1.    指出以下变量数据存储位置

全局变量int(\*g\_pFun)(int);g\_pFun=myFunction;g\_pFun存储的位置(A ) 为全局的函数指针

指向空间的位置( B) 所有函数代码位于TEXT段

函数内部变量 static int nCount;       ( A) 静态变量总是在DATA段或BSS段中

函数内部变量 char p[]=”AAA”;  p 指向空间的位置( C) 局域变量的静态数组,空间在Stack中

函数内部变量 char \*p=”AAA”;  p 指向空间的位置( E) ,”AAA”为一字符常量空间,不同编译器有不同处理方法,大部分保存在TEXT(代码段中),也有编译的rodata段中

函数内部变量 char \*p=new char; p的位置(C ) 指向空间的位置(D ) 所有malloc空间来自于heap(堆)

A.    数据段

B.    代码段

C.    堆栈

D.    堆

E.    不一定, 视情况而定

以上知识参见C语言变量的作用域相关课件

2.    以下程序的输出结果为 ( )

#include <iostream>

main( )

{

using namespace std;

int num[5]={1,2,3,4,5};

cout <<\*((int \*)(&num+1)-1) <<endl;

}

A. 1        B.2        C. 3        D. 4        E. 5       F. 0        G. 未初始化内存,无法确定

在C语言中,一维数组名表示数组的首地址,而且是一个指针.如上例num,

对&num,表示指针的指针.意味着这里强制转换为二维数组指针.

这样 &num+1 等同于 num[5][1],为代码空间. (&num+1)-1表示 num[4][0].即num[4].所以这里答案是E.

扩展题目:

\*((int \*)(num+1)-1)   的值是多少?

Num是首指针,num+1是第二个元素指针,-1后又变成首指针.所以这里是答案是num[0]即,A.1

3.    以下哪些是程序间可靠的通讯方式( C ),哪些可以用于跨主机通讯( C,D ,F).Windows命名管道跨机器也可跨机器.

A. 信号         B. 管道               C. TCP          D. UDP         E. PIPE         F,.串口I/O

4. class a

{

public:

virtual  void  funa( );

virtual  void  funb( );

void  fun( );

static  void  fund( );

static  int  si;

private:

int  i;

char  c;

};

问: 在32位编译器默认情况下,sizeof(a)等于( )字节?

A. 28             B. 25      C.24          D. 20           E. 16        F.12             G. 8

答案在VC++下是 12. 这里需要考虑三个问题,一是虚函数表vtable的入口表地址,二是字节对齐.三 ,静态成员是所有对象共享,不计入sizeof空间.

在大部分C++的实现中,**带有虚函数的类**的前4个BYTE是虚函数vtable表的这个类入口地址.所以sizeof必须要加入这个4个byte的长度,除此外,类的sizoef()为所有数据成员总的sizeof之和,这里是int i,和char c.其中char c被字节对齐为4.这样总长度为

Sizeof(a) = sizeof(vtable)+size(int)+sizeof(char + pad) = 12;

5. 32位Windows 系统或Linux系统下

struct

{

char  a;

char  b;

char  c;

}A;

struct

{

short  a;

short  b;

short  c;

}B;

struct

{

short  a;

long  b;

char  c;

}C;

printf(“%d,%d,%d\n”,sizeof(A),sizeof(B),sizeof(C)); 的执行结果为: ( )

A. 3,6,7         B. 3,6,8         C. 4,8,12              D. 3,6,12      E. 4,6,7         F. 4,8,9

C语法的字节对齐规则有两种情况要字节对齐, 在VC++,gcc测试都是如此

1)    对同一个数据类型（short,int,long）发生了跨段分布,(在32CPU里,即一个数据类型分布在两个段中)才会发生字节对齐.

2)    数据类型的首部和尾部必须有其一是与4对齐.而且违反上一规则.

l  Sizeof(A),sizeof(B)虽然总字节数不能被4整除.但刚好所有数据平均分布在以4为单位的各个段中.所以无需字节对齐,所以结果是 3和6

l  struct {char a;char b;char c;char d;char e;}F; 的sizoef(F)是等于5.

l  用以下实例更加清楚

struct {

char a[20];

short b;

}A;

struct {

char a[21];

short b;

}B;

Sizeof(A)=22,sizoef(B)=24.因为前者没有发生跨段分布.后者,如果不字节对齐.a[21]占用最后一个段的首地址,b无法作到与首部与尾部与4对齐,只能在a[21]与b之间加入一个byte,使用b的尾部与4对齐.

l  C就是比较好理解.要补多个成12

6.    依据程序,以下选择中那个是对的? (  )

class A

{

int  m\_nA;

};

class B

{

int   m\_nB;

};

class C:public A,public B

{

int  m\_nC;

};

void f (void)

{

C\* pC=new C;

B\* pB=dynamic\_cast<B\*>(pC);

A\* pA=dynamic\_cast<A\*>(pC);

}

A. pC= =pB,(int)pC= =(int)B                                  B. pC= =pB,(int)pC!=(int)pB

C. pC!=pB,(int)pC= =(int)pB                                  D. pC!=pB,(int)pC!=(int)pB

这里主要考多态..将程序变为如下比较易懂

#include <stdio.h>

class A

{

public:

int  m\_nA;

};

class B

{

public:

int   m\_nB;

};

class C:public A,public B

{

public:

int  m\_nC;

};

void f (void)

{

C\* pC=new C;

B\* pB=dynamic\_cast<B\*>(pC);

A\* pA=dynamic\_cast<A\*>(pC);

}

void f1 (void)

{

C\* pC=new C;

pC->m\_nA = 1;

pC->m\_nB = 2;

pC->m\_nC = 3;

B\* pB=dynamic\_cast<B\*>(pC);

A\* pA=dynamic\_cast<A\*>(pC);

printf(“A=%x,B=%x,C=%x,iA=%d,iB=%d,iC=%d\n”,pA,pB,pC,(int)pA,(int)pB,(int)pC);

}

void test1();

int main()

{

// test1();

f1();

getchar();

return 0;

}

以上程序输出:

**A=4318d0,B=4318d4,C=4318d0,iA=4397264,iB=4397268,iC=4397264**

即C从,A,B继承下来,由下图可以知道 pA=pC.而pB强制转换后,只能取到C中B的部分.所以pB在pC向后偏移4个BYTE,(即m\_nA)的空间

7,请写出能匹配”[10]:dddddd ”和”[9]:abcdegf ”,不匹配”[a]:xfdf ”的正则表达式\_\_\_\_\_\_\_\_,linux下支持正则的命令有:\_\_\_find,grep\_\_\_\_\_\_\_\_\_

8.如下程序:

int i=1,k=0;

long \*pl=NULL;

char \*pc=NULL;

if(k++&&i++)

k++, pl++, pc++;

if(i++||k++)

i++, pl++, pc++;

printf(“i=%d,k=%d,pl=%ld,pc=%ld\n”,i,k,(long)pl,(long)pc);

打印结果为\_\_i=3,k=1,pl=4,pc=1\_\_\_\_\_\_\_\_

主要测试逻辑表达式的短路操作.

&&操作中,前一个表达式为0,后一表达式不执行

||操作中, 前一个表达式为1,后一表达式不执行

9. 以下程序的输出为\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include<iostream>

using std::cout;

class A

{

public:

void f(void){

cout<< ”A::f” <<’ ‘;

}

virtual void g(void)

{

cout <<”A::g” << ‘ ‘;

}

};

class B : public A

{

public:

void f(void)

{

cout << “B :: f “ << ‘ ‘;

}

void g(void)

{

cout << “B:: g “ << ‘ ‘;

}

};

int main()

{

A\*  pA =new B;

pA->f();

pA->g();

B\* pB = (B\*)pA;

pB->f();

pB->g();

}

A::f B:: g  B :: f  B:: g

多态中虚函数调用.

f()为非虚函数,这样强制转换后,执行本类的同名函数.

G()为虚函数,指针总是执行虚函数,这就是多态..

10.下列代码的作用是删除list lTest 中值为6的元素:

list<int> :: iterator Index = ITest .begin();

for( ;  Index != ITest .end();  ++ Index)

{

if((\*Index) = = 6)

{

ITest .erase(Index);

}

}

请问有什么错误\_\_\_\_ Index = ITest .erase(Index);\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_,

STL的游标处理,erase已经将Index破坏掉,需要用新的Index,否则下一循环的++Index被破坏掉

请写出正确的代码,或者在原代码上修正.

11.找错误\_以下程序:

char\* ptr = malloc(100);

if(!ptr)

{

…

}

…

//ptr 指向的空间不够需要重新分配

ptr = realloc(ptr,200);

if(!ptr)

{

…

}

…

请问有什么错误\_\_\_if(ptr ==NULL)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_,请写出正确的代码,或者在原代码上修正.

12.以下为window NT 下32 位C++程序,请填写如下值

class myclass

{

int a ;

int b;

};

char \*p = “hello”;

char str[] = “world”;

myclass classes[2];

void \*p2= malloc(100);

sizeof(p)=\_4\_\_

sizeof(str)=\_6\_

sizeof(classes)=\_16\_\_

sizeof(p2)=\_4\_\_\_

13.直接在以下程序中的错误的行数后的填空栏中打叉

程序1:

int main(void)

{

int i=10;\_\_\_\_\_

int \*const j=&i;\_\_\_\_\_\_\_

(\*j)++;\_\_\_\_

j++;\_\_\_\*\_\_\_\_\_

}

程序2:

int main(void)

{

int i=20;\_\_\_\_\_

const int \*j=&i;\_\_\_\_\_\_\_\_\_

\*j++;\_\_\_\_\_\_

(\*j)++;\_\_\_\_\*\_\_\_\_

}

主要考const 出现在\*前后不同含意,const 在\*后表示指针本身不能改,const 在\*前面指针内容不能改,程序1中j不能修改指针,所以j++是错,程序2,j不能改改内容,所以

14.用C/C++代码实现以下要求:从1-100中挑选出10个不同的数字,请把可能的所有组合打印出来.

