1.介绍一下STL，详细说明STL如何实现vector。  
STL (标准模版库，Standard Template Library)它由容器算法迭代器组成。  
STL有以下的一些优点：  
可以方便容易地实现搜索数据或对数据排序等一系列的算法；  
调试程序时更加安全和方便；  
即使是人们用STL在UNIX平台下写的代码你也可以很容易地理解（因为STL是跨平台的）。  
vector实质上就是一个动态数组，会根据数据的增加,动态的增加数组空间。

2.如果用VC开发程序，常见这么几个错误，C2001,c2005,c2011,这些错误的原因是什么。  
在学习VC++的过程中，遇到的LNK2001错误的错误消息主要为：  
　 unresolved external symbol “symbol”（不确定的外部“符号”）。  
如果连接程序不能在所有的库和目标文件内找到所引用的函数、变量或标签，将产生此错误消息。  
一般来说，发生错误的原因有两个：一是所引用的函数、变量不存在、拼写不正确或者使用错误；其次可能使用了不同版本的连接库。  
编程中经常能遇到LNK2005错误——重复定义错误，其实LNK2005错误并不是一个很难解决的错误.  
3.继承和委派有什么分别，在决定使用继承或者委派的时候需要考虑什么。  
在OOD,OOP中，组合优于继承.  
当然多态的基础是继承，没有继承多态无从谈起。  
当对象的类型不影响类中函数的行为时，就要使用模板来生成这样一组类。  
当对象的类型影响类中函数的行为时，就要使用继承来得到这样一组类.  
4.指针和引用有什么分别；如果传引用比传指针安全，为什么？如果我使用常量指针难道不行吗？  
(1) 引用在创建的同时必须初始化，即引用到一个有效的对象；而指针在定义的时候不必初始化，可以在定义后面的任何地方重新赋值．  
(2) 不存在ＮＵＬＬ引用，引用必须与合法的存储单元关联;而指针则可以是NULL.  
(3) 引用一旦被初始化为指向一个对象，它就不能被改变为另一个对象的引用；而指针在任何时候都可以改变为指向另一个对象．给引用赋值并不是改变它和原始对象的绑定关系．  
(4) 引用的创建和销毁并不会调用类的拷贝构造函数  
(5) 语言层面，引用的用法和对象一样；在二进制层面，引用一般都是通过指针来实现的，只不过编译器帮我们完成了转换.  
不存在空引用，并且引用一旦被初始化为指向一个对象，它就不能被改变为另一个对象的引用，显得很安全。  
const 指针仍然存在空指针，并且有可能产生野指针.  
总的来说：引用既具有指针的效率，又具有变量使用的方便性和直观性．  
5.参数传递有几种方式；实现多态参数传递采用什么方式，如果没有使用某种方式原因是什么；  
传值，传指针或者引用  
6.结合一个项目说明你怎样应用设计模式的理念。  
设计模式更多考虑是扩展和重用，而这两方面很多情况下，往往会被忽略。  
不过，我不建议滥用设计模式，以为它有可能使得简单问题复杂化.  
7.介绍一下你对设计模式的理解。（这个过程中有很多很细节的问题随机问的）  
设计模式概念是由建筑设计师Christopher Alexander提出:”每一个模式描述了一个在我们周围不断重复发生的问题, 以及该问题的解决方案的核心.这样,你就能一次又一次地使用该方案而不必做重复劳动.”上述定义是对设计模式的广义定义.将其应用到面向对象软件的领域 内,就形成了对设计模式的狭义定义.  
可以简单的认为:设计模式就是解决某个特定的面向对象软件问题的特定方法， 并且已经上升到理论程度。  
框架与设计模式的区别:  
1,设计模式和框架针对的问题域不同.设计模式针对面向对象的问题域;框架针对特定业务的问题域  
2,设计模式比框架更为抽象.设计模式在碰到具体问题后,才能产生代码;框架已经可以用代码表示  
3,设计模式是比框架更小的体系结构元素.框架中可以包括多个设计模式  
设计模式就像武术中基本的招式.将这些招式合理地纵组合起来,就形成套路(框架)，框架是一种半成品.  
8.C++和C定义结构的分别是什么。  
C language 的结构仅仅是数据的结合  
C plus plus的struct 和 class 其实具备几乎一样的功能，只是默认的访问属性不一样而已。  
9.构造函数可否是虚汗数，为什么？析构函数呢，可否是纯虚的呢？  
构造函数不能为虚函数，要构造一个对象，必须清楚地知道要构造什么，否则无法构造一个对象。  
析构函数可以为纯虚函数。  
10.拷贝构造函数相关问题，深拷贝，浅拷贝，临时对象等。  
深拷贝意味着拷贝了资源和指针，而浅拷贝只是拷贝了指针，没有拷贝资源  
这样使得两个指针指向同一份资源，造成对同一份析构两次，程序崩溃。  
临时对象的开销比局部对象小些。  
11.结合1个你认为比较能体现OOP思想的项目，用UML来描述。  
(最好这个项目继承，多态，虚函数都有体现）这个问题大概会占面试时间的一半，并且会问很多问题，一不小心可能会被问住）  
12.基类的有1个虚函数，子类还需要申明为virtual吗？为什么。  
不申明没有关系的。  
不过，我总是喜欢显式申明，使得代码更加清晰。  
13.C也可以通过精心封装某些函数功能实现重用，那C++的类有什么优点吗，难道仅仅是为实现重用。  
并不仅仅是这样的。  
OOD，OOP从根本上改变了程序设计模式和设计思想，具备重大和深远的意义。  
类的三大最基本的特征：封装，继承，多态.  
