服务器之间建立一个连接，且没有超时重发等机制，故而传输速度很快

**21. 编写socket套接字的步骤**

**22. TCP三次握手和四次挥手, 以及各个状态的作用**

**http://hi.baidu.com/suxinpingtao51/item/be5f71b3a907dbef4ec7fd0e?qq-pf-to=pcqq.c2c**

**23. HTTP协议**

            http（超文本传输协议）是一个基于请求与响应模式的、无状态的、应用层的协议，常基于TCP的连接方式，

     HTTP1.1版本中给出一种持续连接的机制，绝大多数的Web开发，都是构建在HTTP协议之上的Web应用。

    TCP 和 HTTP区别： http://blog.csdn.net/lemonxuexue/article/details/4485877

**24. 使用过的 shell 命令**

**cp , mv , rm , mkdir , touch , pwd , cd  , ls , top , cat , tail , less , df , du , man , find , kill , sudo , cat**

**25. 使用过的 vim 命令**

wq!, dd , dw , yy , p , i , %s/old/new/g , /abc 向后搜索字符串abc ， ？abc向前搜索字符串abc

**26. 使用过的 gdb 命令**

**http://blog.csdn.net/dadalan/article/details/3758025**

**27. 常见**[**算法**](http://lib.csdn.net/base/datastructure)

  快速排序、堆排序和归并排序

**堆排序 ： http://blog.csdn.net/xiaoxiaoxuewen/article/details/7570621**

**快速排序、归并排序： http://blog.csdn.net/morewindows/article/details/6684558**

稳定性分析 **http://baike.baidu.com/link?url=ueoZ3sNIOvMNPrdCKbd8mhfebC85B4nRc-7hPEJWi-hFo5ROyWH2Pxs9RtvLFRJL**

**28. C库函数实现**

**29. 静态链表和动态链表的区别**

 http://blog.csdn.net/toonny1985/article/details/4868786

**31. 大并发( epoll )**

**优点:**

**http://blog.csdn.net/sunyurun/article/details/8194979**

**实例：**

**http://www.cnblogs.com/ggjucheng/archive/2012/01/17/2324974.html**

**32. 海量数据处理的知识点，（hash表， hash统计）**

hash表： http://hi.baidu.com/05104106/item/62736054402852c09e26679b

    海量数据处理方法： http://blog.csdn.net/v\_july\_v/article/details/7382693

**33. 什么时候要用虚析构函数**

       通过基类的指针来删除派生类的对象时，基类的析构函数应该是虚的。否则其删除效果将无法实现。

       一般情况下，这样的删除只能够删除基类对象，而不能删除子类对象，形成了删除一半形象，从而千万内存泄漏。

      原因：

              在公有继承中，基类对派生类及其对象的操作，只能影响到那些从基类继承下来的成员。

              如果想要用基类对非继承成员进行操作，则要把基类的这个操作（函数）定义为虚函数。  
              那么，析构函数自然也应该如此：如果它想析构子类中的重新定义或新的成员及对象，当然也应该声明为虚的。

      注意：

      如果不需要基类对派生类及对象进行操作，则不能定义虚函数（包括虚析构函数），因为这样会增加内存开销。

**34. c++怎样让返回对象的函数不调用拷贝构造函数**

    拷贝构造函数前加 “explicit” 关键字

**35. 孤儿进程和僵尸进程**

   http://www.cnblogs.com/Anker/p/3271773.html

**各大计算机公司 笔试及面试 题目**

**http://blog.csdn.net/huyfaeng/article/category/880022**

面试题有难有易，不能因为容易，我们就轻视，更不能因为难，我们就放弃。我们面对高薪就业的态度永远不变，那就是坚持、坚持、再坚持。出现问题，找原因；遇到困难，想办法。我们一直坚信只有在坚持中才能看到希望，而不是看到希望才去坚持。

人生没有如果，只有结果和后果。既然选择了，就不后悔。年轻就是资本，年轻就要吃苦，就要历练。就要学会在坚持中成长。如此感慨，至深的心得体会，绝对的经验之谈。

OK，进入正题，下面就是《必须掌握的20道技术面试题》。

**问1：请用简单的语言告诉我C++ 是什么？**

答：C++是在C语言的基础上开发的一种面向对象编程语言，应用广泛。C++支持多种编程范式 －－面向对象编程、泛型编程和过程化编程。 其编程领域众广，常用于系统开发，引擎开发等应用领域，是最受广大程序员受用的最强大编程语言之一,支持类：类、封装、重载等特性!