15.有一个非常大的全局数组int a[],长度n超过2的24次方,写一个针对该数组的查找算法unsigned search(int value)(返回值下标),插入算法insert(int value,unsigned index).再次注意该数组的长度很长.

题目不太清,可能可以把数值本身作下标.并且按顺序排序.

16.有两个单向链表,表头pHeader1,pHeader2,请写一个函数判断这两个链表是否有交叉.如果有交叉,给出交叉点.程序不能改变链表的内容,可以使用额外的空间,时间复杂度尽量小,最好给出两种解.(双重循环的解由于时间复杂度高，不算正解).

1.移动链表指针,如果最终

17.编写程序,将一棵树从根节点到叶子的所有最长路径都打印出来.比如一棵树从跟到最末端的叶子最远要经

过4个节点,那么就把到所有要经过4个节点才能到达的叶子的搜索路径(所有途径节点)都分别打印出来.

18.请分别对一个链表和一个数组进行排序,并指出你排序的算法名称以及为何选择该算法

数组可用交换法排序

19.有单向链表,其中节点结构为Node{int value;Node \*pNext};只知道指向某个节点的指针pCurrent;并且知道该节点不是尾节点,有什么办法把他删除吗?要求不断链.

从链表头开始,找到pCurrent上一个结点pPrev,然后 pPrev->pNext = pCurrent->pNext;

20.问题A:用什么方法避免c/c++编程中的头文件重复包含?问题B:假设解决了重复包含问题,但是又需要在两个不同的头文件中引用各申明的类,应该如何处理?具体代码如下:

在头文件Man.h中

….

Class Cman

{

….

CFace m\_face;

};

….

在头文件Face.h中

…

Class CFace

{

…

Cman \*m\_owner;

};

….

这样类CMan.CFace就相互引用了,该如何处理呢?

1.#ifndef ….

#define …..

2.类的前向声明

21.多线程和单线程各自分别在什么时候效率更高?

多线程在并发,并且各线程无需访问共享数据情况详细最高

如果多线程过于频繁切换,或共享数据很多情况下,使用单线程较好

22.在程序设计中,对公共资源(比如缓冲区等)的操作和访问经常需要使用锁来进行保护,但在大并发系统中过多的锁会导致效率很低,通常有那些方法可以尽量避免或减少锁的使用?

减少锁的粒度,每次尽可能减少锁范围

采用队列处理,这样无需使用锁.

23.请详细阐述如何在release版本(windows程序或linux程序都可以)中,查找段错误问题.

可以用编译器生成map文件来定位源码.通过地址反查源码

24.假设你编译链接release版本后得到一个可执行程序(由多个cpp文件和H文件编译),结果可执行程序文件非常大,你如何找到造成文件太大的可能原因,可能的原因是什么?

使用一个已经初始化的巨大的全局数组

25.在编写C++赋值运算符时有哪些要注意的地方?

返回值,参数最好用引用

减少友元函数使用,移植有问题.

26.假设你是参与设计嫦娥卫星的嵌入式单板软件工程师,其中有一个快速搜索可能要用到哈希变或者平衡二叉树,要求不管什么条件下,单板必须在指定的短时间内有输出,你会采取那种算法?为什么用这种算法,为什么不用另一种算法?

HASH.HASH访问速度较快.

27.strcpy()容易引起缓冲区溢出问题,请问有什么函数可以替代以减少风险,为什么?

strncpy

28.请指出spinlock,mutex,semaphore,critical section的作用与区别,都在哪些场合使用.

spin\_lock Linux 内核自旋锁. Mutex Windows 互质量, semaphore 　POSIX　,critical section　Windows

29.在哪些方法使阻塞模式的recv函数在没有收到数据的情况下返回(不能将socket修改为非阻塞模式)请描述得详细点.

使用select

30.有3个红色球,2个白色球,1个绿色球.取出两个不同颜色的球就能变成两个第三种颜色的球(比如:取出1红球,1白球,就能变成2个绿球).问,最少几次变化能将所有球都变成同一颜色,说明步骤和原因?

31.单向链表的反转是一个经常被问到的一个面试题，也是一个非常基础的问题。比如一个链表是这样的： 1->2->3->4->5 通过反转后成为5->4->3->2->1。

最容易想到的方法遍历一遍链表，利用一个辅助指针，存储遍历过程中当前指针指向的下一个元素，然后将当前节点元素的指针反转后，利用已经存储的指针往后面继续遍历。源代码如下：

1. struct linka {
2. int data;
3. linka\* next;
4. };
5. void reverse(linka\*& head) {
6. if(head ==NULL)
7. return;
8. linka \*pre, \*cur, \*ne;
9. pre=head;
10. cur=head->next;
11. while(cur)
12. {
13. ne = cur->next;
14. cur->next = pre;
15. pre = cur;
16. cur = ne;
17. }
18. head->next = NULL;
19. head = pre;
20. }

还有一种利用递归的方法。这种方法的基本思想是在反转当前节点之前先调用递归函数反转后续节点。源代码如下。不过这个方法有一个缺点，就是在反转后的最后一个结点会形成一个环，所以必须将函数的返回的节点的next域置为NULL。因为要改变head指针，所以我用了引用。算法的源代码如下：

1. linka\* reverse(linka\* p,linka\*& head)
2. {
3. if(p == NULL || p->next == NULL)
4. {
5. head=p;
6. return p;
7. }
8. else
9. {
10. linka\* tmp = reverse(p->next,head);
11. tmp->next = p;
12. return p;
13. }
14. }

**32.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 ;  
}

**33.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。

**34. 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。

**45. 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。

**36. 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 。

**37. 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。

**38. “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;

**39. “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。

**40. 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://blog.csdn.net/wfwd/archive/2006/05/30/763551.aspx

**41. 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。

**42. 下面关于“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=0×4241; /\*联合成员赋值\*/  
printf(“%c%c\n”, number.half.first, mumber.half.second);  
number.half.first=’a'; /\*联合中结构成员赋值\*/  
number.half.second=’b';  
printf(“%x\n”, number.i);  
getch();  
}  
答案： AB (0×41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）

**43. 已知strcpy的函数原型：char \*strcpy(char \*strDest, const char \*strSrc)其中strDest 是目的字符串，strSrc 是源字符串。不调用C++/C 的字符串库函数，请编写函数 strcpy。**

答案：  
char \*strcpy(char \*strDest, const char \*strSrc)  
{  
if ( strDest == NULL || strSrc == NULL)  
return NULL ;  
if ( strDest == strSrc)  
return strDest ;  
char \*tempptr = strDest ;  
while( (\*strDest++ = \*strSrc++) != ‘\0’)  
return tempptr ;  
}

**44. .h头文件中的ifndef/define/endif 的作用？**  
答：防止该头文件被重复引用。

**45. ＃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。

**46.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”？**

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);

实际上，在连接阶段，连接器会从模块A生成的目标文件moduleA.obj中寻找\_foo\_int\_int这样的符号！

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” \*/  
extern int add( int x, int y );  
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：**

**47. 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://blog.csdn.net/wfwd/archive/2006/05/30/763753.aspx

http://blog.csdn.net/wfwd/archive/2006/05/30/763760.aspx

**48.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。

**49. 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：是指子类重新定义复类虚函数的方法。

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）。

**50. 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。

**51. 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.

**52. 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。

**53. 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。

**54. 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。

**55. 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。

**56.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。

**57.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。

**58. 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。

**59. 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。

**60.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)

**61.请说出const与#define 相比，有何优点？**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。  
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。

**62.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  
}

**63.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）

**64. 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。

**65. 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.

**66. 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

**67.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  
}

**68. 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 ;  
}

**69. 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

**70. 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 ;  
}

**71. 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)。

**72. 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/wfwd/archive/2006/05/30/763797.aspx

**73. 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;  
}

74.不使用库函数，编写函数int strcmp(char  \*source, char \*dest)相等返回0，不等返回-1

int StrCmp(char  \*source, char \*dest)

{

assert(source !=NULL) ;

assert(dest!=NULL) ;

while(\*source= =\*dest&&\*source&&\*dest)

{

source++;

dest++;

}

return (\*source!=\*dest)?-1:0;

}

75.写一个函数，实现将一个字符串中的’\t’替换成四个’\*’， ’\t’个数不定。如char \*p=”ht\thdsf\t\ttt\tfds dfsw\t ew\t”,替换后p=”ht\*\*\*\*hdsf\*\*\*\*\*\*\*\*tt\*\*\*\*fds dfsw\*\*\*\* ew\*\*\*\*”。

char \*Replace(char \*Sorce)

{

char \*pTemp=Sorce;

int iCount=0;

int iSizeOfSorce=0;

while(\*pTemp!=’\0′)

{

if(‘\t’==\*pTemp)

iCount++;

pTemp++;

}

iSizeOfSorce=pTemp-Sorce;

char \*pNewStr=new char[iSizeOfSorce+3\*iCount\*sizeof(char)+1];

char \*pTempNewStr=pNewStr;

pTemp=Sorce;

while(\*pTemp!=’\0′)

{

if(‘\t’==\*pTemp)

{

for(int iLoop=0; iLoop<4; iLoop++)

{

\*pTempNewStr=’\*';

pTempNewStr++;

}

pTemp++;

}

else

{

\*pTempNewStr=\*pTemp;

pTemp++;

pTempNewStr++;

}

}

\*pTempNewStr=’\0′;

return pNewStr;

}

**76．写一函数实现将一个字符串中的数字字符全部去掉。**

**void RemoveNum(char strSrc[])**

**{**

**char \*p=strSrc;**

**char \*q;**

**while(\*p!=’\0′)**

**{**

**if(\*p>=’0′&&\*p<=’9′)**

**{**

**q=p;**

**while(\*q!=’\0′)**

**{**

**\*q=\*(q+1);**

**q++;**

**}**

**}**

**else**

**{**

**p++;**

**}**

**}**

**}**

77、链表节点结构如下：

struct STUDENT

{

long num;

float score;