14.C++特点是什么，如何实现多态？画出基类和子类在内存中的相互关系。  
多态的基础是继承，需要虚函数的支持，简单的多态是很简单的。  
子类继承父类大部分的资源，不能继承的有构造函数，析构函数，拷贝构造函数，operator=函数，友元函数等等  
15.为什么要引入抽象基类和纯虚函数？  
主要目的是为了实现一种接口的效果。  
16.介绍一下模板和容器。如何实现？（也许会让你当场举例实现）  
模板可以说比较古老了，但是当前的泛型编程实质上就是模板编程。  
它体现了一种通用和泛化的思想。  
STL有7种主要容器：vector,list,deque,map,multimap,set,multiset.  
17.你如何理解MVC。简单举例来说明其应用。  
MVC模式是observer 模式的一个特例,典型的有MFC里面的文档视图架构。  
18.多重继承如何消除向上继承的二义性。  
使用虚拟继承即可.

1. 以下三条输出语句分别输出什么？[C易]  
char str1[] = “abc”;  
char str2[] = “abc”;  
const char str3[] = “abc”;  
const char str4[] = “abc”;  
const char\* str5 = “abc”;  
const char\* str6 = “abc”;  
cout << boolalpha << ( str1==str2 ) << endl; // 输出什么？  
cout << boolalpha << ( str3==str4 ) << endl; // 输出什么？  
cout << boolalpha << ( str5==str6 ) << endl; // 输出什么？  
2. 非C++内建型别 A 和 B，在哪几种情况下B能隐式转化为A？[C++中等]  
答：  
a. class B : public A { ……} // B公有继承自A，可以是间接继承的  
b. class B { operator A( ); } // B实现了隐式转化为A的转化  
c. class A { A( const B& ); } // A实现了non-explicit的参数为B（可以有其他带默认值的参数）构造函数  
d. A& operator= ( const A& ); // 赋值操作，虽不是正宗的隐式类型转换，但也可以勉强算一个  
3. 以下代码中的两个sizeof用法有问题吗？[C易]  
void UpperCase( char str[] ) // 将 str 中的小写字母转换成大写字母  
{  
for( size\_t i=0; i  
if( 'a'<=str[i] && str[i]<='z' )  
str[i] -= ('a'-'A' );  
}  
char str[] = "aBcDe";  
cout << "str字符长度为: " << sizeof(str)/sizeof(str[0]) << endl;  
UpperCase( str );  
cout << str << endl;  
1.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。  
2. 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。  
3. 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。  
4. 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 。  
5. 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  
#include   
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。  
6.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;  
7. 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。  
8. 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</vals[9];  
</vals[0];

1. 以下三条输出语句分别输出什么？[C易]  
char str1[] = “abc”;  
char str2[] = “abc”;  
const char str3[] = “abc”;  
const char str4[] = “abc”;  
const char\* str5 = “abc”;  
const char\* str6 = “abc”;  
cout << boolalpha << ( str1==str2 ) << endl; // 输出什么？  
cout << boolalpha << ( str3==str4 ) << endl; // 输出什么？  
cout << boolalpha << ( str5==str6 ) << endl; // 输出什么？  
2. 非C++内建型别 A 和 B，在哪几种情况下B能隐式转化为A？[C++中等]  
答：  
a. class B : public A { ……} // B公有继承自A，可以是间接继承的  
b. class B { operator A( ); } // B实现了隐式转化为A的转化  
c. class A { A( const B& ); } // A实现了non-explicit的参数为B（可以有其他带默认值的参数）构造函数  
d. A& operator= ( const A& ); // 赋值操作，虽不是正宗的隐式类型转换，但也可以勉强算一个  
3. 以下代码中的两个sizeof用法有问题吗？[C易]  
void UpperCase( char str[] ) // 将 str 中的小写字母转换成大写字母  
{  
for( size\_t i=0; i  
if( 'a'<=str[i] && str[i]<='z' )  
str[i] -= ('a'-'A' );  
}  
char str[] = "aBcDe";  
cout << "str字符长度为: " << sizeof(str)/sizeof(str[0]) << endl;  
UpperCase( str );  
cout << str << endl;  
1.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。  
2. 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。  
3. 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。  
4. 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 。  
5. 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  