**问2：C和C++的区别？**

答：c++在c的基础上增添类，C是一个结构化语言，它的重点在于算法和数据结构。C程序的设计首要考虑的是如何通过一个过程，对输入（或环境条件）进行运算处理得到输出（或实现过程（事务）控制），而对于C++，首要考虑的是如何构造一个对象模型，让这个模型能够契合与之对应的问题域，这样就可以通过获取对象的状态信息得到输出或实现过程（事务）控制。

**问3：什么是面向对象（OOP）？**

答：面向对象是一种对现实世界理解和抽象的方法、思想，通过将需求要素转化为对象进行问题处理的一种思想。

**问4：什么是多态？**

答：多态是指相同的操作或函数、过程可作用于多种类型的对象上并获得不同的结果。不同的对象，收到同一消息可以产生不同的结果，这种现象称为多态。

**问5：设计模式懂嘛，简单举个例子？**

答：设计模式（Design pattern）是一套被反复使用、多数人知晓的、经过分类编目的、代码设计经验的总结。

比如单例模式，保证一个类仅有一个实例，并提供一个访问它的全局访问点。

适用于：当类只能有一个实例而且客户可以从一个众所周知的访问点访问它时；当这个唯一实例应该是通过子类化可扩展的，并且客户应该无需更改代码就能使用一个扩展的实例时。

比如工厂模式，定义一个用于创建对象的接口，让子类决定实例化哪一个类。Factory Method 使一个类的实例化延迟到其子类。

适用于：当一个类不知道它所必须创建的对象的类的时候；当一个类希望由它的子类来指定它所创建的对象的时候；当类将创建对象的职责委托给多个帮助子类中的某一个，并且你希望将哪一个帮助子类是代理者这一信息局部化的时候。

**问6：STL库用过吗？常见的STL容器有哪些？算法用过哪几个？**

答：STL包括两部分内容：容器和算法。（重要的还有融合这二者的迭代器）

容器，即存放数据的地方。比如array等。

在STL中，容器分为两类：序列式容器和关联式容器。

序列式容器，其中的元素不一定有序，但都可以被排序。如：vector、list、deque、stack、queue、heap、priority\_queue、slist；

关联式容器，内部结构基本上是一颗平衡二叉树。所谓关联，指每个元素都有一个键值和一个实值，元素按照一定的规则存放。如：RB-tree、set、map、multiset、multimap、hashtable、hash\_set、hash\_map、hash\_multiset、hash\_multimap。

下面各选取一个作为说明。

vector：它是一个动态分配存储空间的容器。区别于c++中的array，array分配的空间是静态的，分配之后不能被改变，而vector会自动重分配（扩展）空间。

set：其内部元素会根据元素的键值自动被排序。区别于map，它的键值就是实值，而map可以同时拥有不同的键值和实值。

算法，如排序，复制……以及个容器特定的算法。这点不用过多介绍，主要看下面迭代器的内容。

迭代器是STL的精髓，我们这样描述它：迭代器提供了一种方法，使它能够按照顺序访问某个容器所含的各个元素，但无需暴露该容器的内部结构。它将容器和算法分开，好让这二者独立设计。

