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 operatordelete 来释放内存。" delete 与 New 配套，delete []与 new []配套

```
MemTest*mTest1=newMemTest[10];
MemTest*mTest2=newMemTest;
int*pInt1=newint[10];
int*pInt2=newint;
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 C++ JAVA共同点，不同之处？

4.继承优缺点。

类继承是在编译时刻静态定义的,且可直接使用,类继承可以较方便地改变父类的实现。但是类继承也有一些不足之处。首先，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。更糟的是，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。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。

（待补充）

5.C++有哪些性质（面向对象特点）

封装，继承和多态。

在面向对象程序设计语言中，封装是利用可重用成分构造软件系统的特性，它不仅支持系统的可重用性，而且还有利于提高系统的可扩充性；消息传递可以实现发送一个通用的消

息而调用不同的方法；封装是实现信息隐蔽的一种技术，其目的是使类的定义和实现分离。

6.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
JAVA 无析构函数深拷贝和浅拷贝

7.多态，虚函数，纯虚函数

8.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。

9.什么是“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。

10.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。

11.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 。

12.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。

例 3

```
＃i nclude <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。

13.“引用”与多态的关系？

引用是除指针外另一个可以产生多态效果的手段。这意味着，一个基类的引用可以指向它的派生类实例。例 4

Class A; Class B : Class A{...}; B b; A& ref = b;

14.“引用”与指针的区别是什么？

指针通过某个指针变量指向一个对象后，对它所指向的变量间接操作。程序中使用指针，程序的可读性差；而引用本身就是目标变量的别名，对引用的操作就是对目标变量的操作。此外，就是上面提到的对函数传 ref 和 pointer 的区别。

15.什么时候需要“引用”？

流操作符<<和>>、赋值操作符=的返回值、拷贝构造函数的参数、赋值操作符=的参数、其它情况都推荐使用引用。以上 2-8 参考：http://develop.csai.cn/c/NO0000021.htm

16.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。

17.面关于“联合”的题目的输出？

a)

```
#i nclude <stdio.h>
union
{
int i;
char x[2];
}a;

void main()
{
a.x[0] = 10;
a.x[1] = 1;
printf("%d",a.i);
}
```

答案：266 (低位低地址，高位高地址，内存占用情况是 Ox010A）

b)

```
main()
{
    union{                  /*定义一个联合*/
        int i;
        struct{             /*在联合中定义一个结构*/
            char first;
            char second;
        }half;
    }number;
    number.i=0x4241;          /*联合成员赋值*/
    printf("%c%cn", number.half.first, number.half.second);
    number.half.first='a';    /*联合中结构成员赋值*/
    number.half.second='b';
    printf("%xn", number.i);
    getch();
}
```

答案： AB (0x41 对应'A',是低位；Ox42 对应'B',是高位）
6261 (number.i 和 number.half 共用一块地址空间）

18.关联、聚合(Aggregation)以及组合(Composition)的区别？

涉及到 UML 中的一些概念：关联是表示两个类的一般性联系，比如“学生”和“老师”就

是一种关联关系；聚合表示 has-a 的关系，是一种相对松散的关系，聚合类不需要对被聚合类负责，如下图所示，用空的菱形表示聚合关系：从实现的角度讲，聚合可以表示为:

class A {...} class B { A* a; .....}

而组合表示 contains-a 的关系，关联性强于聚合：组合类与被组合类有相同的生命周期，组合类要对被组合类负责，采用实心的菱形表示组合关系：实现的形式是:

class A{...} class B{ A a; ...}

参考文章：http://www.cnitblog.com/Lily/archive/2006/02/23/6860.html

http://www.vckbase.com/document/viewdoc/?id=422

19.面向对象的三个基本特征，并简单叙述之？

1. 封装：将客观事物抽象成类，每个类对自身的数据和方法实行 protection(private, protected,public)

2. 继承：广义的继承有三种实现形式：实现继承（指使用基类的属性和方法而无需额外编码的能力）、可视继承（子窗体使用父窗体的外观和实现代码）、接口继承（仅使用属性和方法，实现滞后到子类实现）。前两种（类继承）和后一种（对象组合=>接口继承以及纯虚函数）构成了功能复用的两种方式。

3. 多态：是将父对象设置成为和一个或更多的他的子对象相等的技术，赋值之后，父对象就可以根据当前赋值给它的子对象的特性以不同的方式运作。简单的说，就是一句话：允许将子类类型的指针赋值给父类类型的指针。

20.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）。

21.多态的作用？

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。

22.Ado与Ado.net的相同与不同？

除了“能够让应用程序处理存储于 DBMS 中的数据“这一基本相似点外，两者没有太多共同之处。但是 Ado 使用 OLE DB 接口并基于微软的 COM 技术，而 ADO.NET 拥有自

己的 ADO.NET 接口并且基于微软的.NET 体系架构。众所周知.NET 体系不同于 COM 体系，ADO.NET 接口也就完全不同于 ADO 和 OLE DB 接口，这也就是说 ADO.NET 和 ADO 是两种数据访问方式。ADO.net 提供对 XML 的支持。

23.New delete 与malloc free 的联系与区别?

答案：都是在堆(heap)上进行动态的内存操作。用 malloc 函数需要指定内存分配的字节数并且不能初始化对象，new 会自动调用对象的构造函数。delete 会调用对象的 destructor，而 free 不会调用对象的 destructor.

24.#define DOUBLE(x) x+x ，i = 5*DOUBLE(5)； i 是多少？

答案：i 为 30。

25.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。

26. 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。

27. 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。

28. 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。

29.struct 和 class 的区别

答案：struct 的成员默认是公有的，而类的成员默认是私有的。struct 和 class 在其他方面是功能相当的。从感情上讲，大多数的开发者感到类和结构有很大的差别。感觉上结构仅仅象一堆缺乏封装和功能的开放的内存位，而类就象活的并且可靠的社会成员，它有智能服务，有牢固的封装屏障和一个良好定义的接口。既然大多数人都这么认为，那么只有在你的类有很少的方法并且有公有数据（这种事情在良好设计的系统中是存在的!）时，你也许应该使用 struct 关键字，否则，你应该使用 class 关键字。

30.当一个类A 中没有任何成员变量与成员函数,这时sizeof(A)的值是多少？

答案：如果不是零，请解释一下编译器为什么没有让它为零。（Autodesk）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。

31. 在 8086 汇编下，逻辑地址和物理地址是怎样转换的？（Intel）

答案：通用寄存器给出的地址，是段内偏移地址，相应段寄存器地址*10H+通用寄存器内地址，就得到了真正要访问的地址。

32. 比较C++中的 4 种类型转换方式？

请参考：http://blog.csdn.net/wfwd/archive/2006/05/30/763785.aspx，重点是 static_cast, dynamic_cast 和 reinterpret_cast 的区别和应用。

dynamic_casts 在帮助你浏览继承层次上是有限制的。它不能被用于缺乏虚函数的类型上，它被用于安全地沿着类的继承关系向下进行类型转换。如你想在没有继承关系的类型中进行转换，你可能想到 static_cast

33.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)

34.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。

35.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
}
```

36.类成员函数的重载、覆盖和隐藏区别？

答案：a.成员函数被重载的特征：

（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。

b.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。

c.“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）

37.求出两个数中的较大这

There are two int variables: a and b, don’t use “if”, “? :”, “switch”or other judgement statements, find out the biggest one of the two numbers.

答案：((a + b) + abs(a - b)) / 2

38.如何打印出当前源文件的文件名以及源文件的当前行号？

答案：

```
cout << __FILE__ ;
cout<<__LINE__ ;
```
__FILE__和__LINE__是系统预定义宏，这种宏并不是在某个文件中定义的，而是由编译器定义的。

39. main 主函数执行完毕后，是否可能会再执行一段代码，给出说明？

答案：可以，可以用_onexit 注册一个函数，它会在 main 之后执行 int fn1(void), fn2(void), fn3(void), fn4 (void);

```
void main( void )
{
String str("zhanglin");
_onexit( fn1 );
_onexit( fn2 );
_onexit( fn3 );
```

```
_onexit( fn4 );
printf( "This is executed first.n" );
}
int fn1()
{
printf( "next.n" );
return 0;
}
int fn2()
{
printf( "executed " );
return 0;
}
int fn3()
{
printf( "is " );
return 0;
}
int fn4()
{
printf( "This " );
return 0;
}
```

The _onexit function is passed the address of a function (func) to be called when the program terminates normally. Successive calls to _onexit create a register of functions that are executed in LIFO (last-in-first-out) order. The functions passed to _onexit cannot take parameters.

40.如何判断一段程序是由C 编译程序还是由C++编译程序编译的？

答案：

```
#ifdef __cplusplus
cout<<"c++";
#else
cout<<"c";
#endif
```

41.文件中有一组整数，要求排序后输出到另一个文件中

答案：

```
＃i nclude<iostream>
＃i nclude<fstream>
using namespace std;

void Order(vector<int>& data) //bubble sort
{
```

```
int count = data.size() ;
int tag = false ; //  设置是否需要继续冒泡的标志位
for ( int i = 0 ; i < count ; i++)
{
for ( int j = 0 ; j < count - i - 1 ; j++)
{
if ( data[j] > data[j+1])
{
tag = true ;
int temp = data[j] ;
data[j] = data[j+1] ;
data[j+1] = temp ;
}
}
if ( !tag )
break ;
}
}

void main( void )
{
vector<int>data;
ifstream in("c:\data.txt");
if ( !in)
{
cout<<"file error!";
exit(1);
}
int temp;
while (!in.eof())
{
in>>temp;
data.push_back(temp);
}
in.close(); //关闭输入文件流
Order(data);
ofstream out("c:\result.txt");
if ( !out)
{
cout<<"file error!";
exit(1);
}
for ( i = 0 ; i < data.size() ; i++)
out<<data[i]<<" ";
```

```
out.close(); //关闭输出文件流
}
```

42.链表题：一个链表的结点结构

```
    struct Node
{
int data ;
Node *next ;
};
typedef struct Node Node ;
```

(1)已知链表的头结点 head,写一个函数把这个链表逆序 (Intel)