#include   
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。  
6.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;  
7. 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。  
8. 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</vals[9];  
</vals[0];

1) 什么是预编译，何时需要预编译：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。  
2) char \* const p;  
　　char const \* p  
　　const char \*p  
　　上述三个有什么区别？  
　　char \* const p; //常量指针，p的值不可以修改  
　　char const \* p；//指向常量的指针，指向的常量值不可以改  
　　const char \*p； //和char const \*p  
3)　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;  
　　cout << ( str3 == str4 ) << endl;  
　　cout << ( str5 == str6 ) << endl;  
　　cout << ( str7 == str8 ) << endl;  
　　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。  
4) 以下代码中的两个sizeof用法有问题吗？  
　　void UpperCase( char str[] ) // 将 str 中的小写字母转换成大写字母  
　　{  
　　for( size\_t i=0; i  
　if( 'a'<=str[i] && str[i]<='z' )  
str[i] -= ('a'-'A' );  
　　}  
　　char str[] = "aBcDe";  
　　cout << "str字符长度为: " << sizeof(str)/sizeof(str[0]) << endl;  
　　UpperCase( str );  
　　cout << str << endl;  
答：函数内的sizeof有问题。根据语法，sizeof如用于数组，只能测出静态数组的大小，无法检测动态分配的或外部数组大小。函数外的 str是一个静态定义的数组，因此其大小为6，函数内的str实际只是一个指向字符串的指针，没有任何额外的与数组相关的信息，因此sizeof作用于上 只将其当指针看，一个指针为4个字节，因此返回4。  
5) 一个32位的机器,该机器的指针是多少位？  
指针是多少位只要看地址总线的位数就行了。80386以后的机子都是32的数据总线。所以指针的位数就是4个字节了。  
6)　main()  
　　{  
　　　int a[5]={1,2,3,4,5};  
　　　int \*ptr=(int \*)(&a+1);  
　　　printf("%d,%d",\*(a+1),\*(ptr-1));  
　　}  
　　输出：2,5  
　　\*(a+1）就是a[1]，\*(ptr-1)就是a[4],执行结果是2，5  
　　&a+1不是首地址+1，系统会认为加一个a数组的偏移，是偏移了一个数组的大小（本例是5个int）  
　　int \*ptr=(int \*)(&a+1);  
　　则ptr实际是&(a[5]),也就是a+5  
　　 原因如下：  
　　&a是数组指针，其类型为 int (\*)[5];  
　　而指针加1要根据指针类型加上一定的值，不同类型的指针+1之后增加的大小不同。  
　　a是长度为5的int数组指针，所以要加 5\*sizeof(int)  
　　所以ptr实际是a[5]  
　　但是prt与(&a+1)类型是不一样的(这点很重要)  
　　所以prt-1只会减去sizeof(int\*)  
　　a,&a的地址是一样的，但意思不一样，a是数组首地址，也就是a[0]的地址，&a是对象（数组）首地址，a+1是数组下一元素的地址，即a[1],&a+1是下一个对象的地址，即a[5].  
7)　请问以下代码有什么问题：  
　　int main()  
　　{  
　　　char a;  
　　　char \*str=&a;  
　　　strcpy(str,"hello");  
　　　printf(str);  
　　　return 0;  
　　}  
　　没有为str分配内存空间，将会发生异常。问题出在将一个字符串复制进一个字符变量指针所指地址。虽然可以正确输出结果，但因为越界进行内在读写而导致程序崩溃。  
8)  
　　char\* s="AAA";  
　　printf("%s",s);  
　　s[0]='B';  
　　printf("%s",s);  
　　有什么错？  
　　"AAA"是字符串常量。s是指针，指向这个字符串常量，所以声明s的时候就有问题。  
　　cosnt char\* s="AAA";  
　　然后又因为是常量，所以对是s[0]的赋值操作是不合法的。  
9) 写一个“标准”宏，这个宏输入两个参数并返回较小的一个。  
　　.#define Min(X, Y) ((X)>(Y)?(Y):(X))//结尾没有;  
10) 嵌入式系统中经常要用到无限循环，你怎么用C编写死循环。  
　　while(1){}或者for(;;)  
软件开发网 www.mscto.cn  
11) 关键字static的作用是什么？  
　　定义静态变量  
12) 关键字const有什么含意？  
　　表示常量不可以修改的变量。  
13) 关键字volatile有什么含意？并举出三个不同的例子？  
　　提示编译器对象的值可能在编译器未监测到的情况下改变。  
14) 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。  
15) 有以下表达式：  
　　int a=248; b=4;  
　　int const c=21;  
　　const int \*d=&a;  
　　int \*const e=&b;  
　　int const \*f const =&a;  
　　请问下列表达式哪些会被编译器禁止？为什么？  
　　\*c=32;d=&b;\*d=43;e=34;e=&a;f=0x321f;  
　　\*c 这是个什么东东，禁止  
　　\*d 说了是const， 禁止  
　　e = &a 说了是const 禁止  
　　const \*f const =&a; 禁止  
16) 交换两个变量的值，不使用第三个变量。即a=3,b=5,交换之后a=5,b=3;  
　　有两种解法, 一种用算术算法, 一种用^(异或)  
　　a = a + b;  
　　b = a – b;  
　　a = a – b;  
　　or  
　　a = a^b;// 只能对int,char..  