**问7：数据结构会吗？项目开发过程中主要用到那些？**

答：数据结构中主要会用到数组，链表，树（较少），也会用到栈和队列的思想。

**问8：const知道吗？解释其作用。**

答：

1.const 修饰类的成员变量，表示成员常量，不能被修改。

2.const修饰函数承诺在本函数内部不会修改类内的数据成员，不会调用其它非 const 成员函数。

3.如果 const 构成函数重载，const 对象只能调用 const 函数，非 const 对象优先调用非 const 函数。

4.const 函数只能调用 const 函数。非 const 函数可以调用 const 函数。

5.类体外定义的 const 成员函数，在定义和声明处都需要 const 修饰符。。

**问9：类的static变量在什么时候初始化？函数的static变量在什么时候初始化？**

答：类的静态成员变量在类实例化之前就已经存在了，并且分配了内存。函数的static变量在执行此函数时进行初始化。

**问10：堆和栈的区别？堆和栈的生命周期？**

答：

**一、堆栈空间分配区别：**

1、栈（操作系统）：由操作系统自动分配释放 ，存放函数的参数值，局部变量的值等。其操作方式类似于数据结构中的栈；

2、堆（操作系统）： 一般由程序员分配释放， 若程序员不释放，程序结束时可能由OS回收，分配方式倒是类似于链表。

**二、堆栈缓存方式区别：**

1、栈使用的是一级缓存， 他们通常都是被调用时处于存储空间中，调用完毕立即释放；

2、堆是存放在二级缓存中，生命周期由虚拟机的垃圾回收算法来决定（并不是一旦成为孤儿对象就能被回收）。所以调用这些对象的速度要相对来得低一些。

**三、堆栈数据结构区别：**

堆（数据结构）：堆可以被看成是一棵树，如：堆排序；

栈（数据结构）：一种先进后出的数据结构。

**问11：C和C++的区别？**

答：C++在C的基础上增添类

C是一个结构化语言，它的重点在于算法和数据结构。

C程序的设计首要考虑的是如何通过一个过程，对输入（或环境条件）进行运算处理得到输出（或实现过程（事务）控制），而对于C++，首要考虑的是如何构造一个对象模型，让这个模型能够契合与之对应的问题域，这样就可以通过获取对象的状态信息得到输出或实现过程（事务）控制。

**问12：解释下封装、继承和多态？**

答：

**一、封装：**

封装是实现面向对象程序设计的第一步，封装就是将数据或函数等集合在一个个的单元中（我们称之为类）。

封装的意义在于保护或者防止代码（数据）被我们无意中破坏。

**二、继承：**

继承主要实现重用代码，节省开发时间。

子类可以继承父类的一些东西。

**三、多态**

多态：同一操作作用于不同的对象，可以有不同的解释，产生不同的执行结果。在运行时，可以通过指向基类的指针，来调用实现派生类中的方法。

**问13：指针和引用的区别？**

答：

1. 指针是一个变量，只不过这个变量存储的是一个地址，指向内存的一个存储单元；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”得到的是指针本身的大小；

7. 指针和引用的自增(++)运算意义不一样；

8. 指针可以有多级，但是引用只能是一级（int \*\*p；合法 而 int &&a是不合法的）

9.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。

**问14：什么是内存泄漏？面对内存泄漏和指针越界，你有哪些方法？你通常采用哪些方法来避免和减少这类错误？**

答：用动态存储分配函数动态开辟的空间，在使用完毕后未释放，结果导致一直占据该内存单元即为内存泄露。

使用的时候要记得指针的长度。

malloc的时候得确定在那里free.

对指针赋值的时候应该注意被赋值指针需要不需要释放.

动态分配内存的指针最好不要再次赋值.

**问15：常用的排序算法有哪些？简单描述几个排序算法的优缺点？**

答：选择、冒泡、快速、\*\*、希尔、归并、堆排等。

1.快排：是冒泡排序的一种改进。

优点：快，数据移动少

缺点：稳定性不足

2.归并：分治法排序，稳定的排序算法，一般用于对总体无序，但局部有序的数列。

优点：效率高O(n)，稳定

缺点：比较占用内存

**问16：new和malloc的区别？**

答：

1、malloc与free是C++/C语言的标准库函数，new/delete是C++的运算符。它们都可用于申请动态内存和释放内存。

2、对于非内部数据类型的对象而言，光用maloc/free无法满足动态对象的要求。对象在创建的同时要自动执行构造函数，对象在消亡之前要自动执行析构函数。

3、由于malloc/free是库函数而不是运算符，不在编译器控制权限之内，不能够把执行构造函数和析构函数的任务强加于malloc/free。因此C++语言需要一个能完成动态内存分配和初始化工作的运算符new，以一个能完成清理与释放内存工作的运算符delete。注意new/delete不是库函数。