STUDENT \*pNext;

};

编写实现将两棵有序（按学号从小到大）的链表合并的函数，要求合并后的链表有序（按学号从小到大）

STUDENT \*EmergeList(STUDENT \*pHead1,STUDENT \*pHead2)

{

//取小者为表头

STUDENT \* pHead;

STUDENT \*p1;

STUDENT \*p2;

STUDENT \*pCur;

STUDENT \*pTemp;

if (pHead1->num<= pHead2->num)

{

pHead = pHead1;

p2 = pHead2;

p1=pHead1->pNext;

}

else

{

pHead = pHead2;

p1 = pHead1;

p2=pHead2->pNext;

}

pCur=pHead;

while(p1!=NULL&&p2!=NULL)

{

if(p2->num<p1->num)

{

pCur->pNext=p2;

p2=p2->pNext;

pCur=pCur->pNext;

}

else if(p2->num>p1->num)

{

pCur->pNext=p1;

p1=p1->pNext;

pCur=pCur->pNext;

}

else if(p2->num==p1->num)

{

pCur->pNext=p1;

p1=p1->pNext;

pCur=pCur->pNext;

pTemp= p2;

p2=p2->pNext;

delete pTemp;

pTemp = NULL;

}

}

if(NULL==p1)

{

pCur->pNext=p2;

}

else if(NULL==p2)

{

pCur->pNext=p1;

}

return pHead;

}

78、封装CMyString类，要求声明和实现分开，声明见MyString.h，该类的声明可以添加内容，完成STLTest文件夹下main文件的要求。

参见STLTest Answer文件夹

79.请你分别画出OSI的七层网络结构图和TCP/IP的五层结构图。

80.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？

81.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？

82.请问C++的类和C里面的struct有什么区别？

83.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？

84. 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

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

85. 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<sizeof(str)/sizeof(str[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;

86. 以下代码有什么问题？[C难]

void char2Hex( char c ) // 将字符以16进制表示

{

char ch = c/0×10 + ’0′; if( ch > ’9′ ) ch += (‘A’-’9′-1);

char cl = c%0×10 + ’0′; if( cl > ’9′ ) cl += (‘A’-’9′-1);

cout << ch << cl << ‘ ‘;

}

char str[] = “I love 中国”;

for( size\_t i=0; i<strlen(str); ++i )

char2Hex( str[i] );

cout << endl;

87. 以下代码有什么问题？[C++易]

struct Test

{

Test( int ) {}

Test() {}

void fun() {}

};

void main( void )

{

Test a(1);

a.fun();

Test b();

b.fun();

}

88. 以下代码有什么问题？[C++易]

cout << (true?1:”1″) << endl;

89. 以下代码能够编译通过吗，为什么？[C++易]

unsigned int const size1 = 2;

char str1[ size1 ];

unsigned int temp = 0;

cin >> temp;

unsigned int const size2 = temp;

char str2[ size2 ];

90. 以下代码中的输出语句输出0吗，为什么？[C++易]

struct CLS

{

int m\_i;

CLS( int i ) : m\_i(i) {}

CLS()

{

CLS(0);

}

};

CLS obj;

cout << obj.m\_i << endl;

91. C++中的空类，默认产生哪些类成员函数？[C++易]

答：

class Empty

{

public:

Empty();                             // 缺省构造函数

Empty( const Empty& );               // 拷贝构造函数

~Empty();                            // 析构函数

Empty& operator=( const Empty& ); // 赋值运算符

Empty\* operator&();                  // 取址运算符

const Empty\* operator&() const;      // 取址运算符 const

};

92. 以下两条输出语句分别输出什么？[C++难]

float a =1.0f;

cout << (int)a << endl;

cout << (int&)a << endl;

cout << boolalpha << ( (int)a == (int&)a ) << endl; // 输出什么？

float b =0.0f;

cout << (int)b << endl;

cout << (int&)b << endl;

cout << boolalpha << ( (int)b == (int&)b ) << endl; // 输出什么？

93. 以下反向遍历array数组的方法有什么错误？[STL易]

vector array;

array.push\_back( 1 );

array.push\_back( 2 );

array.push\_back( 3 );

for( vector::size\_type i=array.size()-1; i>=0; –i ) // 反向遍历array数组

{

cout << array[i] << endl;

}

94. 以下代码有什么问题？[STL易]

typedef vector IntArray;

IntArray array;

array.push\_back( 1 );

array.push\_back( 2 );

array.push\_back( 2 );

array.push\_back( 3 );

// 删除array数组中所有的2

for( IntArray::iterator itor=array.begin(); itor!=array.end(); ++itor )

{

if( 2 == \*itor ) array.erase( itor );

}

95. 写一个函数，完成内存之间的拷贝。[考虑问题是否全面]

答：

void\* mymemcpy( void \*dest, const void \*src, size\_t count )

{

char\* pdest = static\_cast<char\*>( dest );

const char\* psrc = static\_cast<const char\*>( src );

if( pdest>psrc && pdest<psrc+cout ) 能考虑到这种情况就行了

{

for( size\_t i=count-1; i!=-1; –i )

pdest[i] = psrc[i];

}

else

{

for( size\_t i=0; i<count; ++i )

pdest[i] = psrc[i];

}

return dest;

}

int main( void )

{

char str[] = “0123456789″;

mymemcpy( str+1, str+0, 9 );

cout << str << endl;

system( “Pause” );

return 0;

}

华为C/C++笔试题(附答案)2008年02月15日星期五 18:001.写出判断ABCD四个表达式的是否正确, 若正确, 写出经过表达式中 a的值(3分)

96.int a = 4;

(A)a += (a++); (B) a += (++a) ;(C) (a++) += a;(D) (++a) += (a++);

a = ?

答：C错误，左侧不是一个有效变量，不能赋值，可改为(++a) += a;

改后答案依次为9,10,10,11

97.某32位系统下, C++程序，请计算sizeof 的值(5分).

char str[] = “http://www.ibegroup.com/”

char \*p = str ;

int n = 10;

请计算

sizeof (str ) = ？（1）

sizeof ( p ) = ？（2）

sizeof ( n ) = ？（3）

void Foo ( char str[100]){

请计算

sizeof( str ) = ？（4）

}

void \*p = malloc( 100 );

请计算

sizeof ( p ) = ？（5）

答：（1）17 （2）4 （3） 4 （4）4 （5）4

98. 回答下面的问题. (4分)

(1).头文件中的 ifndef/define/endif 干什么用？预处理

答：防止头文件被重复引用

(2). ＃i nclude 和 ＃i nclude “filename.h” 有什么区别？

答：前者用来包含开发环境提供的库头文件，后者用来包含自己编写的头文件。

(3).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”声明？

答：函数和变量被C++编译后在符号库中的名字与C语言的不同，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。由于编译后的名字不同，C++程序不能直接调

用C 函数。C++提供了一个C 连接交换指定符号extern“C”来解决这个问题。

(4). switch()中不允许的数据类型是?

答：实型

99. 回答下面的问题(6分)

(1).Void GetMemory(char \*\*p, int num){

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

}

void Test(void){

char \*str = NULL;

GetMemory(&str, 100);

strcpy(str, “hello”);

printf(str);

}

请问运行Test 函数会有什么样的结果？

答：输出“hello”

(2). void Test(void){

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

strcpy(str, “hello”);

free(str);

if(str != NULL){

strcpy(str, “world”);

printf(str);

}

}

请问运行Test 函数会有什么样的结果？

答：输出“world”

100. char \*GetMemory(void){

char p[] = “hello world”;

return p;

}

void Test(void){

char \*str = NULL;

str = GetMemory();

printf(str);

}

请问运行Test 函数会有什么样的结果？

答：无效的指针，输出不确定

101. 编写strcat函数(6分)

已知strcat函数的原型是char \*strcat (char \*strDest, const char \*strSrc);

其中strDest 是目的字符串，strSrc 是源字符串。

（1）不调用C++/C 的字符串库函数，请编写函数 strcat

答：

VC源码：

char \* \_\_cdecl strcat (char \* dst, const char \* src)

{

char \* cp = dst;

while( \*cp )