　　b = a^b;  
　　a = a^b;  
　　or  
　　a ^= b ^= a;  
17) c和c++中的struct有什么不同？  
　　c和c++中struct的主要区别是c中的struct不可以含有成员函数，而c++中的struct可以。c++中struct和class的主要区别在于默认的存取权限不同，struct默认为public，而class默认为private。

18) #include   
　　#include   
　　void getmemory(char \*p)  
　　{　  
　　　p=(char \*) malloc(100);  
　　　strcpy(p,”hello world”);  
　　}　  
　　int main( )  
　　{  
　　　char \*str=NULL;  
　　　getmemory(str);  
　　　printf(“%s/n”,str);  
　　　free(str);  
　　　return 0;  
　　}  
　　程序崩溃，getmemory中的malloc 不能返回动态内存， free（）对str操作很危险  
19) char szstr[10];  
　　strcpy(szstr,”0123456789″);  
　　产生什么结果？为什么？  
　　长度不一样，会造成非法的OS  
20) 列举几种进程的同步机制，并比较其优缺点。  
　　原子操作 、 信号量机制 　、 自旋锁 、　　管程，会合，分布式系统  
21) 进程之间通信的途径  
1. 管道 2. 信号 3. 消息队列 4.共享内存  
22) 进程死锁的原因  
（1）竞争资源  
（2）进程推进顺序不当  
23) 死锁的4个必要条件  
　　互斥、部分分配、不可剥夺、循环  
24) 死锁的处理  
　　鸵鸟策略、预防策略、避免策略、检测与解除死锁  
25) 操作系统中进程调度策略有哪几种？  
　　FCFS(先来先服务)，优先级，时间片轮转，多级反馈  
26) 类的静态成员和非静态成员有何区别？  
　　类的静态成员每个类只有一个，非静态成员每个对象一个  
27) 纯虚函数如何定义？使用时应注意什么？  
　　virtual void f()=0;  
　　是接口，子类必须要实现  
28) 数组和链表的区别  
　　数组：数据顺序存储，固定大小  
　　连表：数据可以随机存储，大小可动态改变  
29) ISO的七层模型是什么？tcp/udp是属于哪一层？tcp/udp有何优缺点？  
　　应用层  
　　表示层  
　　会话层  
　　运输层  
　　网络层  
　　物理链路层  
　　物理层  
　　tcp /udp属于运输层  
　　TCP 服务提供了数据流传输、可靠性、有效流控制、全双工操作和多路复用技术等。  
与 TCP 不同， UDP 并不提供对 IP 协议的可靠机制、流控制以及错误恢复功能等。由于 UDP 比较简单， UDP 头包含很少的字节，比 TCP 负载消耗少。  
　　tcp: 提供稳定的传输服务，有流量控制，缺点是包头大，冗余性不好  
　　udp: 不提供稳定的服务，包头小，开销小  
30) (void \*)ptr 和 (\*(void\*\*))ptr的结果是否相同？  
其中ptr为同一个指针(void \*)ptr 和 (\*(void\*\*))ptr值是相同的  
32) 　　int main()  
　　{  
　　　int x=3;  
　　　printf(“%d”,x);  
　　　return 1;  
　　}  
　　问函数既然不会被其它函数调用，为什么要返回1？  
　　mian中，c标准认为0表示成功，非0表示错误。具体的值是某中具体出错信息  
33) 要对绝对地址0×100000赋值，我们可以用(unsigned int\*)0×100000 = 1234;那么要是想让程序跳转到绝对地址是0×100000去执行，应该怎么做？  
　　\*((void (\*)( ))0×100000 ) ( );  
　　首先要将0×100000强制转换成函数指针,即:  
　　(void (\*)())0×100000  
　　然后再调用它:  
　　\*((void (\*)())0×100000)();  
　　用typedef可以看得更直观些:  
　　typedef void(\*)() voidFuncPtr;  
　　\*((voidFuncPtr)0×100000)();  
34) 已知一个数组table，用一个宏定义，求出数据的元素个数  
　　#define NTBL  
　　#define NTBL (sizeof(table)/sizeof(table[0]))  
35) 线程与进程的区别和联系? 线程是否具有相同的堆栈? dll是否有独立的堆栈?  
　　进程是死的，只是一些资源的集合，真正的程序执行都是线程来完成的，程序启动的时候操作系统就帮你创建了一个主线程。  
　　每个线程有自己的堆栈。DLL中有没有独立的堆栈？  
　　这个问题不好回答，或者说这个问题本身是否有问题。因为DLL中的代码是被某些线程所执行，只有线程拥有堆栈，如果DLL中的代码是EXE中 的线程所调用，那么这个时候是不是说这个DLL没有自己独立的堆栈？如果DLL中的代码是由DLL自己创建的线程所执行，那么是不是说DLL有独立的堆 栈？  
　　以上讲的是堆栈，如果对于堆来说，每个DLL有自己的堆，所以如果是从DLL中动态分配的内存，最好是从DLL中删除，如果你从DLL中分配内存，然后在EXE中，或者另外一个DLL中删除，很有可能导致程序崩溃。  
36) unsigned short A = 10;  
　　printf(“~A = %u\n”, ~A);  
　　char c=128;  
　　printf(“c=%d\n”,c);  
输出多少？并分析过程  
　　第一题，～A ＝0xfffffff5,int值 为－11，但输出的是uint。所以输出4294967285  
　　第二题，c＝0×10,输出的是int，最高位为1，是负数，所以它的值就是0×00的补码就是128，所以输出－128。  
这两道题都是在考察二进制向int或uint转换时的最高位处理。  
37) 分析下面的程序：  
　　void GetMemory(char \*\*p,int num)  
　　{  
　　　\*p=(char \*)malloc(num);  
　　}  
　　int main()  
　　{  
char \*str=NULL;  
　　 GetMemory(&str,100);  
　　　strcpy(str,”hello”);  
　　　free(str);  
　if(str!=NULL)  
　{  
strcpy(str,”world”);  
　}  
printf(“\n str is %s”,str); 软件开发网 www.mscto.com  
getchar();  
　　}  
　　问输出结果是什么？  
　　输出str is world。  
　　free 只是释放的str指向的内存空间,它本身的值还是存在的.所以free之后，有一个好的习惯就是将str=NULL.  