4、C++程序经常要调用C函数，而C程序只能用malloc/free管理动态内存。

5、new可以认为是malloc加构造函数的执行。new出来的指针是直接带类型信息的。而malloc返回的都是void指针。

**问17：TCP和UDP通信的差别？什么是IOCP？**

答：

1.TCP面向连接， UDP面向无连接的

2.TCP有保障的，UDP传输无保障的

3.TCP是效率低的，UDP效率高的

4.TCP是基于流的，UDP基于数据报文

5.TCP传输重要数据，UDP传输不重要的数据

IOCP全称I/O Completion Port，中文译为I/O完成端口。

IOCP是一个异步I/O的API，它可以高效地将I/O事件通知给应用程序。

与使用select()或是其它异步方法不同的是，一个套接字[socket]与一个完成端口关联了起来，然后就可继续进行正常的Winsock操作了。然而，当一个事件发生的时候，此完成端口就将被操作系统加入一个队列中。然后应用程序可以对核心层进行查询以得到此完成端口。

**问18：同步IO和异步IO的区别？**

答：

**A. 同步**

所谓同步，就是在发出一个功能调用时，在没有得到结果之前，该调用就不返回。

按照这个定义，其实绝大多数函数都是同步调用（例如sin isdigit等）。

但是一般而言，我们在说同步、异步的时候，特指那些需要其他部件协作或者需要一定时间完成的任务。

最常见的例子就是 SendMessage。

该函数发送一个消息给某个窗口，在对方处理完消息之前，这个函数不返回。

当对方处理完毕以后，该函数才把消息处理函数所返回的值返回给调用者。

**B. 异步**

异步的概念和同步相对。

当一个异步过程调用发出后，调用者不会立刻得到结果。

实际处理这个调用的部件是在调用发出后，通过状态、通知来通知调用者，或通过回调函数处理这个调用。

**问19：解释C++中静态函数和静态变量？**

答：

(1)类静态数据成员在编译时创建并初始化：在该类的任何对象建立之前就存在，不属于任何对象，而非静态类成员变量则是属于对象所有的。类静态数据成员只有一个拷贝，为所有此类的对象所共享。

(2)类静态成员函数属于整个类，不属于某个对象，由该类所有对象共享。

1、static 成员变量实现了同类对象间信息共享。

2、static 成员类外存储，求类大小，并不包含在内。

3、static 成员是命名空间属于类的全局变量，存储在 data 区的rw段。

4、static 成员只能类外初始化。

5、可以通过类名访问（无对象生成时亦可），也可以通过对象访问。

1、静态成员函数的意义，不在于信息共享，数据沟通，而在于管理静态数据成员，完成对静态数据成员的封装。

2、静态成员函数只能访问静态数据成员。原因：非静态成员函数，在调用时 this指针时被当作参数传进。而静态成员函数属于类，而不属于对象，没有 this 指针。

**问20：说下你对内存的了解？**

答：

1.栈 - 由编译器自动分配释放

2.堆 - 一般由程序员分配释放，若程序员不释放，程序结束时可能由OS回收

3.全局区(静态区)，全局变量和静态变量的存储是放在一块的，初始化的全局变量和静态变量在一块区域，未初始化的全局变量和未初始化的静态变量在相邻的另一块区域。- 程序结束释放

4.另外还有一个专门放常量的地方。- 程序结束释放

5 程序代码区，存放2进制代码。

在函数体中定义的变量通常是在栈上，用malloc， calloc， realloc等分配内存的函数分配得到的就是在堆上。在所有函数体外定义的是全局量，加了static修饰符后不管在哪里都存放在全局区(静态区)，在所有函数体外定义的static变量表示在该文件中有效，不能extern到别的文件用，在函数体内定义的static表示只在该函数体内有效。另外，函数中的"adgfdf"这样的字符串存放在常量区。

**1.new、delete、malloc、free关系**

delete会调用对象的析构函数,和new对应free只会释放内存，new调用构造函数。malloc与free是C++/C语言的标准库函数，new/delete是C++的运算符。它们都可用于申请动态内存和释放内存。对于非内部数据类型的对象而言，光用maloc/free无法满足动态对象的要求。对象在创建的同时要自动执行构造函数，对象在消亡之前要自动执行析构函数。由于malloc/free是库函数而不是运算符，不在编译器控制权限之内，不能够把执行构造函数和析构函数的任务强加于malloc/free。因此C++语言需要一个能完成动态内存分配和初始化工作的运算符new，以及一个能完成清理与释放内存工作的运算符delete。注意new/delete不是库函数。