cp++; /\* find end of dst \*/

while( \*cp++ = \*src++ ) ; /\* Copy src to end of dst \*/

return( dst ); /\* return dst \*/

}

（2）strcat能把strSrc 的内容连接到strDest，为什么还要char \* 类型的返回值？

答：方便赋值给其他变量

102.MFC中CString是类型安全类么？

答：不是，其它数据类型转换到CString可以使用CString的成员函数Format来转换

103.C++中为什么用模板类。

答：(1)可用来创建动态增长和减小的数据结构

（2）它是类型无关的，因此具有很高的可复用性。

（3）它在编译时而不是运行时检查数据类型，保证了类型安全

（4）它是平台无关的，可移植性

（5）可用于基本数据类型

104.CSingleLock是干什么的。

答：同步多个线程对一个数据类的同时访问

105.NEWTEXTMETRIC 是什么。

答：物理字体结构，用来设置字体的高宽大小

106.程序什么时候应该使用线程，什么时候单线程效率高。

答：1．耗时的操作使用线程，提高应用程序响应

2．并行操作时使用线程，如C/S架构的服务器端并发线程响应用户的请求。

3．多CPU系统中，使用线程提高CPU利用率

4．改善程序结构。一个既长又复杂的进程可以考虑分为多个线程，成为几个独立或半独

立的运行部分，这样的程序会利于理解和修改。

其他情况都使用单线程。

107.Windows是内核级线程么。

答：见下一题

108.Linux有内核级线程么。

答：线程通常被定义为一个进程中代码的不同执行路线。从实现方式上划分，线程有两

种类型：“用户级线程”和“内核级线程”。 用户线程指不需要内核支持而在用户程序

中实现的线程，其不依赖于操作系统核心，应用进程利用线程库提供创建、同步、调度

和管理线程的函数来控制用户线程。这种线程甚至在象 DOS 这样的操作系统中也可实现

，但线程的调度需要用户程序完成，这有些类似 Windows 3.x 的协作式多任务。另外一

种则需要内核的参与，由内核完成线程的调度。其依赖于操作系统核心，由内核的内部

需求进行创建和撤销，这两种模型各有其好处和缺点。用户线程不需要额外的内核开支

，并且用户态线程的实现方式可以被定制或修改以适应特殊应用的要求，但是当一个线

程因 I/O 而处于等待状态时，整个进程就会被调度程序切换为等待状态，其他线程得不

到运行的机会；而内核线程则没有各个限制，有利于发挥多处理器的并发优势，但却占

用了更多的系统开支。

Windows NT和OS/2支持内核线程。Linux 支持内核级的多线程

109.C++中什么数据分配在栈或堆中，New分配数据是在近堆还是远堆中？

答：栈: 存放局部变量，函数调用参数,函数返回值，函数返回地址。由系统管理

堆: 程序运行时动态申请，new 和　malloc申请的内存就在堆上

110.使用线程是如何防止出现大的波峰。

答：意思是如何防止同时产生大量的线程，方法是使用线程池，线程池具有可以同时提

高调度效率和限制资源使用的好处，线程池中的线程达到最大数时，其他线程就会排队

等候。

111.一般数据库若出现日志满了，会出现什么情况，是否还能使用？

答：只能执行查询等读操作，不能执行更改，备份等写操作，原因是任何写操作都要记

录日志。也就是说基本上处于不能使用的状态。

112 SQL Server是否支持行级锁，有什么好处？

答：支持，设立封锁机制主要是为了对并发操作进行控制，对干扰进行封锁，保证数据

的一致性和准确性，行级封锁确保在用户取得被更新的行到该行进行更新这段时间内不

被其它用户所修改。因而行级锁即可保证数据的一致性又能提高数据操作的迸发性。

113 关于内存对齐的问题以及sizof()的输出

答：编译器自动对齐的原因：为了提高程序的性能，数据结构（尤其是栈）应该尽可能

地在自然边界上对齐。原因在于，为了访问未对齐的内存，处理器需要作两次内存访问

；然而，对齐的内存访问仅需要一次访问。

114. int i=10, j=10, k=3; k\*=i+j; k最后的值是？

答：60，此题考察优先级，实际写成： k\*=(i+j);，赋值运算符优先级最低

115.对数据库的一张表进行操作,同时要对另一张表进行操作,如何实现?

答：将操作多个表的操作放入到事务中进行处理

116.TCP/IP 建立连接的过程?(3-way shake)

答：在TCP/IP协议中，TCP协议提供可靠的连接服务，采用三次握手建立一个连接。

第一次握手：建立连接时，客户端发送syn包(syn=j)到服务器，并进入SYN\_SEND状

态，等待服务器确认；

第二次握手：服务器收到syn包，必须确认客户的SYN（ack=j+1），同时自己也发送一个

SYN包（syn=k），即SYN+ACK包，此时服务器进入SYN\_RECV状态；

第三次握手：客户端收到服务器的SYN＋ACK包，向服务器发送确认包ACK(ack=k+1)

，此包发送完毕，客户端和服务器进入ESTABLISHED状态，完成三次握手。

117.ICMP是什么协议,处于哪一层?

答：Internet控制报文协议，处于网络层（IP层）

118.触发器怎么工作的?

答：触发器主要是通过事件进行触发而被执行的，当对某一表进行诸如UPDATE、 INSERT

、 DELETE 这些操作时，数据库就会自动执行触发器所定义的SQL 语句，从而确保对数

据的处理必须符合由这些SQL 语句所定义的规则。

119.winsock建立连接的主要实现步骤?

答：服务器端：socker()建立套接字，绑定（bind）并监听（listen），用accept（）

等待客户端连接。

客户端：socker()建立套接字，连接（connect）服务器，连接上后使用send()和recv（

），在套接字上写读数据，直至数据交换完毕，closesocket()关闭套接字。

服务器端：accept（）发现有客户端连接，建立一个新的套接字，自身重新开始等待连

接。该新产生的套接字使用send()和recv（）写读数据，直至数据交换完毕，closesock

et()关闭套接字。

120.动态连接库的两种方式?

答：调用一个DLL中的函数有两种方法：

1．载入时动态链接（load-time dynamic linking），模块非常明确调用某个导出函数

，使得他们就像本地函数一样。这需要链接时链接那些函数所在DLL的导入库，导入库向

系统提供了载入DLL时所需的信息及DLL函数定位。

2．运行时动态链接（run-time dynamic linking），运行时可以通过LoadLibrary或Loa

dLibraryEx函数载入DLL。DLL载入后，模块可以通过调用GetProcAddress获取DLL函数的

出口地址，然后就可以通过返回的函数指针调用DLL函数了。如此即可避免导入库文件了

。

121.IP组播有那些好处?

答：Internet上产生的许多新的应用，特别是高带宽的多媒体应用，带来了带宽的急剧

消耗和网络拥挤问题。组播是一种允许一个或多个发送者（组播源）发送单一的数据包

到多个接收者（一次的，同时的）的网络技术。组播可以大大的节省网络带宽，因为无

论有多少个目标地址，在整个网络的任何一条链路上只传送单一的数据包。所以说组播

技术的核心就是针对如何节约网络资源的前提下保证服务质量。

122. 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<sizeof(str)/sizeof(str[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

123，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。

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。

124.

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].

125.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。

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。

126、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。

127、将树序列化 转存在数组或 链表中

128.下面哪个实体不是SIP协议定义的（A）。

A ）MGC    B）UA     C）proxy     D）Redirector

UA = User Agent 用户代理,指客户端的协议栈

PROXY = SIP Proxy ,指SIP服务器

Redirector = 重定向模块。一般用于跨服务器通讯

129．VOIP中本端与对端呼叫接通后，将通话转接到第三方称之为（C）。

A）呼叫转接   B）呼叫前转   C）呼叫转移   D）三方通话

跟普通电话相同，A，B都是没有接通前就把呼叫转接。D是指三方同时在通话。

130．VOIP的主要优点是（D）

A）价格便宜并且能为客户提供更好的增值服务。

B）语音质量比传统的PSTN电话好。

C）通话的安全性和可靠性有更高的保障。

D）不需要服务提供商就可以使用。

音质，可靠性是传统电话好。这题的问题在于，增值服务是指什么？如是SP则VOIP不支持。还是服务器提供商是指什么？VOIP需要服务器。

131．下面哪个技术不属于语音处理技术（D）

A）静音检测

B）分组丢失补偿

C）同声消除

D）网路穿越

D是网络传输的问题，主要是穿透NAT网关。特别是比较复杂的网络。

132．SIP协议是使用下面哪种方式编解码的（）

A）ASN.1     B）BER      C）ABNF    D）PER

网络应用

133．在不同网络中MTU会发生变化，由于MTU的变化，会导致那些值也相应发生变化（A）

A）IP总长度字段

B）TCP MSS字段

C）UDP长度字段

D）TCP长度字段

待查,MTU变化会让IP分片或重组。因此变化就是IP

134．下列关于TCP和UDP的正确的说法是（C）

A）TCP实时性比UDP好

B）TCP比UDP效率高

C）TCP首部比UDP的首部长

D）TCP安全性比UDP高

实时性，效率。安全性，TCP不见得比UDP高

135．一个套接口应该匹配（D）

A）源目标IP

B）源目标IP端口

C）目标IP和目标端口

D）本地IP和本地端口

SOCKET相当一IP连接上用端口标识队列

136．TCP服务器出现异常并马上重启，通常会出现那些情况（）

A）socket调用失败

B）bind调用失败

C）listen调用失败

D）select调用失败

此题有问题，一般软件很难自动重启。而TCP服务器可以在任何一个阶段出问题，上述哪一个都能出现，这个本意应该是指Select.

底层开发

137．在一台以字节为最小存储单位的机器上，如果把0×12345678写到从0×0000开始的地址上，下列关于big—endian和little—enddian说法正确的是（B）

A）在big—endian模式下，地址0×0000到0×0003存储的数据依次为：0×56，0×78，0×12，0×34

B）在big—endian模式下，地址0×0000到0×0003存储的数据依次为：0×12，0×34，0×56，0×78

C）在little—endian模式下，地址0×0000到0×0003存储的数据依次为：0×34，0×12，0×78，0×56

D）在little—endian模式下，地址0×0000到0×0003存储的数据依次为：0×56，0×78，0×12，0×34

138．以下关于交叉编译器概述正确的是（A）

A）交叉编译器一般按照CPU类型分类，不同的CPU就有不同的交叉编译器

B）交叉编译器的速度比其他编译器的速度要快

C）linux开发环境中的交叉编译器不是gcc编译器

D）交叉编译器编译出来的目标文件一般也能在开发机（普通PC）上运行

139．以下关于linux下中断的说法正确的是（）

A）中断只能由硬件产生

B）中断服务例程不能运行参数传递

C）中断服务例程可以在函数结束的时候返回一个值

D）中断服务例程是不可中断的

D,B?

140．以下关于linux下系统调用的说法错误的是（）

A）应用程序通过系统调用访问内核

B）每个系统调用都有一个唯一的系统调用号

C）用户可以定制自己的系统调用

D）

可能是A，系统调用在内核执行。但这里访问比较模糊。

141．关于SPI说法正确的是（）

A）SPI工作在全双工模式下

B）

C）

D）SPI接口一般工作在主从模式下

C语言

142．Char Test[10];char \*pTest=test;问：&Test在数值上等于（A）

A）Test    B) Test[0]    C)&pTest     D)符号表某个符号的地址

&Test相当于二维指针首指针，TEST是一维的首指针

143．在顺序表{3,6,8,10,12,15,16,18,21,25,30}中，用二分法查找关键码值11，所雪的关键码比较次数为（）B?

