1. C和C++的区别：
2. 宏观上：C:面向过程编程，通过main函数调用各个子函数，程序运行的顺序是程序员事先决定好的。C++面向对象编程，将数据和数据的操作放在类中，以事件来驱动对象处理。主要区别是解决问题的思想方法不同。
3. 语法上：类型检查更为严格（C中的const可以用指针修改，C++中的不能），增加了异常处理，运算符重载，标准模板库，命名空间等。
4. 变量声明的位置：c中变量声明和代码是分开的，必须在函数开始处声明。C++中变量可以在任意处声明，只要保证先声明后使用的原则就可以。
5. struct:C++中的struct结构体支持成员函数的定义，C中的不支持。
6. bool类型：C++中才有。
7. 函数：C语言函数没有默认参数值，C++中有
8. 引用：C++中有引用，函数调用可以作为左值。
9. C++中struct和Class的区别：
10. 默认的继承访问权限。struct是public的，class是private的。
11. C++的缺点：
    1. 内容多，学起来难
    2. 没有垃圾回收机制，可能引起内存泄漏。
12. 线程和进程：
    1. 进程有独立的地址空间，线程只是一个进程中不同的执行路径，线程有自己的堆栈和局部变量，但线程之间没有单独的地址空间，一个线程死掉就等于整个进程死掉，所以多进程的程序要比多线程的程序健壮。进程切换时，耗费资源较大，效率差，但对于一些要求同时进行并且又要共享某些变量的并发操作，只能用线程，不能用进程。
    2. 一个程序至少有一个进程，一个进程至少有一个线程。多个线程共享内存。
    3. 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。
    4. 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。
13. TCP和UDP：
    1. TCP可靠，稳定，TCP在传递数据之前，会有三次握手来建立连接，有确认，窗口，重传，拥塞控制机制，在数据传完之后，还会断开连接用来节约系统资源。缺点是慢，占用系统资源高。
    2. UDP的优点：快，无状态传输协议。缺点：不可靠，不稳定，因为UDP没有TCP那些可靠的机制。
    3. TCP用于一些要求可靠的应用，比如HTTP，HTTPS，FTP等传输文件的协议，POP、SMTP等邮件传输协议。
    4. UDP用于对通讯质量要求不高，要求通迅速度尽量快。如QQ语音，QQ视频等。
    5. 小结：基于连接和无连接，对系统资源要求多和少，TCP保证数据正确性，UDP可能丢包。
14. 堆和栈：
    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；
    2. 堆区（heap），由程序员分配释放，与数据结构中的堆是两回事。
    3. 全局区（static），存放全局变量和静态变量。
15. static:
    1. 静态全局变量：在全局区分配内存，未初始化的静态全局变量会被自动初始化为0，静态全局变量不能为其它文件所用，其它文件可以定义相同名字的变量，不会发生冲突。
    2. 静态函数:在函数的返回类型前加上static关键字，它只能在声明它的文件当中可见，不能被其它文件使用。
    3. 静态数据成员：是类的成员，而非对象，只分配一次内存，供所有对象共用，静态数据成员要在类内声明，类外定义和初始化。有两种访问方式，＜类对象名＞.＜静态数据成员名＞ 或 ＜类类型名＞::＜静态数据成员名＞。好处：各个对象有相同的某项属性时，可以节省空间。
    4. 静态成员函数：不具有this指针，从这个意义来讲，它无法访问属于类对象的非静态数据成员，也无法访问非静态成员函数
    5. 父类中定义的static变量与其子类共用。
16. const:
    1. \*在const左边，则const修饰指针，\*在const右边，则const修饰指针指向的值。
17. 函数指针：
    1. 含义：一个指针，指向一个函数。
    2. 使用方法：int (\*pFun)(int a, int b)或者typedef int(\*pFun)(int,int);
    3. 用途：将函数作为参数传递给函数。
18. strcpy和memcpy:
    1. 复制内容不同：strcpy只用于字符串复制，memcpy提供了一般内存的复制，对复制的内容没有限制。
    2. 复制方法不同：strcpy不用指定长度，它遇到字符的串结束符”\0”才结束。memcpym则是根据其第3个参数决定复制的长度。
    3. 用途不同：通常在复制字符串时用strcpy，复制其他类型数据时一般用memcpy。
19. 实现一个strcpy和memcpy:
    1. char \* Strcpy(char \* dest, const char \* src) // 实现src到dest的复制

{

char \*r = dest;

if(dest==NULL||src==NULL)