**2.delete与 delete []区别**

delete只会调用一次析构函数，而delete[]会调用每一个成员的析构函数。在More Effective C++中有更为详细的解释：“当delete操作符用于数组时，它为每个数组元素调用析构函数，然后调用operator delete来释放内存。”delete与new配套，delete []与new []配套

MemTest \*mTest1=new MemTest[10];

MemTest \*mTest2=new MemTest;

Int \*pInt1=new int [10];

Int \*pInt2=new int;

delete[]pInt1; //-1-

delete[]pInt2; //-2-

delete[]mTest1;//-3-

delete[]mTest2;//-4-

在-4-处报错。

这就说明：对于内建简单数据类型，delete和delete[]功能是相同的。对于自定义的复杂数据类型，delete和delete[]不能互用。delete[]删除一个数组，delete删除一个指针。简单来说，用new分配的内存用delete删除；用new[]分配的内存用delete[]删除。delete[]会调用数组元素的析构函数。内部数据类型没有析构函数，所以问题不大。如果你在用delete时没用括号，delete就会认为指向的是单个对象，否则，它就会认为指向的是一个数组。

**3.C++有哪些性质（面向对象特点）**

封装，继承和多态。

**4.子类析构时要调用父类的析构函数吗？**

析构函数调用的次序是先派生类的析构后基类的析构，也就是说在基类的的析构调用的时候,派生类的信息已经全部销毁了。定义一个对象时先调用基类的构造函数、然后调用派生类的构造函数；析构的时候恰好相反：先调用派生类的析构函数、然后调用基类的析构函数。

**5.多态，虚函数，纯虚函数**

多态：是对于不同对象接收相同消息时产生不同的动作。C++的多态性具体体现在运行和编译两个方面：在程序运行时的多态性通过继承和虚函数来体现；

在程序编译时多态性体现在函数和运算符的重载上；

虚函数：在基类中冠以关键字 virtual 的成员函数。 它提供了一种接口界面。允许在派生类中对基类的虚函数重新定义。

纯虚函数的作用：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它进行定义。作为接口而存在 纯虚函数不具备函数的功能，一般不能直接被调用。

从基类继承来的纯虚函数，在派生类中仍是虚函数。如果一个类中至少有一个纯虚函数，那么这个类被称为抽象类（abstract class）。

抽象类中不仅包括纯虚函数，也可包括虚函数。抽象类必须用作派生其他类的基类，而不能用于直接创建对象实例。但仍可使用指向抽象类的指针支持运行时多态性。

**6.求下面函数的返回值（微软）**

int func(x)

{

int countx = 0;

while(x)

{

countx ++;

x = x&(x-1);

}

return countx;

}

假定x = 9999。 答案：8

思路：将x转化为2进制，看含有的1的个数。

**7.什么是“引用”？申明和使用“引用”要注意哪些问题？**

答：引用就是某个目标变量的“别名”(alias)，对应用的操作与对变量直接操作效果完全相同。申明一个引用的时候，切记要对其进行初始化。引用声明完毕后，相当于目标变量名有两个名称，即该目标原名称和引用名，不能再把该引用名作为其他变量名的别名。声明一个引用，不是新定义了一个变量，它只表示该引用名是目标变量名的一个别名，它本身不是一种数据类型，因此引用本身不占存储单元，系统也不给引用分配存储单元。不能建立数组的引用。

**8.将“引用”作为函数参数有哪些特点？**

（1）传递引用给函数与传递指针的效果是一样的。这时，被调函数的形参就成为原来主调函数中的实参变量或对象的一个别名来使用，所以在被调函数中对形参变量的操作就是对其相应的目标对象（在主调函数中）的操作。

（2）使用引用传递函数的参数，在内存中并没有产生实参的副本，它是直接对实参操作；而使用一般变量传递函数的参数，当发生函数调用时，需要给形参分配存储单元，形参变量是实参变量的副本；如果传递的是对象，还将调用拷贝构造函数。因此，当参数传递的数据较大时，用引用比用一般变量传递参数的效率和所占空间都好。

（3）使用指针作为函数的参数虽然也能达到与使用引用的效果，但是，在被调函数中同样要给形参分配存储单元，且需要重复使用"\*指针变量名"的形式进行运算，这很容易产生错误且程序的阅读性较差；另一方面，在主调函数的调用点处，必须用变量的地址作为实参。而引用更容易使用，更清晰。