```
    Node * ReverseList(Node *head) //链表逆序
{
if ( head == NULL || head->next == NULL )
return head;
Node *p1 = head ;
Node *p2 = p1->next ;
Node *p3 = p2->next ;
p1->next = NULL ;
while ( p3 != NULL )
{
p2->next = p1 ;
p1 = p2 ;
p2 = p3 ;
p3 = p3->next ;
}
p2->next = p1 ;
head = p2 ;
return head ;
}
```

(2)已知两个链表 head1 和 head2 各自有序，请把它们合并成一个链表依然有序。(保留所有结点，即便大小相同）

```
Node * Merge(Node *head1 , Node *head2)
{
if ( head1 == NULL)
return head2 ;
if ( head2 == NULL)
return head1 ;
Node *head = NULL ;
Node *p1 = NULL;
Node *p2 = NULL;
if ( head1->data < head2->data )
```

```
{
head = head1 ;
p1 = head1->next;
p2 = head2 ;
}
else
{
head = head2 ;
p2 = head2->next ;
p1 = head1 ;
}
Node *pcurrent = head ;
while ( p1 != NULL && p2 != NULL)
{
if ( p1->data <= p2->data )
{
pcurrent->next = p1 ;
pcurrent = p1 ;
p1 = p1->next ;
}
else
{
pcurrent->next = p2 ;
pcurrent = p2 ;
p2 = p2->next ;
}
}
if ( p1 != NULL )
pcurrent->next = p1 ;
if ( p2 != NULL )
pcurrent->next = p2 ;
return head ;
}
```

(3)已知两个链表 head1 和 head2 各自有序，请把它们合并成一个链表依然有序，这次要求用递归方法进行。 (Autodesk)

答案：

```
Node * MergeRecursive(Node *head1 , Node *head2)
{
if ( head1 == NULL )
return head2 ;
if ( head2 == NULL)
return head1 ;
Node *head = NULL ;
if ( head1->data < head2->data )
```

```
{
head = head1 ;
head->next = MergeRecursive(head1->next,head2);
}
else
{
head = head2 ;
head->next = MergeRecursive(head1,head2->next);
}
return head ;
```

---

41. 分析一下这段程序的输出 (Autodesk)

```
class B
{
public:
B()
{
cout<<"default constructor"<<endl;
}
~B()
{
cout<<"destructed"<<endl;
}
B(int i):data(i)     //B(int) works as a converter ( int -> instance of   B)
{
cout<<"constructed by parameter " << data <<endl;
}
private:
int data;
};

B Play( B b)
{
return b ;
}
```

(1) results:

```
int main(int argc, char* argv[])      constructed by parameter 5
{                                     destructed   B(5)形参析构
B t1 = Play(5); B t2 = Play(t1);          destructed   t1 形参析构
return 0;                                             destructed   t2    注意顺序!
}                                   destructed   t1
```

(2) results:

```
int main(int argc, char* argv[])      constructed by parameter 5
```

```
{                              destructed   B(5)形参析构
B t1 = Play(5); B t2 = Play(10);        constructed by parameter 10
return 0;                                        destructed   B(10)形参析构
}                              destructed   t2   注意顺序!
                                    destructed   t1
```

43.写一个函数找出一个整数数组中，第二大的数 （microsoft）

答案：

```
const int MINNUMBER = -32767 ;
int find_sec_max( int data[] , int count)
{
int maxnumber = data[0] ;
int sec_max = MINNUMBER ;
for ( int i = 1 ; i < count ; i++)
{
if ( data[i] > maxnumber )
{
sec_max = maxnumber ;
maxnumber = data[i] ;
}
else
{
if ( data[i] > sec_max )
sec_max = data[i] ;
}
}
return sec_max ;
}
```

44.写一个在一个字符串(n)中寻找一个子串(m)第一个位置的函数。

KMP 算法效率最好，时间复杂度是 O (n+m), 详见：http://www.zhanglihai.com/blog/c_335_kmp.html

46.多重继承的内存分配问题：

比如有 class A : public class B, public class C {} 那么 A 的内存结构大致是怎么样的？这个是 compiler-dependent 的, 不同的实现其细节可能不同。如果不考虑有虚函数、虚继承的话就相当简单；否则的话，相当复杂。可以参考《深入探索 C++对象模型》，或者：
http://blog.csdn.net/rainlight/archive/2006/03/03/614792.aspx
http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/dnarvc/html/jangrayhood.asp

47.如何判断一个单链表是有环的？（注意不能用标志位，最多只能用两个额外指针）

```
struct node { char val; node* next;}
bool check(const node* head) {} //return false : 无环；true: 有环一种 O（n）的办法就是（搞
```

两个指针，一个每次递增一步，一个每次递增两步，如果有环的话两者必然重合，反之亦然）：

```
bool check(const node* head)
{
   if(head==NULL)   return false;
   node *low=head, *fast=head->next;
   while(fast!=NULL && fast->next!=NULL)
   {
      low=low->next;
      fast=fast->next->next;
      if(low==fast) return true;
   }
   return false;
}
```

48.指针找错题

分析这些面试题，本身包含很强的趣味性;而作为一名研发人员，通过对这些面试题的深入剖析则可进一步增强自身的内功。

2.找错题 试题 1：

以下是引用片段：

```
void test1()   //数组越界
      {
      char string[10];
      char* str1 = "0123456789";
      strcpy( string, str1 );
      }
```

试题 2：

以下是引用片段：

```
  void test2()
      {
      char string[10], str1[10];
      int i;
      for(i=0; i<10; i++)
      {
      str1= 'a';
      }
      strcpy( string, str1 );
      }
```

试题 3：

以下是引用片段：

```
void test3(char* str1)
      {
      char string[10];
      if( strlen( str1 ) <= 10 )
```

{
strcpy(string, str1);
}
}
解答：
试题 1 字符串 str1 需要 11 个字节才能存放下(包括末尾的'\0')，而 string 只有 10 个字节的空间，strcpy 会导致数组越界;对试题 2，如果面试者指出字符数组 str1 不能在数组内结束可以给 3 分;如果面试者指出 strcpy(string,str1)调用使得从 str1 内存起复制到 string 内存起所复制的字节数具有不确定性可以给 7 分，在此基础上指出库函数 strcpy 工作方式的给 10 分;对试题 3，if(strlen(str1) <= 10)应改为 if(strlen(str1) <10)，因为 strlen 的结果未统计'\0'所占用的 1 个字节。剖析：考查对基本功的掌握
(1)字符串以'\0'结尾;
(2)对数组越界把握的敏感度;
(3)库函数 strcpy 的工作方式，

49.如果编写一个标准strcpy函数

总分值为 10，下面给出几个不同得分的答案：2 分 以下是引用片段：

```
void strcpy( char *strDest, char *strSrc )
{
while( (*strDest++ = * strSrc++) != '\0' );
}
```

4 分 以下是引用片段：

```
void strcpy( char *strDest, const char *strSrc )
//将源字符串加 const，表明其为输入参数，加 2 分
{
while( (*strDest++ = * strSrc++) != '\0' );
}
```

7 分 以下是引用片段：

```
void strcpy(char *strDest, const char *strSrc)
{
//对源地址和目的地址加非 0 断言，加 3 分
assert( (strDest != NULL) &&(strSrc != NULL) );
while( (*strDest++ = * strSrc++) != '\0' );
}
```

10 分 以下是引用片段：

```
//为了实现链式操作，将目的地址返回，加 3 分!
char * strcpy( char *strDest, const char *strSrc )
{
assert( (strDest != NULL) &&(strSrc != NULL) );
char *address = strDest;
while( (*strDest++ = * strSrc++) != '\0' );
return address;
}
```

从 2 分到 10 分的几个答案我们可以清楚的看到，小小的 strcpy 竟然暗藏着这么多玄机，

真不是盖的!需要多么扎实的基本功才能写一个完美的 strcpy 啊!

(4)对 strlen 的掌握，它没有包括字符串末尾的'\0'。

读者看了不同分值的 strcpy 版本，应该也可以写出一个 10 分的 strlen 函数了，完美的版本为： int strlen(const char *str) //输入参数 const 以下是引用片段：

```
{
assert( strt != NULL ); //断言字符串地址非 0
int len=0; //注，一定要初始化。
while( (*str++) != '\0' )
{
len++;
}
return len;
}
```

试题 4：以下是引用片段：

```
void GetMemory( char *p )
{
p = (char *) malloc( 100 );
}
void Test( void )
{
char *str = NULL;
GetMemory( str );
strcpy( str, "hello world" );
printf( str );
}
```

试题 5：

以下是引用片段：

```
char *GetMemory( void )
{
char p[] = "hello world";
return p;
}
void Test( void )
{
char *str = NULL;
str = GetMemory();
printf( str );
}
```

试题 6：以下是引用片段：

```
void GetMemory( char **p, int num )
{
*p = (char *) malloc( num );
}
void Test( void )
```

```
{
char *str = NULL;
GetMemory( &str, 100 );
strcpy( str, "hello" );
printf( str );
}
```

试题 7：以下是引用片段：

```
void Test( void )
{
char *str = (char *) malloc( 100 );
strcpy( str, "hello" );
free( str );
... //省略的其它语句
}
```

解答：试题 4 传入中 GetMemory(char *p)函数的形参为字符串指针，在函数内部修改形参并不能真正的改变传入形参的值，执行完

```
char *str = NULL;
GetMemory( str );
```

后的 str 仍然为 NULL;试题 5 中

```
char p[] = "hello world";
return p;
```

的 p[]数组为函数内的局部自动变量，在函数返回后，内存已经被释放。这是许多程序员常犯的错误，其根源在于不理解变量的生存期。

试题 6 的 GetMemory 避免了试题 4 的问题，传入 GetMemory 的参数为字符串指针的指针，但是在 GetMemory 中执行申请内存及赋值语句 tiffanybracelets

```
*p = (char *) malloc( num );
```

后未判断内存是否申请成功，应加上：

```
if ( *p == NULL )
{
...//进行申请内存失败处理

}
```

试题 7 存在与试题 6 同样的问题，在执行

```
char *str = (char *) malloc(100);
```

后未进行内存是否申请成功的判断;另外，在 free(str)后未置 str 为空，导致可能变成一个“野”指针，应加上：