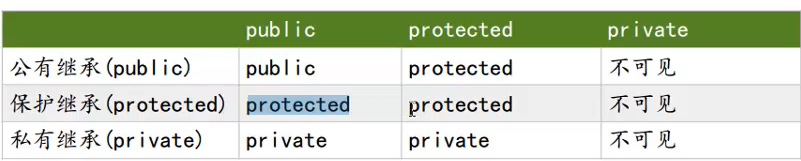
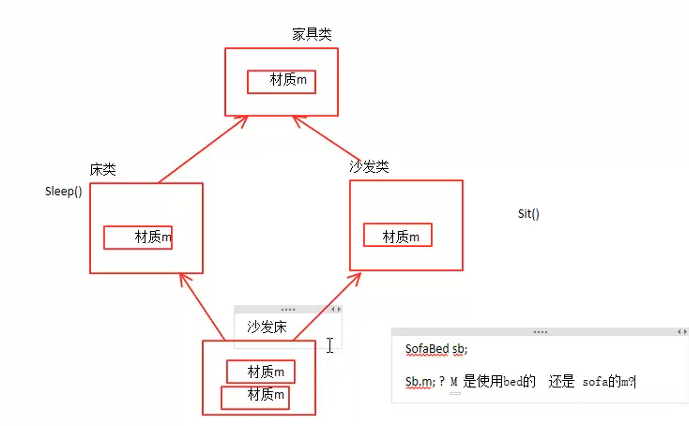
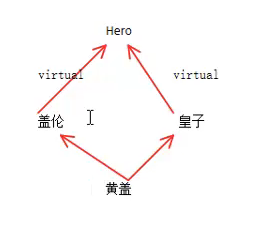
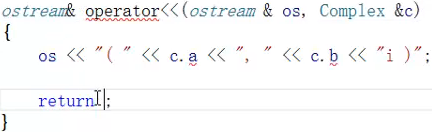
return NULL;

while((\*r++=\*src++)!='\0');

return dest; //返回值是为了支持链式表达

}

* 1. ####################

1. size\_t:
   1. 一个基本的无符号整数的c/c++类型，它是sizeof操作符返回的结果类型.
   2. vector使用的下标为size\_t.
2. 高内聚，低耦合：
   1. 函数功能单一，函数与其它函数关联度低。
3. 类与类的关系：
   1. B has A
   2. B use A
   3. B is A
4. 访问继承权限， public,protected,private:
   1. 单个类中：类内外可以访问，类内可以访问，类内可以访问。
5. 派生类成员的标识和访问：
   1. 
   2. 只要是父类中的private成员，不管是什么继承方式，儿子都访问不了
   3. 如果是公有继承，儿子中的访问控制权限不变。
   4. 如果是保护继承，儿子中父亲的public在这变成了protected，
   5. 如果是私有继承，儿子中的父亲的public和protected都变成private.
6. 子类对象和父类对象：
   1. 子类对象可以当作父类对象使用
   2. 子类对象可以直接赋值给父类对象
   3. 子类对象可以直接初始化父类对象
   4. 父类指针可以直接指向子类对象（父类指针可以实现父类功能，因此实现子类功能也没问题）
   5. 父类引用可以直接引用子类对象
7. 父类和子类出现重名变量：
   1. 使用父类域调用父类变量
8. 多继承和虚继承：
   1. 多继承为C++特性，可能会出现菱形形状，引起歧义，
   2. 虚继承可解决这个问题：将父亲类继承爷爷类改成虚继承，防止儿子在多继承我的时候，出现爷爷中的变量会拷贝多份。
9. 多态和虚函数：
   1. 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。
   2. 虚函数实现多态：父类中的某个函数是虚函数，而在子类中重写了这个函数，即可实现多态，用指向基类的指针分别调用父类和派生类的方法，即传递父类对象调用父类方法，传递子类对象调用子类方法。
   3. 若多态要实现析构函数时，需要把父类析构函数定义为虚析构函数。
   4. 多态发生的三个必要条件：
      1. 要有继承
      2. 要有虚函数重写
      3. 父类指针或引用指向子类对象
   5. 多态原理：虚函数表
10. memset:
    1. 作用：初始化函数，将某一块内存中的内容全部设置为指定的值，这个函数通常为新申请的内存做初始化工作。
    2. void \*memset(void \*s,int ch,size\_t n);
11. 重载、重写、重定义：
    1. 重载在同一个作用域。
    2. 重定义，发生在父类和子类之间。
       1. 普通函数重定义：如果父类的普通成员函数，被子类重写，说是重定义。
       2. 虚函数重写：如果父类的虚函数，被子类重写，就是虚函数重写，这个函数会发生多态。如果说一个类有virtual虚函数关键字，在编译器给这个对象开辟空间的时候，会默认增加一个指针，vptr，指向一个虚函数表。传入对象时，vptr指针会去对应的虚函数表中查找函数。
12. 抽象类和纯虚函数
    1. 如果一个类具有一个纯虚函数，称此类为抽象类，抽象类不能实例化。
    2. 纯虚函数类似于java中的接口。
    3. 如果一个普通类要继承一个抽象类的话，必须实现此纯虚函数，若不重写，则子类仍为抽象类。
    4. 架构师写抽象类。
13. 左移右移操作符重载：
    1. 
    2. <<和>>最好定义为友员函数。