此时str指向空间的内存已被回收,如果输出语句之前还存在分配空间的操作的话,这段存储空间是可能被重新分配给其他变量的,  
尽管这段程序确实是存在大大的问题（上面各位已经说得很清楚了），但是通常会打印出world来。  
这是因为，进程中的内存管理一般不是由操作系统完成的，而是由库函数自己完成的。  
　　当你malloc一块内存的时候，管理库向操作系统申请一块空间（可能会比你申请的大一些），然后在这块空间中记录一些管理信息（一般是在你 申请的内存前面一点），并将可用内存的地址返回。但是释放内存的时候，管理库通常都不会将内存还给操作系统，因此你是可以继续访问这块地址的。  
　　char a[10],strlen(a)为什么等于15？运行的结果  
38) #include “stdio.h”  
　　#include “string.h”  
　　void main()  
　　{  
　　　char aa[10];  
　　　printf(“%d”,strlen(aa));  
　　}  
　　sizeof()和初不初始化，没有关系；  
　　strlen()和初始化有关。  
39) char (\*str)[20];  
　　char \*str[20];  
40) long a=0×801010;  
　　a+5=?  
　　0×801010用二进制表示为：“1000 0000 0001 0000 0001 0000”，十进制的值为8392720，再加上5就是8392725罗  
41) 给定结构  
　　struct A  
　　{  
　char t:4;  
　char k:4;  
　unsigned short i:8;  
　unsigned long m;  
　　};  
　　问sizeof(A) = ?  
　　 给定结构  
　　struct A  
　　{  
　char t:4; 4位  
　char k:4; 4位  
　unsigned short i:8; 8位  
　 unsigned long m; // 偏移2字节保证4字节对齐  
　　}; // 共8字节  
42) 下面的函数实现在一个数上加一个数，有什么错误？请改正。  
　　int add\_n ( int n )  
　　{  
　　static int i = 100;  
　　i += n;  
　　return i;  
　　}  
　　当你第二次调用时得不到正确的结果，难道你写个函数就是为了调用一次？问题就出在 static上？  
43) 分析一下  
　　#include  
　　#include   
　　#include   
　　#include   
　　#include   
　　#include   
　　typedef struct AA  
　　{  
　int b1:5;  
　 int b2:2;  
　　}AA;  
　　void main()  
　　{  
　　　AA aa;  
　　　char cc[100];  
　　　strcpy(cc,”0123456789abcdefghijklmnopqrstuvwxyz”);  
　memcpy(&aa,cc,sizeof(AA));  
　　　cout << aa.b1 <<endl;  
　　　cout << aa.b2 <<endl;  
　　}  
　　答案是 -16和１  
　　首先sizeof(AA)的大小为4,b1和b2分别占5bit和2bit.经过strcpy和memcpy后,aa的4个字节所存放的值 是: 0,1,2,3的ASC码，即00110000,00110001,00110010,00110011所以，最后一步：显示的是这４个字节的前５ 位，和之后的２位分别为：10000,和01，因为int是有正负之分　　  
　　所以：答案是-16和１  
44) 求函数返回值，输入x=9999;  
　　int func （ x ）  
　　{  
　　int countx = 0;  
　　 while ( x )  
　　 {  
　 　countx ++;  
　　x = x&(x-1);  
　　 }  
　　 return countx;  
　　}  
　　结果呢？  
　　知道了这是统计9999的二进制数值中有多少个1的函数，且有9999＝9×1024＋512＋256＋15  
　　9×1024中含有1的个数为2；  
　　512中含有1的个数为1；  
　　256中含有1的个数为1；  
　　15中含有1的个数为4； 软件开发网 www.mscto.com  
　　故共有1的个数为8，结果为8。  
　　1000 - 1 = 0111，正好是原数取反。这就是原理。  
　　用这种方法来求1的个数是很效率很高的。  
　　不必去一个一个地移位。循环次数最少。  
　　int a,b,c 请写函数实现C=a+b ,不可以改变数据类型,如将c改为long int,关键是如何处理溢出问题  
　　bool add (int a, int b,int \*c)  
　　{  
　　　\*c=a+b;  
　　　 return (a>0 && b>0 &&(\*ca || \*c>b)));  
　　}  
45) 分析：  
　　struct bit  
　　{  
　　 int a:3;  
　　int b:2;  
　 int c:3;  
　　};  
　　int main()  
　　{  
　　 bit s;  
　　char \*c=(char\*)&s;  
　 cout<<sizeof(bit)<<endl;  
　　 \*c=0×99;  
　 cout << s.a <<<s.b<<endl<<s.c<<endl;  
　 int a=-1;  
　　printf(“%x”,a);  
　　 return 0;  
　　}  
　　输出为什么是  