```
str = NULL;
```

试题 6 的 Test 函数中也未对 malloc 的内存进行释放。

剖析：

试题 4～7 考查面试者对内存操作的理解程度，基本功扎实的面试者一般都能正确的回答其中 50~60 的错误。但是要完全解答正确，却也绝非易事。

软件开发网 www.mscto.com

对内存操作的考查主要集中在：

(1)指针的理解;

(2)变量的生存期及作用范围;

(3)良好的动态内存申请和释放习惯。

再看看下面的一段程序有什么错误：

以下是引用片段：

```
swap( int* p1,int* p2 )
    {
    int *p;
    *p = *p1;
    *p1 = *p2;
    *p2 = *p;
    }
```

在 swap 函数中，p 是一个“野”指针，有可能指向系统区，导致程序运行的崩溃。在 VC++ 中 DEBUG 运行时提示错误“Access Violation”。该程序应该改为

以下是引用片段：

```
swap( int* p1,int* p2 )
    {
    int p;
    p = *p1;
    *p1 = *p2;
    *p2 = p;
    }
```

50.String 的具体实现

已知 String 类定义如下：

```
class String
{
public:
String(const char *str = NULL); // 通用构造函数
String(const String &another); // 拷贝构造函数
~ String(); // 析构函数
String & operater =(const String &rhs); // 赋值函数
private:
char *m_data; // 用于保存字符串
};
```

尝试写出类的成员函数实现。

答案：

```
String::String(const char *str)
{
if ( str == NULL ) //strlen 在参数为 NULL 时会抛异常才会有这步判断
{
```

```
m_data = new char[1] ;
m_data[0] = '\0' ;
}
else
{
m_data = new char[strlen(str) + 1];
strcpy(m_data,str);
}

}

String::String(const String &another)
        {
   m_data = new char[strlen(another.m_data) + 1];
   strcpy(m_data,other.m_data);
   }

   String& String::operator =(const String &rhs)
   {
   if ( this == &rhs)
   return *this ;
   delete []m_data; //删除原来的数据，新开一块内存
   m_data = new char[strlen(rhs.m_data) + 1];
   strcpy(m_data,rhs.m_data);
   return *this ;
   }


   String::~String()
   {
   delete []m_data ;
   }
```

51.h头文件中的ifndef/define/endif 的作用？

答：防止该头文件被重复引用。

52.＃i nclude<file.h> 与 ＃i nclude "file.h"的区别？

答：前者是从 Standard Library 的路径寻找和引用 file.h，而后者是从当前工作路径搜寻并引用 file.h。

53.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 语言不支持函数重载。C++提供了 C 连接交换指定符号 extern “C”

解决名字匹配问题。

首先，作为 extern 是 C/C++语言中表明函数和全局变量作用范围（可见性）的关键字，该关键字告诉编译器，其声明的函数和变量可以在本模块或其它模块中使用。

通常，在模块的头文件中对本模块提供给其它模块引用的函数和全局变量以关键字 extern 声明。例如，如果模块 B 欲引用该模块 A 中定义的全局变量和函数时只需包含模块 A 的头文件即可。这样，模块 B 中调用模块 A 中的函数时，在编译阶段，模块 B 虽然找不到该函数，但是并不会报错；它会在连接阶段中从模块 A 编译生成的目标代码中找到此函数

extern "C"是连接申明(linkage declaration),被 extern "C"修饰的变量和函数是按照 C 语言方式编译和连接的,来看看 C++中对类似 C 的函数是怎样编译的：

作为一种面向对象的语言，C++支持函数重载，而过程式语言 C 则不支持。函数被 C++编译后在符号库中的名字与 C 语言的不同。例如，假设某个函数的原型为：

void foo(int x, int y);

该函数被 C 编译器编译后在符号库中的名字为_foo，而 C++编译器则会产生像_foo_int_int 之类的名字（不同的编译器可能生成的名字不同，但是都采用了相同的机制，生成的新名字称为“mangled name”）。

_foo_int_int 这样的名字包含了函数名、函数参数数量及类型信息，C++就是靠这种机制来实现函数重载的。例如，在 C++中，函数 void foo(int x, int y)与 void foo(int x, float y)编译生成的符号是不相同的，后者为_foo_int_float。

同样地，C++中的变量除支持局部变量外，还支持类成员变量和全局变量。用户所编写程序的类成员变量可能与全局变量同名，我们以"."来区分。而本质上，编译器在进行编译时，与函数的处理相似，也为类中的变量取了一个独一无二的名字，这个名字与用户程序中同名的全局变量名字不同。

未加 extern "C"声明时的连接方式

假设在 C++中，模块 A 的头文件如下：

```
// 模块 A 头文件   moduleA.h
#ifndef MODULE_A_H
#define MODULE_A_H
int foo( int x, int y );
#endif
```

在模块 B 中引用该函数：

```
// 模块 B 实现文件　moduleB.cpp
＃i nclude "moduleA.h"
foo(2,3);
```

加 extern "C"声明后的编译和连接方式

加 extern "C"声明后，模块 A 的头文件变为：

```
// 模块 A 头文件　moduleA.h
#ifndef MODULE_A_H
#define MODULE_A_H
extern "C" int foo( int x, int y );
#endif
```

在模块 B 的实现文件中仍然调用 foo(2,3)，其结果是：
（1）模块 A 编译生成 foo 的目标代码时，没有对其名字进行特殊处理，采用了 C 语言的方式；

（2）连接器在为模块 B 的目标代码寻找 foo(2,3)调用时，寻找的是未经修改的符号名_foo。

如果在模块 A 中函数声明了 foo 为 extern "C"类型，而模块 B 中包含的是 extern int foo(int x, int y) ，则模块 B 找不到模块 A 中的函数；反之亦然。

所以，可以用一句话概括 extern “C”这个声明的真实目的（任何语言中的任何语法特性的诞生都不是随意而为的，来源于真实世界的需求驱动。我们在思考问题时，不能只停留在这个语言是怎么做的，还要问一问它为什么要这么做，动机是什么，这样我们可以更深入地理解许多问题）：实现 C++与 C 及其它语言的混合编程。

明白了 C++中 extern "C"的设立动机，我们下面来具体分析 extern "C"通常的使用技巧：

extern "C"的惯用法

（1）在 C++中引用 C 语言中的函数和变量，在包含 C 语言头文件（假设为 cExample.h）时，需进行下列处理：

```
extern "C"
{
＃i nclude "cExample.h"
}
```

而在 C 语言的头文件中，对其外部函数只能指定为 extern 类型，C 语言中不支持 extern "C"声明，在.c 文件中包含了 extern "C"时会出现编译语法错误。

C++引用 C 函数例子工程中包含的三个文件的源代码如下：

```
/* c 语言头文件：cExample.h */
#ifndef C_EXAMPLE_H
#define C_EXAMPLE_H
extern int add(int x,int y);
#endif


/* c 语言实现文件：cExample.c */
＃i nclude "cExample.h"
int add( int x, int y )
{
return x + y;
}


// c++实现文件，调用 add：cppFile.cpp
extern "C"
{
＃i nclude "cExample.h"
}
int main(int argc, char* argv[])
{
add(2,3);
return 0;
}
```

如果 C++调用一个 C 语言编写的.DLL 时，当包括.DLL 的头文件或声明接口函数时，应加 extern "C" {　}。

（2）在 C 中引用 C++语言中的函数和变量时，C++的头文件需添加 extern "C"，但是在 C 语言中不能直接引用声明了 extern "C"的该头文件，应该仅将 C 文件中将 C++中定义的 extern "C"函数声明为 extern 类型。

C 引用 C++函数例子工程中包含的三个文件的源代码如下：

```
//C++头文件  cppExample.h
#ifndef CPP_EXAMPLE_H
#define CPP_EXAMPLE_H
extern "C" int add( int x, int y );
#endif
```

```
//C++实现文件  cppExample.cpp
＃i nclude "cppExample.h"
int add( int x, int y )
{
return x + y;
}


/* C 实现文件  cFile.c
/*  这样会编译出错：＃i nclude "cExample.h" */
int main( int argc, char* argv[] )
{
add( 2, 3 );
return 0;
}
```

15 题目的解答请参考《C++中 extern “C”含义深层探索》注解：

几道 c 笔试题(含参考答案)

1.

What is displayed when f() is called given the code:

```
class Number {
public:
string type;

Number(): type(“void”) { }
explicit Number(short) : type(“short”) { }
Number(int) : type(“int”) { }
};
void Show(const Number& n) { cout << n.type; }
void f()
{
short s = 42;
Show(s);
}
```

a) void

b) short

c) int

d) None of the above

2. Which is the correct output for the following code

```
double dArray[2] = {4, 8}, *p, *q;
p = &dArray[0];
q = p + 1;
```

cout << q - p << endl;
cout << (int)q - (int)p << endl;
a) 1 and 8
b) 8 and 4
c) 4 and 8
d) 8 and 1

第一个选 C；
虽然传入的是 short 类型，但是 short 类型的构造函数被生命被 explicit，也就是只能显示类型转换，不能使用隐式类型转换。
第二个选 A；
第一个是指针加减，按照的是指向地址类型的加减，只跟类型位置有关，q 和 p 指向的数据类型以实际数据类型来算差一个位置，因此是 1。而第二个加减是实际指针值得加减，在内存中一个 double 类型占据 8 个字节，因此是 8

54.Sony笔试题

1．完成下列程序
*
..
....*..
......*...*...
........*....*....*....
..........*.....*.....*.....*.....
............*......*......*......*......*......
..............*.......*.......*.......*.......*.......
#include
#define N 8
int main()
{
 int i;
 int j;
 int k;

 | |
 | |
 | |
 ----
 return 0;
}
2．完成程序，实现对数组的降序排序
#include

```
void sort( );
int main()
{
 int array[]={45，56，76，234，1，34，23，2，3}；  //数字任//意给出
 sort( );
 return 0;
}
void sort( )
{
 ____________________________________________
 | |
 | |
 |-----------------------------------------------|
}
```

3．费波那其数列，1，1，2，3，5……编写程序求第十项。可以用递归，也可以用其他方法，但要说明你选择的理由。