A）2   B）3   C）4     D）5

144．单链表中每个结点中包括一个指针link，它向该结点，现要将指针q指向的新结点放到指针p指向的单链表接点之后，下面的操作序列中哪一个是正确的（C）

A）q:=p^.link;p^.link:=q^:link

B)p^.link:=q^.link;q:=p^.link

C)q^.link:=p^.link;p^.link:=q

D)p^.link:=q;q^.link:=p^.link

145．以下叙述正确的是（C）

A）在C程序中，main函数必须位于程序的最前面

B）C程序的每行中只能写一条语句

C）C语言本身没有输入输出语句

D）在对一个C程序进行编译的过程中，可发现注释中的拼写错误

146．有以下程序

Main()

{

Char a[]=”programming”,b[]=”language”;

Char \*p1,\*p2;

Int i;

P1=a;p2=b;

For(i=0;i<7;i++)

If(\*(p1+i)==\*(p2+i))

Printf(“%c”,\*(p1+i));

)

打印出什么（）

147．请简述以下两个for循环的优缺点（6分）

|  |  |
| --- | --- |
| // 第一个  for (i=0; i<N; i++)  {  if (condition)  DoSomething();  else  DoOtherthing();  } | // 第二个  if (condition)  {  for (i=0; i<N; i++)  DoSomething();  }  else  {  for (i=0; i<N; i++)  DoOtherthing();  } |
| 优点：  当N较大时，效率较高  缺点：  每一个循环都要做一次if判断 | 优点：  当N较小时，效率较高  缺点：  循环次数较多 |

148．位运算：给定一个整型变量a，（1）给bit3置数（2）清除bit3（6分）

a|=(0×1<<3);

a&=~(0×1<<3);

149．评述下面的代码（6分）

Main（）

{

Int a[100]；

Int \*p；

P=(（unsigned int \*）a+1);

Printf(“0x%x”,\*p);

}

1.数组没有赋初值，

2.指针类型转换不一致

3..打印一个未赋值的整数值,p=a[1]

#include <stdio.h>

main()

{

int a[100]={1,2,3};

unsigned int \*p;

p=((unsigned int \*)a+1);

printf(“%x,%x\n”,a,p);

printf(“0x%x”,\*p);

getchar();

}

150．编程题；(10分)

从键盘输入一组字符；

字母按升序排列；其余的字符按升序排列

字母放前，其余的放后

例如：输入：\_@AB-@ab  结果：ABab-@@\_

151.简述需求分析的过程和意义

152.网状、层次数据模型与关系数据模型的最大的区别是什末

153.软件质量保证体系是什末 国家标准中与质量保证管理相关的几个标准是什末 编号和全称是什末号和全称是什末

153文件格式系统有哪几种类型？分别说说win95、win98、winMe、w2k、winNT、winXP分别支持那些文件系统

154．我现在有个程序，发现在WIN98上运行得很慢，怎么判别是程序存在问题还是软硬件系统存在问题？  
155．有关P2P点对点文件传输的原理

156.一台计算机的IP是192.168.10.71子网掩码255.255.255.64与192.168.10.201是同一局域网吗?  
157.internet中e-mail协仪,IE的协仪,NAT是什么,有什么好处,能带来什么问题?DNS是什么,它是如何工作的?  
158.PROXY是如何工作的?  
169.win2k系统内AT命令完成什么功能,Messenger服务是做什么,怎么使用?  
170进程,线程的定义及区别  
171,32位操作系统内,1进程地址空间多大,进程空间与物理内存有什么关系?  
172.网络攻击常用的手段,防火墙如何保证安全.  
173.如何配静态IP,如何测网络内2台计算机通不通,PING一次返几个数据包?  
174.WIN9X与WINNT以上操作系统有”服务”吗,服务是什么,如何停止服务?  
175.AD在WIN2KSERVER上建需什么文件格式,AD是什么?XP多用户下”注销”与”切换”的区别.

176．UDP可以跨网段发送吗?

177.最简单的确认远程计算机(win2K以上)某个监听端口是正常建立的?

178. 找错

void test1()

{

char string[10];

char\* str1=”0123456789″;

strcpy(string, str1);

}

答：表面上并且编译都不会错误。但如果string数组原意表示的是字符串的话，那这个赋值就没有达到意图。最好定义为char string[11]，这样最后一个元素可以存储字符串结尾符’\0′;

void test2()

{

char string[10], str1[10];

for(int I=0; I<10;I++)

{

str1[I] =’a';

}

strcpy(string, str1);

}

答：strcpy使用错误，strcpy只有遇到字符串末尾的’\0′才会结束，而str1并没有结尾标志，导致strcpy函数越界访问，不妨让str1[9]=’\0′，这样就正常了。

void test3(char\* str1)

{

char string[10];

if(strlen(str1)<=10)

{

strcpy(string, str1);

}

}

答：这又会出现第一道改错题的错误了。strlen(str1)算出来的值是不包含结尾符’\0′的，如果str1刚好为10个字符＋1结尾符，string就得不到结尾符了。可将strlen(str1)<=10改为strlen(str1)<10。

179. 找错

#define MAX\_SRM 256

DSN get\_SRM\_no()

{

static int SRM\_no;

int I;

for(I=0;I<MAX\_SRM;I++,SRM\_no++)

{

SRM\_no %= MAX\_SRM;

if(MY\_SRM.state==IDLE)

{

break;

}

}

if(I>=MAX\_SRM)

return (NULL\_SRM);

else

return SRM\_no;

}

答：我不知道这段代码的具体功能，但明显有两个错误

1，SRM\_no没有赋初值

2，由于static的声明，使该函数成为不可重入（即不可预测结果）函数，因为SRM\_no变量放在程序的全局存储区中，每次调用的时候还可以保持原来的赋值。这里应该去掉static声明。

180. 写出程序运行结果

int sum(int a)

{

auto int c=0;

static int b=3;

c+=1;

b+=2;

return(a+b+c);

}

void main()

{

int I;

int a=2;

for(I=0;I<5;I++)

{

printf(“%d,”, sum(a));

}

}

答：8,10,12,14,16

该题比较简单。只要注意b声明为static静态全局变量，其值在下次调用时是可以保持住原来的赋值的就可以。

181.

int func(int a)

{

int b;

switch(a)

{

case 1: b=30;

case 2: b=20;

case 3: b=16;

default: b=0;

}

return b;

}

则func(1)=?

答：func(1)=0，因为没有break语句，switch中会一直计算到b=0。这是提醒我们不要忘了break。呵呵。

182:

int a[3];

a[0]=0; a[1]=1; a[2]=2;

int \*p, \*q;

p=a;

q=&a[2];

则a[q-p]=?

答：a[q-p]=a[2]=2;这题是要告诉我们指针的运算特点

183.

定义 int \*\*a[3][4], 则变量占有的内存空间为：\_\_\_\_\_

答：此处定义的是指向指针的指针数组，对于32位系统，指针占内存空间4字节，因此总空间为3×4×4＝48。

184.

编写一个函数，要求输入年月日时分秒，输出该年月日时分秒的下一秒。如输入2004年12月31日23时59分59秒，则输出2005年1月1日0时0分0秒。

答：

/\*输入年月日时分秒，输出年月日时分秒的下一秒，输出仍然在原内存空间\*/

bool NextMinute(int \*nYear,int \*nMonth,int \*nDate,int \*nHour,int \*nMinute,int \*nSecond)

{

if(\*nYear<0 || \*nMonth>12 || \*nMonth<0 || \*nHour>23 || \*nHour<0 || \*nMinute<0 || \*nMinute>59 || \*nSecond<0 || \*nSecond>59) return false;

int nDays;

switch(\*nMonth)

{

case 1:

case 3:

case 5:

case 7:

case 8:

case 10:

case 12:

nDays=31;

break;

case 2:// 判断闰年

if(\*nYear%400==0||\*nYear%100!=0&&\*nYear%4==0)

{

nDays=29;

}

else

{

nDays=28;

}

break;

default:

nDays=30;

break;

}

if(\*nDate<0 || \*nDate>nDays) return false;

(\*nSecond)++;  // 秒加1

if(\*nSecond>=60)  // 秒满60，做出特殊处理，下面时，日，月等类同

{

\*nSecond=0;

(\*nMinute)++;

if(\*nMinute>=60)

{

\*nMinute=0;

(\*nHour)++;

if(\*nHour>=24)

{

\*nHour=0;

(\*nDate)++;

if(\*nDate>nDays)

{

\*nDate=1;

(\*nMonth)++;

if(\*nMonth>12)

{

\*nMonth=1;

(\*nYear)++;

}

}

}

}

}

return true;

}

/\*示例可运行代码\*/

void main()

{

int nYear=2004,nMonth=12,nDate=31,nHour=23,nMinute=59,nSecond=59;

bool res = NextMinute(&nYear,&nMonth,&nDate,&nHour,&nMinute,&nSecond);

if(res)

printf(“The result:%d-%d-%d %d:%d:%d”,nYear,nMonth,nDate,nHour,nMinute,nSecond);

else

printf(“Input error!\n”);

}

185． 写一个函数，判定运算环境（16位以上字长）是little-endian还是big-endian

186． 操作系统的主要组成部分？

187．操作系统中进程调度策略有哪几种？

188．进程间主要的通讯方式？

189．写出进程的主要状态？

190．以太网物理地址和IP地址转换采用什么协议？

191．IP地址的编码分为哪两部分？

192．写出以太网数据帧格式

/193．8031和8051的主要区别？

194．C++中的空类，默认产生哪些类成员函数？

分析以下程序的执行结果

#include<iostream.h>

class base

{

public:

base(){cout<< “constructing base class”<<endl;}

~base(){cout<<”destructing base class”<<endl;}

};

class subs:public base

{

public:

subs(){cout<<”constructing sub class”<<endl;}

~subs(){cout<<”destructing sub class”<<endl;}

};

void main()