　　4  
　　1  
　　-1  
　　-4  
　　ffffffff  
因为0×99在内存中表示为 100 11 001 , a = 001, b = 11, c = 100。当c为有符合数 时, c = 100, 最高1为表示c为负数，负数在计算机用补码表示，所以c = -4;同理 b = -1;当c为有符合数时, c = 100, 即 c = 4,同理 b = 3。 </sizeof(bit)<<endl;  
</endl;  
</endl;

46) 位域 ：  
　　有些信息在存储时，并不需要占用一个完整的字节， 而只需占几个或一个二进制位。例如在存放一个开关量时，只有0和1 两种状态，用一位二进 位即可。为了节省存储空间，并使处理简便，Ｃ语言又提供了一种数据结构，称为“位域”或“位段”。所谓“位域”是把一个字节中的二进位划分为几个不同的区 域，并说明每个区域的位数。每个域有一个域名，允许在程序中按域名进行操作。这样就可以把几个不同的对象用一个字节的二进制位域来表示。一、位域的定义和 位域变量的说明位域定义与结构定义相仿，其形式为：  
struct 位域结构名 { 位域列表 }; 其中位域列表的形式为：类型说明符位域名：位域长度  
　　　例如：  
　　struct bs  
　　{  
　　　int a:8;  
　　　int b:2;  
　　　int c:6;  
　　};  
　　位域变量的说明与结构变量说明的方式相同。可采用先定义后说明，同时定义说明或者直接说明这三种方式。例如：  
　　struct bs  
　　{  
　　　int a:8;  
　　　int b:2;  
　　　int c:6;  
　　}data;  
　　说明data为bs变量，共占两个字节。其中位域a占8位，位域b占2位，位域c占6位。对于位域的定义尚有以下几点说明：  
　　一个位域必须存储在同一个字节中，不能跨两个字节。如一个字节所剩空间不够存放另一位域时，应从下一单元起存放该位域。也可以有意使某位域从下一单元开始。例如：  
　　　　struct bs  
　　{  
　　　unsigned a:4  
　　　unsigned :0  
　　　unsigned b:4  
　　　unsigned c:4  
　　}  
　　在这个位域定义中，a占第一字节的4位，后4位填0表示不使用，b从第二字节开始，占用4位，c占用4位。  
　　由于位域不允许跨两个字节，因此位域的长度不能大于一个字节的长度，也就是说不能超过8位二进位。  
　　位域可以无位域名，这时它只用来作填充或调整位置。无名的位域是不能使用的。例如：  
　　struct k  
　　{  
　　　int a:1  
　　　int :2  
　　　int b:3  
　　　int c:2  
　　};  
　　从以上分析可以看出，位域在本质上就是一种结构类型，不过其成员是按二进位分配的。  
　　位域的使用位域的使用和结构成员的使用相同，其一般形式为：位域变量名?位域名位域允许用各种格式输出。  
　　main()  
　　{  
　　　struct bs  
　　　{  
　　　　unsigned a:1;  
　　　　unsigned b:3;  
　　　　unsigned c:4;  
　　　}  
　　　bit,\*pbit;  
　　　bit.a=1;  
　　　bit.b=7;  
　　　bit.c=15;  
　　　pri  
47) 改错：  
　　#include   
　　int main(void)  
　　{  
　　　 int \*\*p;  
　　int arr[100];  
　　　 p = &arr;  
　　　 return 0;  
　　}  
　　解答：搞错了,是指针类型不同,int \*\*p; //二级指针&arr; //得到的是指向第一维为100的数组的指针  
　　 #include   
　　int main(void)  
　　{  
　　　int \*\*p, \*q;  
　　　int arr[100];  
　　　q = arr;  
　　　p = &q;  
　　　return 0;  
　　}  
48) 下面这个程序执行后会有什么错误或者效果:  
　　#define MAX 255  
　　int main()  
　　{  
　　unsigned char A[MAX],i;//i被定义为unsigned char  
　　for (i=0;i<=MAX;i++)  
　A[i]=i;  
return 0;  
　　}  
　　解答：死循环加数组越界访问（C/C++不进行数组越界检查）MAX=255 数组A的下标范围为:0..MAX-1,这是其一..  
其二.当i循环到255时,循环内执行:A[255]=255;这句本身没有问题..但是返回for (i=0;i<=MAX;i++)语句时,由于unsigned char的取值范围在(0..255),i++以后i又为0了..无限循环下去。  
49) struct name1  
　　{  
　　char str;  
　　short x;  
　　int num;  
　　}  
　　struct name2  
　　{  
　　char str;  
　　int num;  
　　short x;  
　　}  
　　sizeof(struct name1)=8,sizeof(struct name2)=12  
　　在第二个结构中，为保证num按四个字节对齐，char后必须留出3字节的空间；同时为保证整个结构的自然对齐（这里是4字节对齐），在x后还要补齐2个字节，这样就是12字节。  
50) intel：  
　　A.c 和B.c两个c文件中使用了两个相同名字的static变量,编译的时候会不会有问题?这两个static变量会保存到哪里（栈还是堆或者其他的）?  