```
#include
int Pheponatch(int);
int main()
{
 printf("The 10th is %d",Pheponatch(10));
 return 0;
}
int Pheponatch(int N)
{
--------------------------------
| |
| |
--------------------------------
}
```

4．下列程序运行时会崩溃，请找出错误并改正，并且说明原因。

```
#include
#include
typedef struct{
 TNode* left;
 TNode* right;
 int value;
} TNode;
TNode* root=NULL;
void append(int N);
int main()
{
 append(63);
 append(45);
```

```
  append(32);
  append(77);
  append(96);
  append(21);
  append(17); // Again, 数字任意给出
 }
 void append(int N)
 {
  TNode* NewNode=(TNode *)malloc(sizeof(TNode));
  NewNode->value=N;


  if(root==NULL)
  {
  root=NewNode;
  return;
  }
  else
  {
  TNode* temp;
  temp=root;

  while((N>=temp.value && temp.left!=NULL) || (N !=NULL
 ))
  {
  while(N>=temp.value && temp.left!=NULL)
  temp=temp.left;
  while(N          temp=temp.right;
  }
  if(N>=temp.value)
  temp.left=NewNode;
  else
  temp.right=NewNode;
  return;
  }
 }
```

※ 来源: • 哈工大紫丁香 http://bbs.hit.edu.cn • [FROM:219.217.233.47]

---

mengfd (Icebreaker) 于 (Sun Oct 23 14:59:59 2005) 说道:

55 请你分别画出OSI的七层网络结构图和TCP/IP的五层结构图。

应用层：为应用程序提供服务

表示层：处理在两个通信系统中交换信息的表示方式

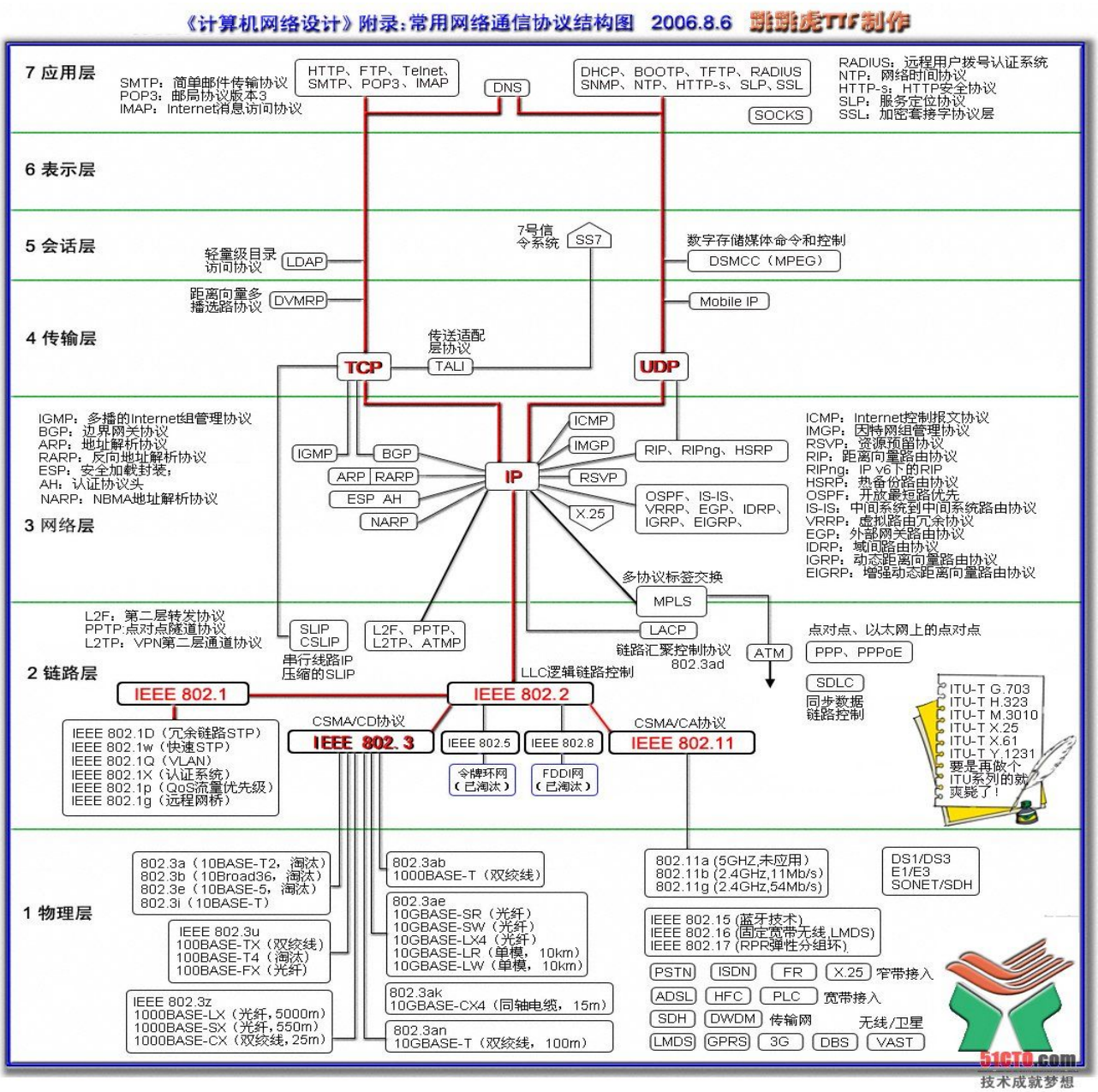
会话层：负责维护两个结点间会话连接的建立、管理和终止，以及数据交换

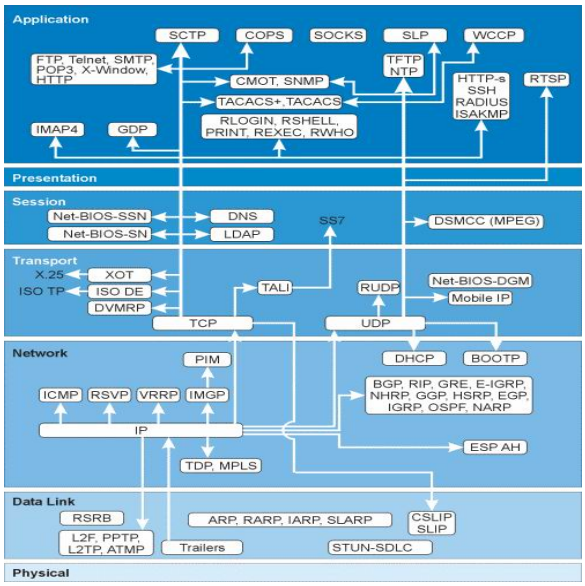
传输层：向用户提供可靠的端到端服务。UDP TCP 协议。

网络层：通过路由选择算法为分组通过通信子网选择最适当的路径，以及实现拥塞控制、网络互联等功能。数据传输单元是分组。IP 地址，路由器，IP 协议。

数据链路层：在物理层提供的服务基础上，数据链路层在通信的实体间建立数据链路连接，传输一帧为单位的数据包（，并采用差错控制与流量控制方法，使有差错的物理线路变成无差错的数据链路。）

物理层：传输比特流。传输单元是比特。调制解调器。





OSI	TCP/IP
Application (X.400, FTAM, VT)	Applications SMTP, TELNET, FTP
Presentation	
Session	
Transport	Transport (Transmission Control Protocol)
Network	Internet (Internet Protocol)
Data Link	Network Interface
Physical	Hardware

56 请你详细地解释一下IP协议的定义，在哪个层上面？主要有什么作用？TCP与UDP呢 ？

网络层。

57.请问交换机和路由器各自的实现原理是什么？分别在哪个层次上面实现的？

交换机：数据链路层。路由器：网络层。

58.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 ？

59.8086 是多少位的系统？在数据总线上是怎么实现的？

8086 微处理器共有 4 个 16 位的段寄存器，在寻址内存单元时，用它们直接或间接地存放段地址。

代码段寄存器 CS：存放当前执行的程序的段地址。

数据段寄存器 DS：存放当前执行的程序所用操作数的段地址。

堆栈段寄存器 SS：存放当前执行的程序所用堆栈的段地址。

附加段寄存器 ES：存放当前执行程序中一个辅助数据段的段地址。

由 cs:ip 构成指令地址，ss:sp 构成堆栈的栈顶地址指针。DS 和 ES 用作数据段和附加段的段地址（段起始地址或段值）

8086／8088 微处理器的存储器管理

1.地址线（码）与寻址范围：N 条地址线　　寻址范围=2N

2.8086 有 20 地址线　　寻址范围为 1MB　由 00000H～FFFFFH

3. 8086 微处理器是一个 16 位结构，用户可用的寄存器均为 16 位：寻址 64KB

4. 8086／8088 采用分段的方法对存储器进行管理。具体做法是：把 1MB 的存储器空间分成若干段，每段容量为 64KB，每段存储器的起始地址必须是一个能被 16 整除的地址码，即在 20 位的二进制地址码中最低 4 位必须是“0”。每个段首地址的高 16 位二进制代码就是该段的段号(称段基地址)或简称段地址，段号保存在段寄存器中。我们可对段寄存器设置不同的值来使微处理器的存储器访问指向不同的段。

5.段内的某个存储单元相对于该段段首地址的差值，称为段内偏移地址(也叫偏移量)用 16 位二进制代码表示。

6.物理地址是由 8086／8088 芯片地址引线送出的 20 位地址码，它用来参加存储器的地址译码，最终读／写所访问的一个特定的存储单元。

7.逻辑地址由某段的段地址和段内偏移地址(也叫偏移量)两部分所组成。写成：

段地址：偏移地址(例如，1234H：0088H)。

8.在硬件上起作用的是物理地址，物理地址＝段基地址×10H 十偏移地址

联想笔试题

1．设计函数 int atoi(char *s)。

2．int i=(j=4,k=8,l=16,m=32); printf(“%d”, i); 输出是多少？

60.解释局部变量、全局变量和静态变量的含义。

4．解释堆和栈的区别。

61.论述含参数的宏与函数的优缺点。

普天 C++笔试题

1．实现双向链表删除一个节点 P，在节点 P 后插入一个节点，写出这两个函数。

2．写一个函数，将其中的\t 都转换成 4 个空格。

61.Windows程序的入口是哪里？写出Windows消息机制的流程。

4．如何定义和实现一个类的成员函数为回调函数？

62.C++里面是不是所有的动作都是main()引起的？如果不是，请举例。

6．C++里面如何声明 const void f(void)函数为 C 程序中的库函数？
7．下列哪两个是等同的
int b;
A const int* a = &b;
B const* int a = &b;
C const int* const a = &b;
D int const* const a = &b;
8．内联函数在编译时是否做参数类型检查？