{

subs s;

}

195.指出下面程序的错误

#define SIZE 5

struct String

{

char \*pData;

};

void  main()

{

char \*pData;

};

void  main()

{

char acValue1[SIZE]={‘H’,’E’,’L’,’L’,’O’};

char acValue2[SIZE]={‘W’,’O’,’L’,’D’};

struct String a,b,c;

a.pData=malloc(SIZE\*sizeof(char));

memcpy(a.pData,acValuel,SIZE);

b.pData=malloc(SIZE\*sizeof(char));

mempcpy(b.pData,acValue2,SIZE);

b=a;

free(a.pData);

c=b;

}

196.指出下面两段程序的区别

【**1**】

main()

{

int loop=1;

int arr[10];

int i=0;

while(loop<5)

{

for(;i<=10;i++)

{

arr[i]=1;

}

loop++;

}

}

【**2**】

main()

{

int arr[10];

int loop=1;

int i=0;

while(loop<5)

{

for(i=0;i<=10;i++)

{

arr[i]=1;

}

loop++;

}

}

197.指出下面程序的错误（函数GetValue返回 unsigned char类型的值）

#define  MAXNUM  400;

unsigned char uclndex,uclnputVar,aucArray[MAXNUM];

for(ucIndx =0;ucIndex<=MAXNUM;ucIndex++)

{

aucArray[ucIndex]=aucArray[ucIndex]+1;

}

ucInputVar=GetValue();

for(ucIndex=0;ucIndex>(ucInputVar-1);ucIndex++)

{

aucArray[ucIndex]=aucArray[ucIndex]\*2+1;

}

198.什么是com和ActiveX，简述DCOM。  
答：COM（Component Object Mode）即组件对象模型，是组件之间相互接口的规范。其作用是使各种软件构件和应用软件能够用一种统一的标准方式进行交互。COM不是一种面向对象的语言，而是一种与源代码无关的二进制标准。  
ActiveX是Microsoft提出的一套基于COM的构件技术标准，实际上是对象嵌入与炼接（OLE）的新版本。基于分布式环境下的COM被称作DCOM(Distribute COM，分布式组件对象模型)，它实现了COM对象与远程计算机上的另一个对象之间直接进行交互。DCOM规范定义了分散对象创建和对象间通信的机制，DCOM是ActiveX的基础，因为ActiveX主要是针对Internet应用开发（相比OLE）的技术，当然也可以用于普通的桌面应用程序。

199.列出3个常用网络协议使用的端口。  
答：HTTP协议用80端口，FTP协议用21端口，POP3协议用110端口

199  什么是ODBC?  
答：ODBC(Open Database Connectivity，开放数据库互连)是微软公司开放服务结构(WOSA，Windows Open Services Architecture)中有关数据库的一个组成部分，它建立了一组规范，并提供了一组对数据库访问的标准API（应用程序编程接口）。ODBC的最大优点是能以统一的方式(用它提供的API访问数据库)处理所有的数据库。

200  结构化编程和goto语句的区别和关系？  
答：结构化编程设计思想采用了模块分解与功能抽象和自顶向下、分而治之的方法，从而有效地将一个较复杂的程序系统设计任务分解成许多易于控制和处理的子程序，便于开发和维护。goto语句可以实现无条件跳转，改变程序流向，破坏结构化编程设计风格。但goto语句在结构化编程中并非不可使用，只是要受到限制的使用。

199  MFC中SendMessage和PostMessage的区别？  
答：PostMessage 和SendMessage的区别主要在于是否等待应用程序做出消息处理。PostMessage只是把消息放入队列，然后继续执行；而SendMessage必须等待应用程序处理消息后才返回继续执行。这两个函数的返回值也不同，PostMessage的返回值表示PostMessage函数执行是否正确，而SendMessage的返回值表示其他程序处理消息后的返回值。  
202．改错  
#include  
#include  
class CBuffer  
{  
char \* m\_pBuffer;  
int m\_size;  
public:  
CBuffer()  
{  
m\_pBuffer=NULL;  
}  
~CBuffer()  
{  
Free();  
}  
void Allocte(int size) （3） {  
m\_size=size;  
m\_pBuffer= new char[size];  
}  
private:  
void Free()  
{  
if(m\_pBuffer!=NULL)  
{  
delete m\_pBuffer;  
m\_pBuffer=NULL;  
}  
}  
public:  
void SaveString(const char\* pText) const  
{  
strcpy(m\_pBuffer, pText);  
}  
char\* GetBuffer() const  
{  
return m\_pBuffer;  
}  
};

void main (int argc, char\* argv[])  
{  
CBuffer buffer1;  
buffer1.SaveString(“Microsoft”);  
printf(buffer1.GetBuffer());  
}

答：改正后  
主要改正SaveString函数  
将  
void SaveString(const char\* pText) const  
{  
strcpy(m\_pBuffer, pText);  
}  
改为  
void SaveString(const char\* pText) （1）  
{  
Allocte(strlen(pText)+1); （2）  
strcpy(m\_pBuffer, pText);  
}  
原因：  
（1） const成员函数表示不会修改数据成员，而SaveString做不到，去掉const声明  
（2） m\_pBuffer指向NULL，必须用Allocte分配空间才能赋值。  
（3） 另外需要将Allocte成员函数声明为私有成员函数更符合实际

203.下来程序想打印“Welcome MSR Asia”，改正错误  
#include  
#include  
char \* GetName (void)  
{  
//To return “MSR Asia” String  
char name[]=”MSR Asia”;  
return name;  
}  
void main(int argc, char\* argv[])  
{  
char name[32];  
//Fill in zeros into name  
for(int i=0;i<=32;i++)  
{  
name[i]=’\0′;  
}  
//copy “Welcome” to name  
name=”Welcome”;  
//Append a blank char  
name[8]=” “;  
//Append string to name  
strcat(name,GetName());  
//print out  
printf(name);  
}

答：改正后为  
#include  
#include  
char \* GetName (void)  
{  
//To return “MSR Asia” String  
//char name[]=”MSR Asia”; （1）  
char \*name=(char \*)malloc(strlen(“MSR Asia”)+1);  
strcpy(name,”MSR Asia”);  
return name;  
}  
void main(int argc, char\* argv[])  
{  
char name[32];  
//Fill in zeros into name  
for(int i=0;i<=32;i++)  
{  
name[i]=’\0′;  
}  
//copy “Welcome” to name  
//name=”Welcome”; （2）  
strcat(name,”Welcome “);  
//Append a blank char  
// name[8]=’ ‘; （3）  
//Append string to name  
char \*p=GetName(); （4）  
strcat(name,p);  
free (p);  
//print out  
printf(name);  
}  
原因：（1）在函数内部定义的变量在函数结束时就清空了，必须动态分配内存  
（2）字符串赋值语句错误，应该用strcat  
（3）该语句无效，可去掉  
（4）定义一个指针指向动态分配的内存，用完后需用free语句释放

204．写出下面程序的输出结果  
#include  
class A  
{  
public:  
void FuncA()  
{  
printf(“FuncA called\n”);  
}  
virtual void FuncB()  
{  
printf(“FuncB called\n”);  
}  
};

class B: public A  
{  
public:  
void FuncA()  
{  
A::FuncA();  
printf(“FuncAB called\n”);  
}  
virtual void FuncB()  
{  
printf(“FuncBB called\n”);  
}  
};

void main(void)  
{  
B b;  
A \*pa;  
pa=&b;  
A \*pa2=new A;  
b.FuncA(); （1）  
b.FuncB(); （2）  
pa->FuncA(); （3）  
pa->FuncB(); （4）  
pa2->FuncA(); （5）  
pa2->FuncB();  
delete pa2;  
}  
答：  
1．b.FuncA(); 输出  
FuncA called  
FuncAB called  
2．b.FuncB();输出  
FuncBB called  
上两者好理解，直接调用类B的相应成员函数  
3．pa->FuncA();输出  
FuncA called 调用类A的FuncA()  
4．pa->FuncB();输出  
FuncBB called调用类B的FuncB()，原因是C++的动态决议机制，当基类函数声明为virtual时，指向派生类对象的基类指针来调用该函数会选择派生类的实现，除非派生类没有才调用基类的虚函数。还有一点注意的是：指向基类类型的指针可以指向基类对象也可以指向派生类对象，如pa=&b;  
5. pa2->FuncA();  
pa2->FuncB();输出  
FuncA called  
FuncB called  
这也好理解，直接调用类A的相应成员函数

206．In the main() function, after ModifyString(text) is called, what’s the value of ‘text’?  
#include  
#include  
int FindSubString(char\* pch)  
{  
int count=0;  
char\* p1=pch;  
while(\*p1!=’\0′)  
{  
if(\*p1==p1[1]-1)  
{  
p1++;  
count++;  
}  
else  
{  
break;  
}  
}  
int count2=count;  
while(\*p1!=’\0′)  
{  
if(\*p1==p1[1]+1)  
{  
p1++;  
count2–;  
}  
else  
{  
break;  
}  
}  
if(count2==0)  
return count;  
return 0;  
}

void ModifyString(char\* pText)  
{  
char\* p1=pText;  
char\* p2=p1;  
while(\*p1!=’\0′)  
{  
int count=FindSubString(p1);  
if(count>0)  
{  
\*p2++=\*p1;  
sprintf(p2, “%I”, count);  
while(\*p2!= ‘\0′)  
{  
p2++;  
}  
p1+=count+count+1;  
}  
else  
{  
\*p2++=\*p1++;  
}  
}  
}  
void main(void)  
{  
char text[32]=”XYBCDCBABABA”;  
ModifyString(text);  
printf(text);  
}  
答：我不知道这个结构混乱的程序到底想考察什么，只能将最后运行结果写出来是XYBCDCBAIBAAP