STL

1. 函数模板
   1. 作用：示例如函数交换时使用模板方便，编写代码时可以忽略具体类型。
   2. 函数模板不允许自动类型转换，普通函数可以
   3. 机制：编译器并不是把函数模板处理成能够处理任何类型的函数，而是通过具体类型产生不同的函数；编译器会对函数模板进行两次编译，在声明的地方对模板代码本身进行编译，在调用的时候对参数替换后的代码进行编译。
2. 类模板：
   1. 函数模板在调用的时候，可以自动类型推导，而类模板必须显式指定类型。
   2. 类模板外部实现不要滥用友元。
   3. 类模板.h和.cpp分开写时，因为模板的两次编译机制，会造成链接时的错误。
   4. 解决方案：.h和.cpp合起来写，改后缀名为.hpp。
3. 类型转换基本语法：
   1. c风格的强制类型转换，统统是Type b = (Typer) a，不做类型检查，不安全。
   2. static\_cast 用于内置的数据类型，还有具有继承关系的指针或者引用。
   3. dynamic\_cast 用于转换具有继承关系的指针或引用，并且只能由子类型转成父类型。
   4. const\_cast 增加或者去除指针或者引用的const性（不是去除本身的const），而是给到一个新的变量。
   5. reinterpret\_cast 强制类型转换用于进行没有任何关联之间的转换，比如一个字符指针转换为一个整形数，无关的指针类型，包括函数指针都可以转换。
4. 异常机制：
   1. 跨函数
   2. 必须处理
   3. 栈解旋：触发异常时，异常所有函数的局部变量会触发析构。
5. cin,cout：
   1. 类定义的全局对象
   2. 与缓冲区交互，而非键盘和显示器等硬件。
   3. I/0分为标准I/0和文件I/O。
6. cin重点掌握函数：
   1. cin.get:一次只能读取一个字符
   2. cin.getline：读取一行数据
   3. cin.ignore:忽略当前字符
7. cout重点掌握函数：
   1. cout.flush:刷新缓冲区
   2. cout.put:向缓冲区写字符
8. STL
   1. 分为容器，算法和迭代器，容器和算法之间通过迭代器进行无缝连接。STL几乎所有的代码都采用了模板类或者模板函数。
   2. 容器：分为序列式容器和关联式容器，容器提供迭代器。序列容器：元素按元素摆放。关联容器：元素按容器规则摆放（如set就是基于红黑树实现的关联容器）。
   3. 迭代器：迭代器可以理解为指针，对指针的操作基本都可以对迭代器操作。实际上，迭代器是一个类，这个类封装一个指针。
   4. 算法：STL提供了大约100个实现算法的模板函数。
9. string:
   1. string封装了char\*，是一个char\*型的容器。
   2. string封装了实用的成员方法，如find,copy,delete,replace
   3. 常用方法：+=，append,find,rfind,replace,compare,substr,insert,erase
10. vector
    1. for each遍历，迭代器遍历。
    2. vector是个动态数组，空间不足时会重新申请一块更大的空间，将旧空间数据拷贝到新空间，然后释放旧空间。
    3. vector是单口容器，所以在尾端插入和删除元素效率较高，在指定位置插入，势必会引起数据元素移动，效率较低。
    4. vector支持随机访问，支持数组下标的一般都支持随机访问。
11. deque
    1. 特性：双端插入和删除元素效率较高；指定位置插入效率低。
12. stack
    1. 先进后出，push(),pop(),top().
13. list
    1. 由一系列结点组成，结点包含两个域，一个数据域，一个指针域。
    2. 内存非连续，添加删除元素的时间复杂度都是常数项，不需要移动元素，比数组添加删除效率高。
    3. 不支持随机访问
14. 二叉搜索树
    1. 二叉树中的节点按照一定的规则排序，使得对二叉树中元素访问更加高效。
    2. 任何节点的元素值一定大于其左子树中的每一个节点的元素值，并且小于其右子树的值。
    3. 平衡二叉树：左右子树深度相差不超过1。
15. set/multiset
    1. 所有元素会根据元素的值自动进行排序。
    2. set以红黑树为底层机制，查找效率高，会自动排序。
    3. set元素不能重复，multiset可以。
    4. 查找操作：find(key),lower\_bound(keyElem),upper\_bound(keyElem),equal\_range(keyElem)