```
void g(base & b){
  b.play;
}
void main(){
  son s;
  g(s);
  return;
}
```

※ 来源:·哈工大紫丁香 http://bbs.hit.edu.cn·[FROM:219.217.233.47]

---

mengfd (Icebreaker) 于 (Sun Oct 23 15:00:14 2005) 说道:

大唐电信
DTT 笔试题
考试时间一小时，第一部分是填空和选择：
1．数列 6，10，18，32，“？”，问“？”是几？
2．某人出 70 买进一个 x，80 卖出，90 买回，100 卖出，这桩买卖怎么样？
3．月球绕地球一圈，至少要多少时间？
4．7 个人用 7 小时挖了 7 米的沟，以同样的速度在 50 小时挖 50 米的沟要多少人？
5．鱼头长 9，鱼尾等于鱼头加半个鱼身，鱼身等于鱼头加鱼尾，问鱼全长多少？
6．一个小姐买了一块手表，回家发现手表比她家的表慢了两分钟，晚上看新闻的时候又发现她家的表比新闻里的时间慢了两分钟，则 。
A 手表和新闻里的时间一样
B 手表比新闻里的时间慢
C 手表比新闻里的时间快

7．王先生看到一则招聘启事，发现两个公司除了以下条件不同外，其他条件都相同

A 半年年薪 50 万，每半年涨 5 万

B 一年年薪 100 万，每一年涨 20 万

王先生想去一家待遇比较优厚的公司，他会去哪家？

10．问哪个袋子里有金子？

A 袋子上的标签是这样写的：B 袋子上的话是对的，金子在 A 袋子。

B 袋子上的标签是这样写的：A 袋子上的话是错的，金子在 A 袋子里。

11．3 个人住酒店 30 块钱，经理找回 5 块钱，服务生从中藏了 2 块钱，找给每人 1 块钱，

3×（1̃01）+2=29，问这是怎么回事？

12．三篇写作，均为书信形式。

（1）一片中文的祝贺信，祝贺某男当了某公司 xx

（2）两篇英文的，一是说有事不能应邀，派别人去；另一篇是讨债的，7 天不给钱就走人（主要考 business letter 格式）。

大唐面试试题

1．什么是中断？中断发生时 CPU 做什么工作？

2．CPU 在上电后，进入操作系统的 main()之前必须做什么工作？

3．简述 ISO OSI 的物理层 Layer1，链路层 Layer2，网络层 Layer3 的任务。

4．有线电话和无线电话有何区别？无线电话特别需要注意的是什么？

63.软件开发五个主要step是什么？

6．你在开发软件的时候，这 5 个 step 分别占用的时间百分比是多少？

7．makefile 文件的作用是什么？

8．UNIX 显示文件夹中，文件名的命令是什么？能使文件内容显示在屏幕的命令是什么

？

9．（选做）手机用户在从一个基站漫游到另一个基站的过程中，都会发生什么?

※ 来源: • 哈工大紫丁香 http://bbs.hit.edu.cn • [FROM:219.217.233.47]

---

mengfd (Icebreaker) 于 (Sun Oct 23 15:01:22 2005) 说道:

网通笔试题

选择题（每题 5 分，只有一个正确答案）

1．中国 1 号信令协议属于 的协议。

A ccs B cas C ip D atm

2．isdnpri 协议全称是 。

A 综合业务模拟网基速协议

B 综合业务模拟网模拟协议
C 综合业务数字网基率协议
D 综合业务数字网基次协议
3．路由协议中， 协议是用距离作为向量的。
A ospf B bgp C is-is D rip
4．中国智能网中，ssp 与 scp 间最上层的 ss7 协议是 。
A incs B is41b C is41c D inap
5．dtmf 全称是 。
A 双音多频 B 多音双频 C 多音三频 D 三音多频
6．计算机的基本组成部分中，不包含下面设备的是 。
A cpu B 输入设备 C 存储器 D 接口
7．脉冲编码调制的简称是 。
A pcm B pam C (delta)M D atm
8．普通电话线接口专业称呼是 。
A rj11 B rj45 C rs232 D bnc
9．现有的公共数据网都采用 。
A 电路交换技术 B 报文交换技术
C 语音插空 D 分组交换
10．ss7 协议中的制止市忙消息简写为 。
A stb B slb C sub D spb
简答题（每题 10 分）
1．简述普通电话与 IP 电话的区别。
2．简述随路信令与公路信令的根本区别。
3．说明掩码的主要作用。
4．ss7 协议中，有三大要素决定其具体定位，哪三大要素？
5．描述 ss7 的基本通话过程。
6．简述通信网的组成结构。
7．面向连接与面向非连接各有何利弊？
8．写出爱尔兰的基本计算公式。
9．数据网主要有哪些设备？
10．中国一号协议是如何在被叫号码中插入主叫号码的？

东信笔试题目
笔试：30 分钟。
1．压控振荡器的英文缩写。
2．动态随机存储器的英文缩写。
3．选择电阻时要考虑什么？
4．单片机上电后没有运转，首先要检查什么？
5．计算机的基本组成部分及其各自的作用。
6．怎样用 D 触发器、与或非门组成二分频电路？

64.static有什么用途？（请至少说明两种）

答 、1.限制变量的作用域(文件级的)。

2.设置变量的存储域(全局数据区)。

65.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。

66.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。

67.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。

68.什么是平衡二叉树？

答 、左右子树都是平衡二叉树 且左右子树的深度差值的绝对值不大于 1。

69.堆栈溢出一般是由什么原因导致的？

答 、1.没有回收垃圾资源

2.层次太深的递归调用

70.什么函数不能声明为虚函数？

答 、constructor

Deconstructor 可以声明为虚函数。

系统为一个空类创建的成员函数有那些。

71.冒泡排序算法的时间复杂度是什么？

答 、O(n^2)

72.写出float x 与“零值”比较的if语句。

答 、if(x>0.000001&&x<-0.000001)

73.Internet采用哪种网络协议？该协议的主要层次结构？

答 、tcp/ip 应用层/传输层/网络层/数据链路层/物理层

74.Internet物理地址和IP地址转换采用什么协议？

答 、ARP (Address Resolution Protocol)（地址解析协议）

75.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。不过是要和“子网掩码”按位与之后才能区分哪些是网络位哪些是主机位。

76.用户输入M,N值，从 1 至N开始顺序循环数数，每数到M输出该数值，直至全部输出。写出C程序。

答 、循环链表，用取余操作做

77.不能做switch()的参数类型是：

答 、switch 的参数不能为实型。

华为

78.局部变量能否和全局变量重名？

答、能，局部会屏蔽全局。要用全局变量，需要使用"::"

局部变量可以与全局变量同名，在函数内引用这个变量时，会用到同名的局部变量，而不会用到全局变量。对于有些编译器而言，在同一个函数内可以定义多个同名的局部变量，比如在两个循环体内都定义一个同名的局部变量，而那个局部变量的作用域就在那个循环体内

79.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

80.全局变量可不可以定义在可被多个.C文件包含的头文件中？为什么？

答 、可以，在不同的 C 文件中以 static 形式来声明同名全局变量。
可以在不同的 C 文件中声明同名的全局变量，前提是其中只能有一个 C 文件中对此变量赋初值，此时连接不会出错

81.语句for(；1 ；)有什么问题？它是什么意思？

答 、和 while(1)相同。

82.do……while和while……do有什么区别？

答 、前一个循环一遍再判断，后一个判断以后再循环

83.请写出下列代码的输出内容

```
#i nclude
main()
{
int a,b,c,d;
a=10;
b=a++;
c=++a;
d=10*a++;
printf("b，c，d：%d，%d，%d"，b，c，d）;
return 0;
}
```

答 、10，12，120

84.statac 全局变量、局部变量、函数与普通全局变量、局部变量、函数

static 全局变量与普通的全局变量有什么区别？static 局部变量和普通局部变量有什么区别？static 函数与普通函数有什么区别？
答 、全局变量(外部变量)的说明之前再冠以 static 就构成了静态的全局变量。全局变量本身就是静态存储方式，静态全局变量当然也是静态存储方式。这两者在存储方式上并无不同。这两者的区别虽在于非静态全局变量的作用域是整个源程序， 当一个源程序由多个源文件组成时，非静态的全局变量在各个源文件中都是有效的。 而静态全局变量则限制了其作用域， 即只在定义该变量的源文件内有效， 在同一源程序的其它源文件中不能使用它。由于静态全局变量的作用域局限于一个源文件内，只能为该源文件内的函数公用， 因此可以避免在其它源文件中引起错误。
从以上分析可以看出， 把局部变量改变为静态变量后是改变了它的存储方式即改变了它的

生存期。把全局变量改变为静态变量后是改变了它的作用域， 限制了它的使用范围。
static 函数与普通函数作用域不同。仅在本文件。只在当前源文件中使用的函数应该说明为内部函数(static)，内部函数应该在当前源文件中说明和定义。对于可在当前源文件以外使用的函数，应该在一个头文件中说明，要使用这些函数的源文件要包含这个头文件
static 全局变量与普通的全局变量有什么区别：static 全局变量只初使化一次，防止在其他文件单元中被引用;
static 局部变量和普通局部变量有什么区别：static 局部变量只被初始化一次，下一次依据上一次结果值；
static 函数与普通函数有什么区别：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
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。

85.设有以下说明和定义：

typedef union {long i; int k[5]; char c;} DATE;
struct data { int cat; DATE cow; double dog;} too;
DATE max;
则语句 printf("%d",sizeof(struct date)+sizeof(max));的执行结果是？
答 、结果是：___52____。DATE 是一个 union, 变量公用空间. 里面最大的变量类型是 int[5], 占用 20 个字节. 所以它的大小是 20
data 是一个 struct, 每个变量分开占用空间. 依次为 int4 + DATE20 + double8 = 32.
所以结果是 20 + 32 = 52.
当然...在某些 16 位编辑器下, int 可能是 2 字节,那么结果是 int2 + DATE10 + double8 = 20

86.-1,2,7,28,,126 请问 28 和 126 中间那个数是什么？为什么？

答 、应该是 4^3-1=63
规律是 n^3-1(当 n 为偶数 0，2，4)
n^3+1(当 n 为奇数 1，3，5)

87.用两个栈实现一个队列的功能？要求给出算法和思路！

答 、设 2 个栈为 A,B, 一开始均为空.
入队:
将新元素 push 入栈 A;
出队:
(1)判断栈 B 是否为空；
(2)如果不为空，则将栈 A 中所有元素依次 pop 出并 push 到栈 B；
(3)将栈 B 的栈顶元素 pop 出；
这样实现的队列入队和出队的平摊复杂度都还是 O(1), 比上面的几种方法要好。

88.在c语言库函数中将一个字符转换成整型的函数是atool()吗，这个函数的原型是什么？

答 、函数名: atol

功 能: 把字符串转换成长整型数
用 法: long atol(const char *nptr);
程序例:

```
＃include
＃include
int main(void)
{
      long l;
      char *str = "98765432";
      l = atol(lstr);
      printf("string = %s integer = %ld\n", str, l);
      return(0);
}
```

89.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

90.用预处理指令#define 声明一个常数，用以表明 1 年中有多少秒（忽略闰年问题）

```
#define SECONDS_PER_YEAR (60 * 60 * 24 * 365)UL
```

我在这想看到几件事情：
1). #define 语法的基本知识（例如：不能以分号结束，括号的使用，等等）
2). 懂得预处理器将为你计算常数表达式的值，因此，直接写出你是如何计算一年中有多少秒而不是计算出实际的值，是更清晰而没有代价的。
3). 意识到这个表达式将使一个 16 位机的整型数溢出-因此要用到长整型符号 L,告诉编译器这个常数是的长整型数。
4). 如果你在你的表达式中用到 UL（表示无符号长整型），那么你有了一个好的起点。记住，第一印象很重要。

91.写一个“标准”宏MIN，这个宏输入两个参数并返回较小的一个。

```
#define MIN(A,B) ((A) <= (B) (A) : (B))
```

这个测试是为下面的目的而设的：
1). 标识#define 在宏中应用的基本知识。这是很重要的，因为直到嵌入(inline)操作符变为标准 C 的一部分，宏是方便产生嵌入代码的唯一方法，
对于嵌入式系统来说，为了能达到要求的性能，嵌入代码经常是必须的方法。
2). 三重条件操作符的知识。这个操作符存在 C 语言中的原因是它使得编译器能产生比 if-then-else 更优化的代码，了解这个用法是很重要的。

3). 懂得在宏中小心地把参数用括号括起来
4). 我也用这个问题开始讨论宏的副作用，例如：当你写下面的代码时会发生什么事？
least = MIN(*p++, b);

92.预处理器标识#error的目的是什么？

如果你不知道答案，请看参考文献 1。这问题对区分一个正常的伙计和一个书呆子是很有用的。只有书呆子才会读 C 语言课本的附录去找出象这种
问题的答案。当然如果你不是在找一个书呆子，那么应试者最好希望自己不要知道答案。

死循环（Infinite loops）

93.嵌入式系统中经常要用到无限循环，你怎么样用C编写死循环呢？

这个问题用几个解决方案。我首选的方案是：
```
while(1)
{
}
```
一些程序员更喜欢如下方案：
```
for(;;)
{
}
```
这个实现方式让我为难，因为这个语法没有确切表达到底怎么回事。如果一个应试者给出这个作为方案，我将用这个作为一个机会去探究他们这样做的
基本原理。如果他们的基本答案是：“我被教着这样做，但从没有想到过为什么。”这会给我留下一个坏印象。
第三个方案是用 goto
```
Loop:
...
goto Loop;
```
应试者如给出上面的方案，这说明或者他是一个汇编语言程序员（这也许是好事）或者他是一个想进入新领域的 BASIC/FORTRAN 程序员。

数据声明（Data declarations）

94.用变量a给出下面的定义

a) 一个整型数（An integer）
b) 一个指向整型数的指针（A pointer to an integer）
c) 一个指向指针的的指针，它指向的指针是指向一个整型数（A pointer to a pointer to an integer）
d) 一个有 10 个整型数的数组（An array of 10 integers）

e) 一个有 10 个指针的数组，该指针是指向一个整型数的（An array of 10 pointers to integers）
f) 一个指向有 10 个整型数数组的指针（A pointer to an array of 10 integers）
g) 一个指向函数的指针，该函数有一个整型参数并返回一个整型数（A pointer to a function that takes an integer as an argument and returns an integer）
h) 一个有 10 个指针的数组，该指针指向一个函数，该函数有一个整型参数并返回一个整型数（ An array of ten pointers to functions that take an integer
argument and return an integer ）

答案是：
a) int a; // An integer
b) int *a; // A pointer to an integer
c) int **a; // A pointer to a pointer to an integer
d) int a[10]; // An array of 10 integers
e) int *a[10]; // An array of 10 pointers to integers
f) int (*a)[10]; // A pointer to an array of 10 integers
g) int (*a)(int); // A pointer to a function a that takes an integer argument and returns an integer
h) int (*a[10])(int); // An array of 10 pointers to functions that take an integer argument and return an integer

人们经常声称这里有几个问题是那种要翻一下书才能回答的问题，我同意这种说法。当我写这篇文章时，为了确定语法的正确性，我的确查了一下书。
但是当我被面试的时候，我期望被问到这个问题（或者相近的问题）。因为在被面试的这段时间里，我确定我知道这个问题的答案。应试者如果不知道
所有的答案（或至少大部分答案），那么也就没有为这次面试做准备，如果该面试者没有为这次面试做准备，那么他又能为什么出准备呢？

Static

95.关键字static的作用是什么？

这个简单的问题很少有人能回答完全。在 C 语言中，关键字 static 有三个明显的作用：
1). 在函数体，一个被声明为静态的变量在这一函数被调用过程中维持其值不变。
2). 在模块内（但在函数体外），一个被声明为静态的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。它是一个本地的全局变量。
3). 在模块内，一个被声明为静态的函数只可被这一模块内的其它函数调用。那就是，这个函数被限制在声明它的模块的本地范围内使用。
大多数应试者能正确回答第一部分，一部分能正确回答第二部分，同是很少的人能懂得第三部分。这是一个应试者的严重的缺点，因为他显然不懂得本地化数
据和代码范围的好处和重要性。

Const

96.关键字const是什么含意？

我只要一听到被面试者说：“const 意味着常数”，我就知道我正在和一个业余者打交道。去年 Dan Saks 已经在他的文章里完全概括了 const 的所有用法，因此 ESP(译者：Embedded Systems Programming)的每一位读者应该非常熟悉 const 能做什么和不能做什么.如果你从没有读到那篇文章，只要能说出 const 意味着“只读”就可以了。尽管这个答案不是完全的答案，但我接受它作为一个正确的答案。（如果你想知道更详细的答案，仔细读一下 Saks 的文章吧。）如果应试者能正确回答这个问题，我将问他一个附加的问题：下面的声明都是什么意思？

```
const int a;
int const a;
const int *a;
int * const a;
int const * a const;
```

前两个的作用是一样，a 是一个常整型数。第三个意味着 a 是一个指向常整型数的指针（也就是，整型数是不可修改的，但指针可以）。第四个意思 a 是一个指向整型数的常指针（也就是说，指针指向的整型数是可以修改的，但指针是不可修改的）。最后一个意味着 a 是一个指向常整型数的常指针（也就是说，指针指向的整型数是不可修改的，同时指针也是不可修改的）。如果应试者能正确回答这些问题，那么他就给我留下了一个好印象。顺带提一句，也许你可能会问，即使不用关键字 const，也还是能很容易写出功能正确的程序，那么我为什么还要如此看重关键字 const 呢？我也如下的几下理由：

1). 关键字 const 的作用是为给读你代码的人传达非常有用的信息，实际上，声明一个参数为常量是为了告诉了用户这个参数的应用目的。如果你曾花很多时间清理其它人留下的垃圾，你就会很快学会感谢这点多余的信息。（当然，懂得用 const 的程序员很少会留下的垃圾让别人来清理的。）

2). 通过给优化器一些附加的信息，使用关键字 const 也许能产生更紧凑的代码。

3). 合理地使用关键字 const 可以使编译器很自然地保护那些不希望被改变的参数，防止其被无意的代码修改。简而言之，这样可以减少 bug 的出现。

Volatile

97.关键字volatile有什么含意 并给出三个不同的例子。

一个定义为 volatile 的变量是说这变量可能会被意想不到地改变，这样，编译器就不会去假设这个变量的值了。精确地说就是，优化器在用到这个变量时必须每次都小心地重新读取这个变量的值，而不是使用保存在寄存器里的备份。下面是 volatile 变量的几个例子：

1). 并行设备的硬件寄存器（如：状态寄存器）

2). 一个中断服务子程序中会访问到的非自动变量(Non-automatic variables)

3). 多线程应用中被几个任务共享的变量

回答不出这个问题的人是不会被雇佣的。我认为这是区分 C 程序员和嵌入式系统程序员的最基本的问题。嵌入式系统程序员经常同硬件、中断、RTOS 等等打交道，所用这些都要求 volatile 变量。不懂得 volatile 内容将会带来灾难。

假设被面试者正确地回答了这是问题（嗯，怀疑这否会是这样），我将稍微深究一下，看一下这家伙是不是直正懂得 volatile 完全的重要性。
1). 一个参数既可以是 const 还可以是 volatile 吗？解释为什么。
2). 一个指针可以是 volatile 吗？解释为什么。
3). 下面的函数有什么错误：