207. Programming (Mandatory)  
Linked list  
a. Implement a linked list for integers,which supports the insertafter (insert a node after a specified node) and removeafter (remove the node after a specified node) methods;  
b. Implement a method to sort the linked list to descending order.  
答：题目的意思是实现一个整型链表，支持插入，删除操作（有特殊要求，都是在指定节点后进行操作），并写一个对链表数据进行降序排序的方法。  
那我们不妨以一个线性链表进行编程。  
// 单链表结构体为  
typedef struct LNode  
{  
int data;  
struct LNode \*next;  
}LNode, \*pLinkList;

// 单链表类  
class LinkList  
{  
private:  
pLinkList m\_pList;  
int m\_listLength;  
public:  
LinkList();  
~LinkList();  
bool InsertAfter(int afternode, int data);//插入  
bool RemoveAfter(int removenode);//删除  
void sort();//排序  
};

实现方法  
//insert a node after a specified node  
bool LinkList::InsertAfter(int afternode, int data)  
{  
LNode \*pTemp = m\_pList;  
int curPos = -1;  
if (afternode > m\_listLength ) // 插入点超过总长度  
{  
return false;  
}  
while (pTemp != NULL) // 找到指定的节点  
{  
curPos++;  
if (curPos == afternode)  
break;  
pTemp = pTemp->next;  
}  
if (curPos != afternode) // 节点未寻到，错误退出  
{  
return false;  
}  
LNode \*newNode = new LNode; // 将新节点插入指定节点后  
newNode->data = data;  
newNode->next = pTemp->next;  
pTemp->next = newNode;  
m\_listLength++;  
return true;  
}

//remove the node after a specified node  
bool LinkList::RemoveAfter(int removenode)  
{  
LNode \*pTemp = m\_pList;  
int curPos=-1;  
if (removenode > m\_listLength) // 删除点超过总长度  
{  
return false;  
}

// 找到指定的节点后一个节点，因为删除的是后一个节点  
while (pTemp != NULL)  
{  
curPos++;  
if (curPos == removenode+1)  
break;  
pTemp = pTemp->next;  
}  
if (curPos != removenode) // 节点未寻到，错误退出  
{  
return false;  
}  
LNode \*pDel = NULL; // 删除节点  
pDel = pTemp->next;  
pTemp->next = pDel->next;  
delete pDel;  
m\_listLength–;  
return true;  
}

//sort the linked list to descending order.  
void LinkList::sort()  
{  
if (m\_listLength<=1)  
{  
return;  
}  
LNode \*pTemp = m\_pList;  
int temp;  
// 选择法排序  
for(int i=0;i<M\_LISTLENGTH-1;I++)  
for(int j=i+1;j<M\_LISTLENGTH;J++)  
if (pTemp[i].data<PTEMP[J].DATA)  
{  
temp=pTemp[i].data;  
pTemp[i].data=pTemp[j].data;  
pTemp[j].data=temp;  
}  
}  
前两个函数实现了要求a，后一个函数sort()实现了要求b

208. Debugging (Mandatory)  
a. For each of the following recursive methods, enter Y in the answer box if the method terminaters (assume i=5), Otherwise enter N.  
（题目意思：判断下面的递归函数是否可以结束）  
static int f(int i){  
return f(i-1)\*f(i-1);  
}  
Ansewr: N,明显没有返回条件语句，无限递归了

static int f(int i){  
if(i==0){return 1;}  
else {return f(i-1)\*f(i-1);}  
}  
Ansewr:Y，当i＝0时可结束递归

static int f(int i){  
if(i==0){return 1;}  
else {return f(i-1)\*f(i-2);}  
}  
Ansewr:N，因为i=1时，f(i-2)=f(-1),进入一个无限递归中

209．编程  
将整数转换成字符串：void itoa(int,char);  
例如itoa(-123,s[])则s=“-123”;  
答：  
char\* itoa(int value, char\* string)  
{  
char tmp[33];  
char\* tp = tmp;  
int i;  
unsigned v;  
char\* sp;  
// 将值转为正值  
if (value < 0)  
v = -value;  
else  
v = (unsigned)value;  
// 将数转换为字符放在数组tmp中  
while (v)  
{  
i = v % 10;  
v = v / 10;  
\*tp++ = i+’0′;  
}  
// 将tmp里的字符填入string指针里，并加上负号（如果有）  
sp = string;  
if (value < 0)  
\*sp++ = ‘-’;  
while (tp > tmp)  
\*sp++ = \*–tp;  
\*sp = 0;  
return string;  
}

210.完成下列程序  
\*  
\*.\*.  
\*..\*..\*..  
\*…\*…\*…\*…  
\*….\*….\*….\*….\*….  
\*…..\*…..\*…..\*…..\*…..\*…..  
\*……\*……\*……\*……\*……\*……\*……  
\*…….\*…….\*…….\*…….\*…….\*…….\*…….\*…….  
#include  
#define N 8  
int main()  
{  
int i;  
int j;  
int k;  
———————————————————  
│ │  
│ │  
│ │  
———————————————————  
return 0;  
}  
答：#define N 8  
int main()  
{  
int i;  
int j;  
int k;

for(i=0;i<N;I++)  
{  
for(j=0;j<I+1;J++)  
{  
printf(“\*”);  
for(k=0;k<I;K++)  
printf(“.”);  
}  
printf(“\n”);  
}  
return 0;  
}

211．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 “stdio.h”  
#include “malloc.h”  
typedef struct TNode  
{  
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  
return 0;  
}

void append(int N)  
{  
TNode\* NewNode=(TNode \*)malloc(sizeof(TNode));  
NewNode->value=N;  
NewNode->left=NULL; //新增  
NewNode->right=NULL; //新增  
if(root==NULL)  
{  
root=NewNode;  
return;  
}  
else  
{  
TNode\* temp;  
temp=root;  
while((N>=temp->value && temp->left!=NULL)||(Nvalue && temp-

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

答：因为新节点的左右指针没有赋NULL值，至使下面的while循环不能正确结束而导致内存越界，最后崩溃（注意结束条件是temp->left!=NULL或temp->right!=NULL）。改正就是增加两条赋值语句，如上文红色部分字体就是新增的两条语句。

212.打印如下图案，共19行，只能有一个for循环（题目已经提供）  
\*   
\*\*\*   
\*\*\*\*\*   
\*\*\*\*\*\*\*   
\*\*\*\*\*\*\*\*\*   
\*\*\*\*\*\*\*\*\*\*\*  
\*\*\*\*\*\*\*\*\*\*\*\*\*   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   
\*\*\*\*\*\*\*\*\*\*\*\*\*   
\*\*\*\*\*\*\*\*\*\*\*   
\*\*\*\*\*\*\*\*\*   
\*\*\*\*\*\*\*   
\*\*\*\*\*   
\*\*\*   
\*   
for(i=0;i<19;i++)  
{

}  
答：  
#include “stdio.h”  
void main()  
{  
for(int i=0;i<19;i++)  
{  
int j=0;  
while (j<19)  
{  
if (i<=9)  
{  
if (j<=9)  
{  
if (i+j>=9)  
printf(“\*”);  
else  
printf(” “);  
}  
else  
if (j-i<=9)  
printf(“\*”);  
else  
printf(” “);  
}  
else  
{  
if (j<=9)  
{  
if (i-j<=9)  
printf(“\*”);  
else  
printf(” “);  
}  
else  
if (j+i<=27)  
printf(“\*”);  
else  
printf(” “);  
}  
j++;  
}  
printf(“\n”);  
}  
}

213.stack data （栈）存在于  
A.rom, B .flash C .eeprom D.ram E .none of the above  
答：D.ram。这题稍微涉及到一点硬件知识，ROM的全称是Read Only Memory，即只读存储器，flash ，eeprom都是ROM家族的一员，RAM是Random Access Memory的简称，意为随机存取存储器，也就是内存了。不管是堆还是栈都是放在内存里的。

214.  
int i;  
int x=0×12345678;  
unsigned char \*p=(unsigned char \*)&x;  
for(i=0;i<SIZEOF(X);I++)  
printf(“%2x”,\*(p+i));  
在80x86pc机器上运行结果？  
答：x在PC机上的内存存放顺序为78 56 34 12，高字节在前，低字节在后，因此输出78563412  
Sun Sparc Unix上运行结果？

215.  
char a[2][2][3]={{{1,6,3},{5,4,15}},{{3,5,33},{23,12,7}} };  
for(int i=0;i<12;i++)  
printf(“%d “,\_\_\_\_\_\_\_);  
在空格处填上合适的语句，顺序打印出a中的数字  
答：\*(\*(\*(a+i/6)+(i/3%2))+i%3)  
这题主要是要将输出的序号依次写出一些，如000,001,002,010,011,012,100,101…然后找序号变化规律

216.请用标准C语言实现一个双向循环链表的查找与删除。  
typedef struct doublecyclelink{  
int key;  
struct doublecyclelink \*prev;  
struct doublecyclelink \*next;  
}DoubleCycleLinkT;  
DoubleCycleLinkT \*findKey(DoubleCycleLinkT \*link,int key);  
遍历整个双向循环链表，将第一个与key值相同的结点移出链表，并返回。  
若没有找到则返回NULL。

答：  
函数为  
DoubleCycleLinkT \*findKey(DoubleCycleLinkT \*link,int key)  
{  
DoubleCycleLinkT \*p;  
p=link->next;  
while (p->next!=link) // 链表结尾  
{  
if (p->key==key) // 查找到key值相同,删除该节点,并返回  
{  
p->prev->next=p->next;  
p->next->prev=p->prev;  
free(p);  
return link;  
}  
else  
p=p->next; // 否则查找下一节点  
}  
if (p->next == link) return NULL; //没找到，返回NULL