static的全局变量，表明这个变量仅在本模块中有意义，不会影响其他模块。他们都放在数据区，但是编译器对他们的命名是不同的。如果要使变量在其他模块也有意义的话，需要使用extern关键字。

51) struct s1  
　　{  
　　　int i: 8;  
　　　 int j: 4;  
　　　 int a: 3;  
　　　double b;  
　　};  
　　struct s2  
　　{  
　　　 int i: 8;  
　　　 int j: 4;  
　　　 double b;  
　　　 int a:3;  
　　};  
　　printf("sizeof(s1)= %d\n", sizeof(s1));  
　　printf("sizeof(s2)= %d\n", sizeof(s2));  
　　result: 16, 24  
　　第一个struct s1  
　　{  
　　　 int i: 8;  
　　　 int j: 4;  
　　　 int a: 3;  
　　　 double b;  
　　};  
　　理论上是这样的，首先是i在相对0的位置，占8位一个字节，然后，j就在相对一个字节的位置，由于一个位置的字节数是4位的倍数，因此不用对 齐，就放在那里了，然后是a，要在3位的倍数关系的位置上，因此要移一位，在15位的位置上放下，目前总共是18位，折算过来是2字节2位的样子，由于 double 是8字节的，因此要在相对0要是8个字节的位置上放下，因此从18位开始到8个字节之间的位置被忽略，直接放在8字节的位置了，因此，总共 是16字节。  
　　第二个最后会对照是不是结构体内最大数据的倍数，不是的话，会补成是最大数据的倍数。  
40. 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 </endl;  
</endl;  
</endl;

42. 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 > maxnumber )  
{  
sec\_max = maxnumber ;  
maxnumber = data ;  
}  
else  
{  
if ( data > sec\_max )  
sec\_max = data ;  
}  
}  
return sec\_max ;  
}

43. 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)。

44. 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。  
45. 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;  
}

1. C++的类和C里面的struct有什么区别？  
struct成员默认访问权限为public，而class成员默认访问权限为private  
2. 析构函数和虚函数的用法和作用  
析构函数是在对象生存期结束时自动调用的函数，用来释放在构造函数分配的内存。  
虚函数是指被关键字virtual说明的函数，作用是使用C++语言的多态特性  
3. 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?  
1) 全局变量的作用用这个程序块,而局部变量作用于当前函数  
2) 前者在内存中分配在全局数据区,后者分配在栈区  
3) 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  
4) 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  
4. 有N个大小不等的自然数（1–N），请将它们由小到大排序.要求程序算法：时间复杂度为O(n)，空间复杂度为O(1)。  
void sort(int e[], int n)  
{  
int i;  
int t;  
for (i=1; i  
{  
t = e[e[i]];  
e[e[i]] = e[i];  
e[i] = t;  
}  
}  
5. 堆与栈的去区别  
A. 申请方式不同  
Stack由系统自动分配，而heap需要程序员自己申请，并指明大小。  
B. 申请后系统的响应不同  
Stack：只要栈的剩余空间大于申请空间，系统就为程序提供内存，否则将抛出栈溢出异常  
Heap：当系统收到程序申请时，先遍历操作系统中记录空闲内存地址的链表，寻找第一个大于所申请空间的堆结点，然后将该结点从空间结点链表中删 除，并将该结点的空间分配给程序。另外，大多数系统还会在这块内存空间中的首地址处记录本次分配的大小，以便于delete语句正确释放空间。而且，由于 找到的堆结点的大小不一定正好等于申请的大小，系统会自动将多余的那部分重新放入空闲链表。  
C. 申请大小限制的不同  
Stack：在windows下，栈的大小是2M（也可能是1M它是一个编译时就确定的常数），如果申请的空间超过栈的剩余空间时，将提示overflow。因此，能从栈获得的空间较小。  
Heap：堆是向高地址扩展的数据结构，是不连续的内存区域。这是由于系统是用链表来存储的空闲内存地址的，自然是不连续的，而链表的遍历方向是由低地址向高地址。堆的大小受限于计算机系统中有效的虚拟内存。由此可见，堆获得的空间比较灵活，也比较大。  
D. 申请效率的比较：  
栈由系统自动分配，速度较快。但程序员是无法控制的。  
堆是由new分配的内存，一般速度比较慢，而且容易产生内存碎片,不过用起来最方便。  
另外，在WINDOWS下，最好的方式是用VirtualAlloc分配内存，他不是在堆，也不是在栈是直接在进程的地址空间中保留一快内存，虽然用起来最不方便。但是速度快，也最灵活。  
E. 堆和栈中的存储内容  
栈：在函数调用时，第一个进栈的是主函数中后的下一条指令（函数调用语句的下一条可执行语句）的地址，然后是函数的各个参数，在大多数的C编译器 中，参数是由右往左入栈的，然后是函数中的局部变量。注意静态变量是不入栈的。当本次函数调用结束后，局部变量先出栈，然后是参数，最后栈顶指针指向最开 始存的地址，也就是主函数中的下一条指令，程序由该点继续运行。  
堆：一般是在堆的头部用一个字节存放堆的大小。堆中的具体内容有程序员安排。  
6. 含参数的宏与函数的优缺点  
宏： 优点：在预处理阶段完成，不占用编译时间，同时，省去了函数调用的开销，运行效率高  
缺点：不进行类型检查，多次宏替换会导致代码体积变大，而且由于宏本质上是字符串替换，故可能会由于一些参数的副作用导致得出错误的结果。  
函数： 优点：没有带参数宏可能导致的副作用，进行类型检查，计算的正确性更有保证。  
缺点：函数调用需要参数、返回地址等的入栈、出栈开销，效率没有带参数宏高  
PS：宏与内联函数的区别  
内联函数和宏都是在程序出现的地方展开，内联函数不是通过函数调用实现的，是在调用该函数的程序处将它展开（在编译期间完成的）；宏同样是；  
不同的是：内联函数可以在编译期间完成诸如类型检测，语句是否正确等编译功能；宏就不具有这样的功能，而且宏展开的时间和内联函数也是不同的（在运行期间展开）  
7. Windows程序的入口是哪里？写出Windows消息机制的流程  
Windows程序的入口是WinMain()函数。  
Windows应用程序消息处理机制：  
A. 操作系统接收应用程序的窗口消息，将消息投递到该应用程序的消息队列中  
B. 应用程序在消息循环中调用GetMessage函数从消息队列中取出一条一条的消息，取出消息后，应用程序可以对消息进行一些预处理。  
C. 应用程序调用DispatchMessage，将消息回传给操作系统。  
D. 系统利用WNDCLASS结构体的lpfnWndProc成员保存的窗口过程函数的指针调用窗口过程，对消息进行处理。  
8. 如何定义和实现一个类的成员函数为回调函数  
A.什么是回调函数?  