**9.在什么时候需要使用“常引用”？**

如果既要利用引用提高程序的效率，又要保护传递给函数的数据不在函数中被改变，就应使用常引用。常引用声明方式：const 类型标识符 &引用名=目标变量名；

例1

int a ;

const int &ra=a;

ra=1; //错误

a=1; //正确

例2

string foo( );

void bar(string & s);

那么下面的表达式将是非法的：

bar(foo( ));

bar("hello world");

原因在于foo( )和"hello world"串都会产生一个临时对象，而在C++中，这些临时对象都是const类型的。因此上面的表达式就是试图将一个const类型的对象转换为非const类型，这是非法的。引用型参数应该在能被定义为const的情况下，尽量定义为const 。

**10.将“引用”作为函数返回值类型的格式、好处和需要遵守的规则?**

格式：类型标识符 &函数名（形参列表及类型说明）{ //函数体 }

好处：在内存中不产生被返回值的副本；（注意：正是因为这点原因，所以返回一个局部变量的引用是不可取的。因为随着该局部变量生存期的结束，相应的引用也会失效，产生runtime error!

注意事项：

（1）不能返回局部变量的引用。这条可以参照Effective C++[1]的Item 31。主要原因是局部变量会在函数返回后被销毁，因此被返回的引用就成为了"无所指"的引用，程序会进入未知状态。

（2）不能返回函数内部new分配的内存的引用。这条可以参照Effective C++[1]的Item 31。虽然不存在局部变量的被动销毁问题，可对于这种情况（返回函数内部new分配内存的引用），又面临其它尴尬局面。例如，被函数返回的引用只是作为一个临时变量出现，而没有被赋予一个实际的变量，那么这个引用所指向的空间（由new分配）就无法释放，造成memory leak。

（3）可以返回类成员的引用，但最好是const。这条原则可以参照Effective C++[1]的Item 30。主要原因是当对象的属性是与某种业务规则（business rule）相关联的时候，其赋值常常与某些其它属性或者对象的状态有关，因此有必要将赋值操作封装在一个业务规则当中。如果其它对象可以获得该属性的非常量引用（或指针），那么对该属性的单纯赋值就会破坏业务规则的完整性。

（4）流操作符重载返回值申明为“引用”的作用：

流操作符<<和>>，这两个操作符常常希望被连续使用，例如：cout << "hello" << endl;　因此这两个操作符的返回值应该是一个仍然支持这两个操作符的流引用。可选的其它方案包括：返回一个流对象和返回一个流对象指针。但是对于返回一个流对象，程序必须重新（拷贝）构造一个新的流对象，也就是说，连续的两个<<操作符实际上是针对不同对象的！这无法让人接受。对于返回一个流指针则不能连续使用<<操作符。因此，返回一个流对象引用是惟一选择。这个唯一选择很关键，它说明了引用的重要性以及无可替代性，也许这就是C++语言中引入引用这个概念的原因吧。

赋值操作符=。这个操作符象流操作符一样，是可以连续使用的，例如：x = j = 10;或者(x=10)=100;赋值操作符的返回值必须是一个左值，以便可以被继续赋值。因此引用成了这个操作符的惟一返回值选择。

＃include<iostream.h>

int &put(int n);

int vals[10];

int error=-1;

void main()

{

put(0)=10; //以put(0)函数值作为左值，等价于vals[0]=10;

put(9)=20; //以put(9)函数值作为左值，等价于vals[9]=20;

cout<<vals[0];

cout<<vals[9];

}

int &put(int n)

{

if (n>=0 && n<=9 ) return vals[n];

else { cout<<"subscript error"; return error; }

}

（5）在另外的一些操作符中，却千万不能返回引用：+-\*/ 四则运算符。它们不能返回引用，Effective C++[1]的Item23详细的讨论了这个问题。主要原因是这四个操作符没有side effect，因此，它们必须构造一个对象作为返回值，可选的方案包括：返回一个对象、返回一个局部变量的引用，返回一个new分配的对象的引用、返回一个静态对象引用。根据前面提到的引用作为返回值的三个规则，第2、3两个方案都被否决了。静态对象的引用又因为((a+b) == (c+d))会永远为true而导致错误。所以可选的只剩下返回一个对象了。