```
int square(volatile int *ptr)
{
return *ptr * *ptr;
}
```

下面是答案：
1). 是的。一个例子是只读的状态寄存器。它是 volatile 因为它可能被意想不到地改变。它是 const 因为程序不应该试图去修改它。
2). 是的。尽管这并不很常见。一个例子是当一个中服务子程序修该一个指向一个 buffer 的指针时。
3). 这段代码的有个恶作剧。这段代码的目的是用来返指针*ptr 指向值的平方，但是，由于*ptr 指向一个 volatile 型参数，编译器将产生类似下面的代码：

```
int square(volatile int *ptr)
{
int a,b;
a = *ptr;
b = *ptr;
return a * b;
}
```

由于*ptr 的值可能被意想不到地该变，因此 a 和 b 可能是不同的。结果，这段代码可能返不是你所期望的平方值！正确的代码如下：

```
long square(volatile int *ptr)
{
int a;
a = *ptr;
return a * a;
}
```

位操作（Bit manipulation）

98.下面的代码输出是什么，为什么？

```
void foo(void)
{
unsigned int a = 6;
int b = -20;
(a+b > 6) puts("> 6") : puts("<= 6");
}
```

这个问题测试你是否懂得 C 语言中的整数自动转换原则，我发现有些开发者懂得极少这些东西。不管如何，这无符号整型问题的答案是输出是“>6”。原因是当表达式中存在有符号类型和无符号类型时所有的操作数都自动转换为无符号类型。因此-20 变成了一个非常大的正整数，所以该表达式计算出的结果大于 6。这一点对于应当频繁用到无符号数据类型的嵌入式系统来说是丰常重要的。如果你答错了这个问题，你也就到了得不到这份工作的边缘。

99.C语言同意一些令人震惊的结构,下面的结构是合法的吗，如果是它做些什么？

```
int a = 5, b = 7, c;
c = a+++b;
```

这个问题将做为这个测验的一个愉快的结尾。不管你相不相信，上面的例子是完全合乎语法的。问题是编译器如何处理它？水平不高的编译作者实际上会争论这个问题，根据最处理原则，编译器应当能处理尽可能所有合法的用法。因此，上面的代码被处理成：
c = a++ + b;
因此, 这段代码持行后 a = 6, b = 7, c = 12。
如果你知道答案，或猜出正确答案，做得好。如果你不知道答案，我也不把这个当作问题。我发现这个问题的最大好处是:这是一个关于代码编写风格，代码的可读性，代码的可修改性的好的话题

今天早上的面试题 9 道，比较难，

100.线形表a、b为两个有序升序的线形表，编写一程序，使两个有序线形表合并成一个有序升序线形表h；

答案在 请化大学 严锐敏《数据结构第二版》第二章例题，数据结构当中，这个叫做：两路归并排序

```
Linklist *unio(Linklist *p,Linklist *q){
linklist *R,*pa,*qa,*ra;
pa=p;
qa=q;
R=ra=p;
while(pa->next!=NULL&&qa->next!=NULL){
if(pa->data>qa->data){
ra->next=qa;
qa=qa->next;
```

```
}
else{
ra->next=pa;
pa=pa->next;
}
}
if(pa->next!=NULL)
ra->next=pa;
if(qa->next!=NULL)
ra->next==qa;
return R;
}
```

101.用递归算法判断数组a[N]是否为一个递增数组。

递归的方法，记录当前最大的，并且判断当前的是否比这个还大，大则继续，否则返回 false 结束：

```
bool fun( int a[], int n )
{
if( n= =1 )
return true;
if( n= =2 )
return a[n-1] >= a[n-2];
return fun( a,n-1) && ( a[n-1] >= a[n-2] );
}
```

102.编写算法，从 10 亿个浮点数当中，选出其中最大的 10000 个。

用外部排序，在《数据结构》书上有《计算方法导论》在找到第 n 大的数的算法上加工

103.编写一unix程序，防止僵尸进程的出现.

同学的 4 道面试题，应聘的职位是搜索引擎工程师，后两道超级难，（希望大家多给一些算发）

1.给两个数组和他们的大小，还有一动态开辟的内存，求交集，把交集放到动态内存 dongtai，并且返回交集个数

```
long jiaoji(long* a[],long b[],long* alength,long blength,long* dongtai[])
```

2.单连表的建立，把'a'--'z'26 个字母插入到连表中，并且倒叙，还要打印！

方法 1：

```
typedef struct val
{    int date_1;
     struct val *next;
}*p;

void main(void)
{    char c;
```

```
        for(c=122;c>=97;c--)
            { p.date=c;
              p=p->next;
             }

        p.next=NULL;
}
}
```

方法 2：

```
node *p = NULL;
node *q = NULL;

node *head = (node*)malloc(sizeof(node));
head->data = ' ';head->next=NULL;

node *first = (node*)malloc(sizeof(node));
first->data = 'a';first->next=NULL;head->next = first;
p = first;

int longth = 'z' - 'b';
int i=0;
while ( i<=longth )
{
node *temp = (node*)malloc(sizeof(node));
temp->data = 'b'+i;temp->next=NULL;q=temp;

head->next = temp; temp->next=p;p=q;
i++;
}

print(head);
```

104.可怕的题目终于来了

象搜索的输入信息是一个字符串，统计 300 万输入信息中的最热门的前十条，我们每次输入的一个字符串为不超过 255byte,内存使用只有 1G,
请描述思想，写出算发（c 语言），空间和时间复杂度，
7.国内的一些帖吧，如 baidu,有几十万个主题，假设每一个主题都有上亿的跟帖子，怎么样设计这个系统速度最好，请描述思想，写出算发（c 语言），空间和时间复杂度，

```
#include     string.h
main(void)
{     char     *src="hello,world";
```

```
    char     *dest=NULL;
    dest=(char     *)malloc(strlen(src));
    int     len=strlen(str);
    char     *d=dest;
    char     *s=src[len];
    while(len--!=0)
       d++=s--;
    printf("%s",dest);
}
```

找出错误!!

```
#include     "string.h"
#include "stdio.h"
#include "malloc.h"
main(void)
{
char     *src="hello,world";
    char     *dest=NULL;
    dest=(char     *)malloc(sizeof(char)*(strlen(src)+1));
    int     len=strlen(src);
    char     *d=dest;
    char     *s=src+len-1;
    while(len--!=0)
       *d++=*s--;
*d='\0';
    printf("%s",dest);
}
```

105.判断字符串是否为回文

```
    bool IsSymmetry(const char* p)
    {
        assert(p!=NULL);
        const char* q=p;
        int len=0;
        while(*q++!='\0')
        {
            len++;
        }
        bool bSign=true;
        q=p+len-1;
        if (0<len)
        {
            for (int i=0;i<len/2;i++)
            {
                if(*p++!=*q--){ bSign=false;break;};
```

```
            }
        }
        if(bSign==true)
        {
            printf("Yes!\n");
        }
        else
        {
            printf("No!\n");
        }
        return bSign;
    }
```

107.ASDL使用的是什么协议？并进行简单描述？

108.Static 作用是什么

首先 static 的最主要功能是隐藏，其次因为 static 变量存放在静态存储区，所以它具备持久性和默认值 0。

109.什么是预编译,何时需要预编译?

预编译又称为预处理,是做些代码文本的替换工作。处理#开头的指令,比如拷贝#include 包含的文件代码，#define 宏定义的替换,条件编译等，就是为编译做的预备工作的阶段，主要处理#开始的预编译指令，预编译指令指示了在程序正式编译前就由编译器进行的操作，可以放在程序中的任何位置。

c 编译系统在对程序进行通常的编译之前，先进行预处理。c 提供的预处理功能主要有以下三种：1）宏定义　2）文件包含　3）条件编译

1、总是使用不经常改动的大型代码体。
2、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。

110.进程和线程的区别

什么是进程（Process）：普通的解释就是，进程是程序的一次执行，而什么是线程（Thread），线程可以理解为进程中的执行的一段程序片段。在一个多任务环境中下面的概念可以帮助我们理解两者间的差别：

进程间是独立的，这表现在内存空间，上下文环境；线程运行在进程空间内。 一般来讲（不使用特殊技术）进程是无法突破进程边界存取其他进程内的存储空间；而线程由于处于进程空间内，所以同一进程所产生的线程共享同一内存空间。 同一进程中的两段代码不能够同时执行，除非引入线程。线程是属于进程的，当进程退出时该进程所产生的线程都会被强制退出并清除。线程占用的资源要少于进程所占用的资源。 进程和线程都可以有优先级。在

线程系统中进程也是一个线程。可以将进程理解为一个程序的第一个线程。

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)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.

111.插入排序和

插入排序基本思想：（假定从大到小排序）依次从后面拿一个数和前面已经排好序的数进行比较，比较的过程是从已经排好序的数中最后一个数开始比较，如果比这个数，继续往前面比较，直到找到比它大的数，然后就放在它的后面，如果一直没有找到，肯定这个数已经比较到了第一个数，那就放到第一个数的前面。那么一般情况下，对于采用插入排序法去排序的一组数，可以先选 取第一个数做为已经排好序的一组数。然后把第二个放到正确位置。

选择排序(Selection Sort)是一种简单直观的排序算法。它的工作原理如下。首先在未排序序列中找到最小元素,存放到排序序列的起始位置,然后,再从剩余未排序元素中继续寻找最小元素,然后放到排序序列末尾。以此类推,直到所有元素均排序完毕。

112.运算符优先级问题

能正确表示 a 和 b 同时为正或同时为负的逻辑表达式是(D)。
sssA、(a>=0||b>=0)＆＆(a<0||b<0)
B、(a>=0＆＆b>=0)&&(a<0&＆b<0)
C、(a+b>0)＆＆(a+b<=0)
D、a*b>0

以下关于运算符优先顺序的描述中正确的是(C)。
A、关系运算符<算术运算符<赋值运算符<逻辑与运算符
B、逻辑与运算符<关系运算符<算术运算符<赋值运算符
C、赋值运算符<逻辑与运算符<关系运算符<算术运算符
D、算术运算符<关系运算符<赋值运算符<逻辑与运算符

113.字符串倒序

写一个函数将"tom is cat" 倒序打印出来，即 "cat is tom"
//a.ch
#define SPACE ' '
#define ENDL '\0'

char* str = "Tom is cat"; // 字符串
char* p1 = str+strlen(str)-1;
char* p2 = p1; // 开始时，p1,p2 都指向字符串结尾处