简而言之，回调函数就是被调用者回头调用调用者的函数。  
使用回调函数实际上就是在调用某个函数（通常是API函数）时，将自己的一个函数（这个函数为回调函数）的地址作为参数传递给那个被调用函数。而该被调用函数在需要的时候，利用传递的地址调用回调函数。  
回调函数，就是由你自己写的，你需要调用另外一个函数，而这个函数的其中一个参数，就是你的这个回调函数名。这样，系统在必要的时候，就会调用你写的回调函数，这样你就可以在回调函数里完成你要做的事。  
B.如何定义和实现一个类的成员函数为回调函数  
要定义和实现一个类的成员函数为回调函数需要做三件事：  
a．声明；  
b．定义；  
c．设置触发条件，就是在你的函数中把你的回调函数名作为一个参数，以便系统调用  
如：  
一、声明回调函数类型  
typedef void (\*FunPtr)(void);  
二、定义回调函数  
class A  
{  
public:  
A();  
static void callBackFun(void) //回调函数，必须声明为static  
{  
cout<<"callBackFun"<<endl;  
}  
virtual ~A();  
};  
三、设置触发条件  
void Funtype(FunPtr p)  
{  
p();  
}  
void main(void)  
{  
Funtype(A::callBackFun);  
}  
C. 回调函数与API函数  
回调和API非常接近，他们的共性都是跨层调用的函数。但区别是API是低层提供给高层的调用，一般这个函数对高层都是已知的；而回调正好相反， 他是高层提供给底层的调用，对于低层他是未知的，必须由高层进行安装，这个安装函数其实就是一个低层提供的API，安装后低层不知道这个回调的名字，但它 通过一个函数指针来保存这个回调函数，在需要调用时，只需引用这个函数指针和相关的参数指针。  
其实：回调就是该函数写在高层，低层通过一个函数指针保存这个函数，在某个事件的触发下，低层通过该函数指针调用高层那个函数。</endl;

1、局部变量能否和全局变量重名？  
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。  
2、如何引用一个已经定义过的全局变量？  
extern  
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。  
3、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？  
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。  
4、语句for( ；1 ；)有什么问题？它是什么意思？  
无限循环，和while(1)相同。  
5、do……while和while……do有什么区别？  
前一个循环一遍再判断，后一个判断以后再循环。  
6、请写出下列代码的输出内容  
#include  
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  
7、请找出下面代码中的所以错误  
说明：以下代码是把一个字符串倒序，如“abcd”倒序后变为“dcba”  
#include “string.h”  
main()  
{  
char\*src=”hello,world”;  
char\* dest=NULL;  
int len=strlen(src);  
dest=(char\*)malloc(len);  
char\* d=dest;  
char\* s=src[len];  
while(len–!=0)  
d++=s–;  
printf(“%s”,dest);  
return 0;  
}  
答：  
方法1：  
int main()  
{  
char\* src = “hello,world”;  
int len = strlen(src);  
char\* dest = (char\*)malloc(len+1);//要为\0分配一个空间  
char\* d = dest;  
char\* s = &src[len-1];//指向最后一个字符  
while( len– != 0 )  
\*d++=\*s–;  
\*d = 0;//尾部要加\0  
printf(“%s\n”,dest);  
free(dest);// 使用完，应当释放空间，以免造成内存汇泄露  
return 0;  
}  
方法2：  
#include   
#include   
main()  
{  
char str[]=”hello,world”;  
int len=strlen(str);  
char t;  
for(int i=0; i  
{  
t=str[i];  
str[i]=str[len-i-1]; str[len-i-1]=t;  
}  
printf("%s",str);  
return 0;  
}

8、-1,2,7,28,,126请问28和126中间那个数是什么？为什么？  
答案应该是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)  
答案：63  
9、用两个栈实现一个队列的功能？要求给出算法和思路！  
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。  
10、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？  
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);  
}  
11、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?  
c用宏定义，c++用inline  
12、直接链接两个信令点的一组链路称作什么?  
PPP点到点连接  
13、接入网用的是什么接口?  
14、voip都用了那些协议?  
15、软件测试都有那些种类?  
黑盒：针对系统功能的测试 白合：测试函数功能，各函数接口  
16、确定模块的功能和模块的接口是在软件设计的那个队段完成的?  
概要设计阶段  
17、enum string  
{  
x1,  
x2,  
x3=10,  
x4,  
x5,  
}x;  
问x= 0×801005，0x8010f4;  
18、unsigned char \*p1;  
unsigned long \*p2;  
p1=(unsigned char \*)0×801000;  
p2=(unsigned long \*)0×810000;  
请问p1+5= 0×801005;  
p2+5= 0×801014;  
19. 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。  
20. 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。  
21. New delete 与malloc free 的联系与区别?  
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.  
22. #define DOUBLE(x) x+x ，i = 5\*DOUBLE(5)； i 是多少？  
答案：i 为30。