**11、结构与联合有和区别？**

(1). 结构和联合都是由多个不同的数据类型成员组成, 但在任何同一时刻, 联合中只存放了一个被选中的成员（所有成员共用一块地址空间）, 而结构的所有成员都存在（不同成员的存放地址不同）。

(2). 对于联合的不同成员赋值, 将会对其它成员重写, 原来成员的值就不存在了, 而对于结构的不同成员赋值是互不影响的。

**12、试写出程序结果：**

int  a=4;

int  &f(int  x)

{    a=a+x;

      return  a;

}

int main(void)

{    int   t=5;

     cout<<f(t)<<endl;  a = 9

    f(t)=20;             a = 20

    cout<<f(t)<<endl;     t = 5,a = 20  a = 25

     t=f(t);                a = 30 t = 30

    cout<<f(t)<<endl;  }    t = 60

}

**13.重载（overload)和重写(overried，有的书也叫做“覆盖”）的区别？**

常考的题目。从定义上来说：

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）。

重写：是指子类重新定义父类虚函数的方法。

从实现原理上来说：

重载：编译器根据函数不同的参数表，对同名函数的名称做修饰，然后这些同名函数就成了不同的函数（至少对于编译器来说是这样的）。如，有两个同名函数：function func(p:integer):integer;和function func(p:string):integer;。那么编译器做过修饰后的函数名称可能是这样的：int\_func、str\_func。对于这两个函数的调用，在编译器间就已经确定了，是静态的。也就是说，它们的地址在编译期就绑定了（早绑定），因此，重载和多态无关！

重写：和多态真正相关。当子类重新定义了父类的虚函数后，父类指针根据赋给它的不同的子类指针，动态的调用属于子类的该函数，这样的函数调用在编译期间是无法确定的（调用的子类的虚函数的地址无法给出）。因此，这样的函数地址是在运行期绑定的（晚绑定）。

**14.有哪几种情况只能用intialization list 而不能用assignment?**

答案：当类中含有const、reference 成员变量；基类的构造函数都需要初始化表。

**15. C++是不是类型安全的？**

答案：不是。两个不同类型的指针之间可以强制转换（用reinterpret cast)。C#是类型安全的。

**16. main 函数执行以前，还会执行什么代码？**

答案：全局对象的构造函数会在main 函数之前执行。

**17. 描述内存分配方式以及它们的区别?**

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。

**18.分别写出BOOL,int,float,指针类型的变量a 与“零”的比较语句。**

答案：

BOOL : if ( !a ) or if(a)

int : if ( a == 0)

float : const EXPRESSION EXP = 0.000001

if ( a < EXP && a >-EXP)

pointer : if ( a != NULL) or if(a == NULL)

**19.请说出const与#define 相比，有何优点？**

答案：

const作用：定义常量、修饰函数参数、修饰函数返回值三个作用。被Const修饰的东西都受到强制保护，可以预防意外的变动，能提高程序的健壮性。

1） const 常量有数据类型，而宏常量没有数据类型。编译器可以对前者进行类型安全检查。而对后者只进行字符替换，没有类型安全检查，并且在字符替换可能会产生意料不到的错误。

2） 有些集成化的调试工具可以对const 常量进行调试，但是不能对宏常量进行调试。

**20.简述数组与指针的区别？**

数组要么在静态存储区被创建（如全局数组），要么在栈上被创建。指针可以随时指向任意类型的内存块。

(1)修改内容上的差别

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

(2) 用运算符sizeof 可以计算出数组的容量（字节数）。sizeof(p),p 为指针得到的是一个指针变量的字节数，而不是p 所指的内存容量。C++/C 语言没有办法知道指针所指的内存容量，除非在申请内存时记住它。注意当数组作为函数的参数进行传递时，该数组自动退化为同类型的指针。

char a[] = "hello world";

char \*p = a;

cout<< sizeof(a) << endl; // 12 字节

cout<< sizeof(p) << endl; // 4 字节

计算数组和指针的内存容量

void Func(char a[100])