```
char t=0; //  临时变量，用来保存被临时替换为 ENDL 的字符
while(str!=p1--)
{
 if(SPACE!=*p1){
    for(p2=p1+1;SPACE!=*p1; p1--, t=*p2, *p2=ENDL);

    // p1+1 指向单词的第一个字母,p2 指向单词的结尾,此时输出这个单词
            printf("%s ",p1+1);
            *p2=t;
            p2=p1;
       }
}
Output:
cat is Tom
```

---

1)写一个递归函数将内存中的字符串翻转"abc"->"cba"
2)写一个函数将"tom is cat" 将内存中的字符串翻转，即 "cat is tomm"

```
#include <stdio.h>
#define SPACE ' '
#define ENDL '\0'
char* s = "The quick brown fox jumps over the lazy dog";
void str_reverse(char* p1,char* p2){
      if(p1==p2)return;
   *p1 = (*p1)+(*p2);
   *p2 = (*p1)-(*p2);
   *p1 = (*p1)-(*p2);
      if(p1==p2-1)return;
      else str_reverse(++p1,--p2);
}
void str_word_reverse(char* str){
      char *q1=str, *q2=str, *t;
      while(*q1==SPACE)q1++;
      if(*q1==ENDL)return; //!
      else q2=q1+1;
      while( (*q2!=SPACE) && (*q2!=ENDL) )q2++;

    t=q2--;
   str_reverse(q1,q2);
      if(*t==ENDL)return;
      else str_word_reverse(t);
}
int
main(int a ,char** b)
```

```
{
   printf("%s\n",s);
   str_reverse(s,s+strlen(s)-1);
   printf("%s\n",s);
   str_word_reverse(s);
   printf("%s\n",s);
        return 0;
}
```

Output:

```
The quick brown fox jumps over the lazy dog
god yzal eht revo spmuj xof nworb kciuq ehT
dog lazy the over jumps fox brown quick The
```

今天同学又问一道题,和上面有些类似，但是要求更严格了一些：

写一个递归函数将内存中的字符串翻转"abc"->"cba",并且函数原型已确定：void reverse(char* p)

其实，要求越多，思路越确定，我的解如下：

```
#include <stdio.h>
#include <string.h>
char* s = "0123456789";
#define ENDL '\0'
void reverse(char* p){
      //这是这种方法的关键，使用 static 为的是能用 str_reverse 的思路，但是不好
      static char* x=0;
      if(x==0)x=p;
      char* q = x+strlen(p)-1;
      if(p==q)return;
      *q=(*p)^(*q);
      *p=(*p)^(*q);
      *q =(*p)^(*q);
      if(q==p+1)return;
      reverse(++p);
}

//这种方法就直观多了，但是当字符串很长的时候就很低效
void reverse2(char* p){
      if(*(p+1)==ENDL)return;
      for(char* o=p+strlen(p)-1,char t=*o;o!=p;o--)
        *o=*(o-1);
      *p=t;
      reverse2(p+1);
}
```

```
int main(int c,char** argv){
        reverse2(s);
        printf("%s\n",s);
        return 0;
}
```

114.交换两个数的宏定义

交换两个参数值的宏定义为：. #define SWAP(a,b) (a)=(a)+(b);(b)=(a)-(b);(a)=(a)-(b);

115.Itearator各指针的区别

游标和指针
我说过游标是指针，但不仅仅是指针。游标和指针很像，功能很像指针，但是实际上，游标是通过重载一元的”*”和”->”来从容器中间接地返回一个值。将这些值存储在容器中并不是一个好主意，因为每当一个新值添加到容器中或者有一个值从容器中删除，这些值就会失效。在某种程度上，游标可以看作是句柄（handle）。通常情况下游标（iterator）的类型可以有所变化，这样容器也会有几种不同方式的转变：

iterator——对于除了 vector 以外的其他任何容器，你可以通过这种游标在一次操作中在容器中朝向前的方向走一步。这意味着对于这种游标你只能使用“++”操作符。而不能使用“--”或“+=”操作符。而对于 vector 这一种容器，你可以使用“+=”、“—”、“++”、“-=”中的任何一种操作符和“<”、“<=”、“>”、“>=”、“==”、“!=”等比较运算符。

116. C++中的class和struct的区别

从语法上，在 C++中（只讨论 C++中）。class 和 struct 做类型定义时只有两点区别：
（一）默认继承权限。如果不明确指定，来自 class 的继承按照 private 继承处理，来自 struct 的继承按照 public 继承处理；
（二）成员的默认访问权限。class 的成员默认是 private 权限，struct 默认是 public 权限。
除了这两点，class 和 struct 基本就是一个东西。语法上没有任何其它区别。
不能因为学过 C 就总觉得连 C++中 struct 和 class 都区别很大，下面列举的说明可能比较无聊，因为 struct 和 class 本来就是基本一样的东西，无需多说。但这些说明可能有助于澄清一些常见的关于 struct 和 class 的错误认识：
（1）都可以有成员函数；包括各类构造函数，析构函数，重载的运算符，友元类，友元结构，友元函数，虚函数，纯虚函数，静态函数；
（2）都可以有一大堆 public/private/protected 修饰符在里边；
（3）虽然这种风格不再被提倡，但语法上二者都可以使用大括号的方式初始化：
A a = {1, 2, 3};不管 A 是个 struct 还是个 class，前提是这个类/结构足够简单，比如所有的成员都是 public 的，所有的成员都是简单类型，没有显式声明的构造函数。
（4）都可以进行复杂的继承甚至多重继承，一个 struct 可以继承自一个 class，反之亦可；一个 struct 可以同时继承 5 个 class 和 5 个 struct，虽然这样做不太好。

（5）如果说 class 的设计需要注意 OO 的原则和风格，那么没任何理由说设计 struct 就不需要注意。
（6）再次说明，以上所有说法都是指在 C++语言中，至于在 C 里的情况，C 里是根本没有“class”，而 C 的 struct 从根本上也只是个包装数据的语法机制。

最后，作为语言的两个关键字，除去定义类型时有上述区别之外，另外还有一点点：“class”这个关键字还用于定义模板参数，就像“typename”。但关键字“struct”不用于定义模板参数。

关于使用大括号初始化
class 和 struct 如果定义了构造函数的话，都不能用大括号进行初始化
如果没有定义构造函数，struct 可以用大括号初始化。
如果没有定义构造函数，且所有成员变量全是 public 的话，可以用大括号初始化。
关于默认访问权限
class 中默认的成员访问权限是 private 的，而 struct 中则是 public 的。
关于继承方式
class 继承默认是 private 继承，而 struct 继承默认是 public 继承。
关于模版

在模版中，类型参数前面可以使用 class 或 typename，如果使用 struct，则含义不同，struct 后面跟的是“non-type template parameter”，而 class 或 typename 后面跟的是类型参数。

class 中有个默认的 this 指针，struct 没有
不同点：构造函数，析构函数 this 指针

117.有关重载函数

返回值类型不同构不成重载
参数参数顺序不同能构成重载
c++函数同名不同返回值不算重载！函数重载是忽略返回值类型的。

成员函数被重载的特征有：
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。
5) 成员函数中 有无 const (函数后面) 也可判断是否重载

118.数据库与T-SQL语言

关系数据库是表的集合，它是由一个或多个关系模式定义。SQL 语言中的数据定义功能包括对数据库、基本表、视图、索引的定义。

119.关系模型的基本概念

关系数据库以关系模型为基础，它有以下三部分组成：

●数据结构——模型所操作的对象、类型的集合

●完整性规则——保证数据有效、正确的约束条件

●数据操作——对模型对象所允许执行的操作方式

关系（Relation）是一个由行和列组成的二维表格，表中的每一行是一条记录（Record），每一列是记录的一个字段（Field）。表中的每一条记录必须是互斥的，字段的值必须具有原子性。

120.SQL语言概述

SQL（结构化查询语言）是关系数据库语言的一种国际标准，它是一种非过程化的语言。通过编写 SQL，我们可以实现对关系数据库的全部操作。

●数据定义语言（DDL）——建立和管理数据库对象

●数据操纵语言（DML）——用来查询与更新数据

●数据控制语言（DCL）——控制数据的安全性

起来是一个很简单的问题，每一个使用过 RDBMS 的人都会有一个概念。
事务处理系统的典型特点是具备 ACID 特征。ACID 指的是 Atomic（原子的）、Consistent（一致的）、Isolated（隔离的）以及 Durable（持续的），它们代表着事务处理应该具备的四个特征：
原子性：组成事务处理的语句形成了一个逻辑单元，不能只执行其中的一部分
一致性：在事务处理执行之前和之后，数据是一致的。
隔离性：一个事务处理对另一个事务处理没有影响。
持续性：当事务处理成功执行到结束的时候，其效果在数据库中被永久纪录下来。

121.C语言中结构化程序设计的三种基本控制结构

顺序结构
选择结构
循环结构

122.CVS是什么

cvs（Concurrent Version System） 是一个版本控制系统。使用它，可以记录下你的源文件的历史。

例如，修改软件时可能会不知不觉混进一些 bug，而且可能过了很久你才会察觉到它们的存

在。有了 cvs，你可以很容易地恢复旧版本，并从中看出到底是哪个修改导致了这个 bug。有时这是很有用的。

CVS 服务器端对每个文件维护着一个修订号,每次对文件的更新，都会使得文件的修订号加 1。在客户端中也对每个文件维护着一个修订号,CVS 通过这两个修订号的关系，来进行 Update,Commit 和发现冲突等操作操作

123.三种基本的数据模型

按照数据结构类型的不同，将数据模型划分为层次模型、网状模型和关系模型。