}

217、请用标准C语言实现下列标准库函数，设计中不得使用其他库函数。  
char \*strstr(char \*str1,char \*str2);  
在字符串str1中，寻找字串str2，若找到返回找到的位置，否则返回NULL。  
答：  
函数为  
char \* strstr ( const char \* str1, const char \* str2 )  
{  
char \*cp = (char \*) str1;  
char \*s1, \*s2;  
if ( !\*str2 )  
return((char \*)str1);  
while (\*cp)  
{  
s1 = cp;  
s2 = (char \*) str2;  
while ( \*s1 && \*s2 && !(\*s1-\*s2) )  
s1++, s2++;  
if (!\*s2)  
return(cp);  
cp++;  
}  
return(NULL);  
}

218.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;  
答：  
假设线性表的双向链表存储结构  
typedef struct DulNode{  
struct DulNode \*prior; //前驱指针  
ElemType data; //数据  
struct DulNode \*next; //后继指针  
}DulNode,\*DuLinkList;  
删除操作  
Status ListDelete\_DuL(DuLinkList &L,int i,ElemType &e)  
{  
if(!(p=GetElemP\_DuL(L,i)))  
return ERROR;  
e=p->data;  
p->prior->next=p->next;  
p->next->prior=p->pror;  
free(p);  
return OK;  
}  
插入操作  
Status ListInsert\_DuL(DuLinkList &L,int i,ElemType &e)  
{  
if(!(p=GetElemP\_DuL(L,i)))  
return ERROR;  
if(!(s=(DuLinkList)malloc(sizeof(DuLNode))))  
return ERROR;

s->data=e;  
s->prior=p->prior;  
p->prior->next=s;  
s->next=p;  
p->prior=s;  
return OK;  
}

219.写一个函数，将其中的\t都转换成4个空格。  
答：  
该函数命名为convert,参数的意义为：  
\*strDest目的字符串,\*strSrc源字符串,length源字符串的长度  
函数实现为：  
char\* convert(char \*strDest, const char \*strSrc,int length)  
{  
char \* cp = strDest;  
int i=0;  
while(\*strSrc && i  
{  
if (\*strSrc==’\t’) //将\t转换成4个空格  
{  
for(int j=0;j<4;j++)  
\*cp++=’ ‘;  
}  
else //否则直接拷贝  
\*cp++=\*strSrc;  
strSrc++;  
i++;  
}  
return strDest;  
}

230.Windows程序的入口是哪里？写出Windows消息机制的流程。  
答：  
Windows程序的入口是WinMain函数  
消息机制：系统将会维护一个或多个消息队列，所有产生的消息都会被放入或是插入队列中。系统会在队列中取出每一条消息，根据消息的接收句柄而将该消息发送给拥有该窗口的程序的消息循环。每一个运行的程序都有自己的消息循环，在循环中得到属于自己的消息并根据接收窗口的句柄调用相应的窗口过程。而在没有消息时消息循环就将控制权交给系统。

231.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

232.C++里面是不是所有的动作都是main()引起的？如果不是，请举例。  
答：不是，比如中断引起的中断处理不是直接由main()引起的，而是由外部事件引起的。

233.C++里面如何声明const void f(void)函数为C程序中的库函数  
答：在该函数前添加extern “C”声明

234. 内联函数在编译时是否做参数类型检查  
答：做类型检查，因为内联函数就是在程序编译时，编译器将程序中出现的内联函数的调用表达式用内联函数的函数体来代替。  
235．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？  
答：IP是Internet Protocol的简称，是网络层的主要协议，作用是提供不可靠、无连接的数据报传送。TCP是Transmit Control Protocol（传输控制协议）的缩写，在运输层，TCP提供一种面向连接的，可靠的字节流服务；UDP是User Datagram Protocol（用户数据报协议）的缩写，在运输层，UDP提供不可靠的传输数据服务

236．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？  
答：交换机属于ＯＳＩ第二层即数据链路层设备。它根据ＭＡＣ地址寻址，通过站表选择路由，站表的建立和维护由交换机自动进行。路由器属于ＯＳＩ第三层即网络层设备，它根据ＩＰ地址进行寻址，通过路由表路由协议产生。交换机最大的好处是快速，路由器最大的好处是控制能力强。

237．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？  
答：一些变量在整个程序中都是可见的，它们称为全局变量。一些变量只能在一个函数中可知，称为局部变量。这就是他们的区别。  
在任何函数外面定义的变量就是全局变量，在函数内部定义的变量是局部变量，这是它们在程序中的实现过程。  
操作系统和编译器是根据程序运行的内存区域知道他们的，程序的全局数据放在所分配内存的全局数据区，程序的局部数据放在栈区。

238. 有两个文件a.txt,b.txt.a.txt中存储的是aaaaaa，b.txt中存储的是bbb。将两个文件合并成c.txt如果是a并b的话存储为abababaaa.要是b并a   的话就是bababaaaa.用c语言编程实现。

#include   “stdio.h”

void   fmerge(FILE   \*fa,FILE   \*fb,FILE   \*fc)

{

char   cha,chb;

cha=fgetc(fa);

chb=fgetc(fb);

while   ((cha!=EOF)&&(chb!=EOF))

{

fputc(cha,fc);

fputc(chb,fc);

cha=fgetc(fa);

chb=fgetc(fb);

}

while   (cha!=EOF)

{

fputc(cha,fc);

cha=fgetc(fa);

}

while   (chb!=EOF)

{

fputc(chb,fc);

chb=fgetc(fb);

}

}

int   main()

{

FILE   \*fa,\*fb,\*fc;

fa=fopen(“a.txt”,”r”);

fb=fopen(“b.txt”,”r”);

fc=fopen(“c.txt”,”w”);

fmerge(fa,fb,fc);

fclose(fa);

fclose(fb);

fclose(fc);

return   0;

}

239.C++:memset ,memcpy 和strcpy 的根本区别？  
#include “memory.h”  
memset用来对一段内存空间全部设置为某个字符，一般用在对定义的字符串进行初始化为‘ ‘或‘\0′；例:char a[100];memset(a, ‘\0′, sizeof(a));  
memcpy用来做内存拷贝，你可以拿它拷贝任何数据类型的对象，可以指定拷贝的数据长度；例：char a[100],b[50]; memcpy(b, a, sizeof(b));注意如用sizeof(a)，会造成b的内存地址溢出。

strcpy就只能拷贝字符串了，它遇到’\0′就结束拷贝；例：char a[100],b[50];strcpy(a,b);如用strcpy(b,a)，要注意a中的字符串长度（第一个‘\0′之前）是否超过50位，如超过，则会造成b的内存地址溢出。

strcpy  
原型：extern char \*strcpy(char \*dest,char \*src);  
用法：#include  
功能：把src所指由NULL结束的字符串复制到dest所指的数组中。  
说明：src和dest所指内存区域不可以重叠且dest必须有足够的空间来容纳src的字符串。  
返回指向dest的指针。  
memcpy  
原型：extern void \*memcpy(void \*dest, void \*src, unsigned int count);  
用法：#include  
功能：由src所指内存区域复制count个字节到dest所指内存区域。  
说明：src和dest所指内存区域不能重叠，函数返回指向dest的指针。  
memset  
原型：extern void \*memset(void \*buffer, char c, int count);  
用法：#include  
功能：把buffer所指内存区域的前count个字节设置成字符c。  
说明：返回指向buffer的指针。

240.ASSERT()是干什么用的

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。例如，变量n在程序中不应该为0，如果为0可能导致错误，你可以这样写程序：  
……  
ASSERT( n != 0);  
k = 10/ n;  
……  
ASSERT只有在Debug版本中才有效，如果编译为Release版本则被忽略。  
assert()的功能类似，它是ANSI C标准中规定的函数，它与ASSERT的一个重要区别是可以用在Release版本中。

241. 二分查找算法：  
1、递归方法实现：  
int BSearch(elemtype a[],elemtype x,int low,int high)  
/\*在下届为low，上界为high的数组a中折半查找数据元素x\*/  
{  
int mid;  
if(low>high) return -1;  
mid=(low+high)/2;  
if(x==a[mid]) return mid;  
if(x<a[mid]) return(BSearch(a,x,low,mid-1));  
else return(BSearch(a,x,mid+1,high));  
}

2、非递归方法实现：  
int BSearch(elemtype a[],keytype key,int n)  
{  
int low,high,mid;  
low=0;high=n-1;  
while(low<=high)  
{  
mid=(low+high)/2;  
if(a[mid].key==key) return mid;  
else if(a[mid].key<key) low=mid+1;  
else high=mid-1;  
}  
return -1;  
}

242，写出下面代码段的输出结果，并说明理由:

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;

1，  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。

243. 以下代码中的两个sizeof用法有问题吗？  
void UpperCase( char str[] )

{  
for( size\_t i=0; i<sizeof(str)/sizeof(str[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;

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，

244，下面程序输出结果是多少:

main()  
{  
int a[5]={1,2,3,4,5};  
int \*ptr=(int \*)(&a+1);

   printf(“%d,%d”,\*(a+1),\*(ptr-1));  
}  
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].

245，请问运行Test函数会有什么样的结果？  
void GetMemory(char \*p)  
{  
p = (char \*)malloc(100);  
}  
void Test(void)  
{  
char \*str = NULL;  
GetMemory(str);  
strcpy(str, “hello world”);  
printf(str);  
}

，请问运行Test函数会有什么样的结果？  
char \*GetMemory(void)  
{  
char p[] = “hello world”;  
return p;  
}  
void Test(void)  
{  
char \*str = NULL;  
str = GetMemory();  
printf(str);  
}

，请问运行Test函数会有什么样的结果？  
Void GetMemory2(char \*\*p, int num)  
{  
\*p = (char \*)malloc(num);  
}  
void Test(void)  
{  
char \*str = NULL;  
GetMemory(&str, 100);  
strcpy(str, “hello”);  
printf(str);  
}

，请问运行Test函数会有什么样的结果？  
void Test(void)  
{  
char \*str = (char \*) malloc(100);  
strcpy(str, “hello”);  
free(str);  
if(str != NULL)  
{  
strcpy(str, “world”);  
printf(str);  
}  
}