{

cout<< sizeof(a) << endl; // 4 字节而不是100 字节

}

**第21题： int (\*s[10])(int) 表示的是什么？**

int (\*s[10])(int) 函数指针数组，每个指针指向一个int func(int param)的函数。

**第22题：栈内存与文字常量区**

           char str1[] = "abc";  
　　char str2[] = "abc";

　　const char str3[] = "abc";  
　　const char str4[] = "abc";

　　const char \*str5 = "abc";  
　　const char \*str6 = "abc";

　　char \*str7 = "abc";  
　　char \*str8 = "abc";

　　cout << ( str1 == str2 ) << endl;//0  分别指向各自的栈内存  
　　cout << ( str3 == str4 ) << endl;//0  分别指向各自的栈内存  
　　cout << ( str5 == str6 ) << endl;//1指向文字常量区地址相同

　　cout << ( str7 == str8 ) << endl;//1指向文字常量区地址相同

　　结果是：0 0 1 1

　　解答：str1,str2,str3,str4是数组变量，它们有各自的内存空间；而str5,str6,str7,str8是指针，它们指向相同的常量区域。

**第23题：将程序跳转到指定内存地址**

**要对绝对地址0x100000赋值，我们可以用(unsigned int\*)0x100000 = 1234;那么要是想让程序跳转到绝对地址是0x100000去执行，应该怎么做？**

**\*((void (\*)( ))0x100000 ) ( );  
　　首先要将0x100000强制转换成函数指针,即:  
　　(void (\*)())0x100000  
　　然后再调用它:  
　　\*((void (\*)())0x100000)();  
　　用typedef可以看得更直观些:  
　　typedef void(\*)() voidFuncPtr;  
　　\*((voidFuncPtr)0x100000)();**

**第24题：int id[sizeof(unsigned long)];这个对吗？为什么？**

 答案:正确 这个 sizeof是编译时运算符，编译时就确定了  ,可以看成和机器有关的常量。

**第25题：引用与指针有什么区别？**

【参考答案】                           
1) 引用必须被初始化，指针不必。

2) 引用初始化以后不能被改变，指针可以改变所指的对象。

3) 不存在指向空值的引用，但是存在指向空值的指针。

**第26题：const  与 #define 的比较 ，const有什么优点?**

 【参考答案】

（1） const 常量有数据类型，而宏常量没有数据类型。编译器可以对前者进行类型安全检查。而对后者只进行字符替换，没有类型安全检查，并且在字符替换可能会产生意料不到的错误（边际效应） 。

（2）  有些集成化的调试工具可以对 const 常量进行调试，但是不能对宏常量进行调试。

第27题：复杂声明

void \* ( \* (\*fp1)(int))[10];

float (\*(\* fp2)(int,int,int))(int);

int (\* ( \* fp3)())[10]();

分别表示什么意思？  
【标准答案】

1.void \* ( \* (\*fp1)(int))[10];   fp1是一个指针，指向一个函数，这个函数的参数为int型，函数的返回值是一个指针，这个指针指向一个数组，这个数组有10个元素，每个元素是一个void\*型指针。

2.float (\*(\* fp2)(int,int,int))(int);   fp2是一个指针，指向一个函数，这个函数的参数为3个int型，函数的返回值是一个指针，这个指针指向一个函数，这个函数的参数为int型，函数的返回值是float型。  
  
3.int (\* ( \* fp3)())[10]();   fp3是一个指针，指向一个函数，这个函数的参数为空，函数的返回值是一个指针，这个指针指向一个数组，这个数组有10个元素，每个元素是一个指针，指向一个函数，这个函数的参数为空，函数的返回值是int型。

**第28题：内存的分配方式有几种?**

【参考答案】

一、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。

二、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中，效率很高，但是分配的内存容量有限。

三、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我们决定，使用非常灵活，但问题也最多。

**第29题：基类的析构函数不是虚函数，会带来什么问题？**

【参考答案】派生类的析构函数用不上，会造成资源的泄漏。

**第30题：全局变量和局部变量有什么区别？是怎么实现的？操作系统和编译器是怎么知道的？**

【参考答案】

生命周期不同：

全局变量随主程序创建和创建，随主程序销毁而销毁；局部变量在局部函数内部，甚至局部循环体等内部存在，退出就不存在；

使用方式不同：通过声明后全局变量程序的各个部分都可以用到；局部变量只能在局部使用；分配在栈区。

操作系统和编译器通过内存分配的位置来知道的，全局变量分配在全局数据段并且在程序开始运行的时候被加载。局部变量则分配在堆栈里面 。

红黑树 c++设计模式 数据库有几种引擎