C语言面试题大汇总
4. static有什么用途？（请至少说明两种）
1.限制变量的作用域(DL:使其只在定义的当前文件中起作用，static是只能由与变量在同一个文件中定义的程序存取的全局变量。也就是说使全局变量成为文件的私有变量，以致其他文件不可以通过将它们定义为extern而存取这些变量。)
2.设置变量的存储域（DL：存储在最开始的静态存储空间里面）
7. 引用与指针有什么区别？
1) 引用必须被初始化，指针不必。
2) 引用初始化以后不能被改变，指针可以改变所指的对象。
2) 不存在指向空值的引用，但是存在指向空值的指针。

8. 描述实时系统的基本特性
在特定时间内完成特定的任务，实时性与可靠性
9. 全局变量和局部变量在内存中是否有区别？如果有，是什么区别？
全局变量储存在静态数据库，局部变量在堆栈
10. 什么是平衡二叉树？
左右子树都是平衡二叉树 且左右子树的深度差值的绝对值不大于1
11. 堆栈溢出一般是由什么原因导致的？
没有回收垃圾资源
12. 什么函数不能声明为虚函数？
constructor
13. 冒泡排序算法的时间复杂度是什么？
O(n^2)
14. 写出float x 与“ 零值 ”比较的if语句。
if(x>0.000001&&x<-0.000001)
16. Internet采用哪种网络协议？该协议的主要层次结构？
tcp/ip 应用层/传输层/网络层/数据链路层/物理层
17. Internet物理地址和IP地址转换采用什么协议？
ARP (Address Resolution Protocol)（地址解析協議）
18.IP地址的编码分为哪俩部分？
IP地址由两部分组成，网络号和主机号。不过是要和“ 子网掩码 ”按位与上之后才能区分哪些是网络位哪些是主机位。

2.用户输入M,N值，从1至N开始顺序循环数数，每数到M输出该数值，直至全部输出。写出C程序。
循环链表，用取余操作做
3.不能做switch()的参数类型是：
switch的参数不能为实型。

華為
1、局部变量能否和全局变量重名？
答：能，局部会屏蔽全局。要用全局变量，需要使用"::"
局部变量可以与全局变量同名，在函数内引用这个变量时，会用到同名的局部变量，而不会用到全局变量。对于有些编译器而言，在同一个函数内可以定义多个同名的局部变量，比如在两个循环体内都定义一个同名的局部变量，而那个局部变量的作用域就在那个循环体内
2、如何引用一个已经定义过的全局变量？
答：extern(在使用该变量的地方还要定义一次，extern只相当于声明，且只能在函数体外定义)/

static(在使用时不用定义，且作用域限制在当前源文件，且只能在函数体内重新赋值)
可以用引用头文件的方式（必须用static声明），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？
答：可以，在不同的C文件中以static形式来声明同名全局变量。
可以在不同的C文件中声明同名的全局变量，前提是其中只能有一个C文件中对此变量赋初值，此时连接不会出错
4、语句for(；1 ；)有什么问题？它是什么意思？
答：和while(1)相同。
5、do……while和while……do有什么区别？
答：前一个循环一遍再判断，后一个判断以后再循环
6、请写出下列代码的输出内容
```
#include<stdio.h>
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

1、static全局变量与普通的全局变量有什么区别？static局部变量和普通局部变量有什么区别？static函数与普通函数有什么区别？
全局变量(外部变量)的说明之前再冠以static 就构成了静态的全局变量。全局变量本身就是静态存储方式， 静态全局变量当然也是静态存储方式。 这两者在存储方式上并无不同。这两者的区别虽在于非静态全局变量的作用域是整个源程序， 当一个源程序由多个源文件组成时，非静态的全局变量在各个源文件中都是有效的。 而静态全局变量则限制了其作用域， 即只在定义该变量的源文件内有效， 在同一源程序的其它源文件中不能使用它。
由于静态全局变量的作用域局限于一个源文件内，只能为该源文件内的函数公用， 因此可以避免在其它源文件中引起错误。
从以上分析可以看出， 把局部变量改变为静态变量后是改变了它的存储方式即改变了它的生存期（静态局部变量在程序运行结束释放空间，而普通静态局部变量在函数退出时释放空间）。
把全局变量改变为静态变量后是改变了它的作用域， 限制了它的使用范围。
static函数与普通函数作用域不同。仅在本文件。只在当前源文件中使用的函数应该说明为内部函数(static)，内部函数应该在当前源文件中说明和定义。对于可在当前源文件以外使用的函数，应该在一个头文件中说明，要使用这些函数的源文件要包含这个头文件
（用static声明的即内部函数，内部函数指只能被本文件的其他函数所调用的函数，
内部函数在C++实际上可以通过类名修饰符访问
其他均为外部函数）

static全局变量与普通的全局变量有什么区别：static全局变量只初使化一次，防止在其他文件单元中被引用;
static局部变量和普通局部变量有什么区别：static局部变量只被初始化一次，下一次依据上一次结果值；

static函数与普通函数有什么区别：static函数在内存中只有一份，普通函数在每个被调用中维持一份拷贝
2、程序的局部变量存在于（堆栈）中，全局变量存在于（静态区）中，动态申请数据存在于（堆）中。
3、设有以下说明和定义：

```
typedef union {long i; int k[5]; char c;} DATE;
struct data { int cat; DATE cow; double dog;} too;
DATE max;
```

则语句 printf("%d",sizeof(struct date)+sizeof(max));的执行结果是：___52____
答：DATE是一个union, 变量公用空间. 里面最大的变量类型是int[5], 占用20个字节. 所以它的大小是20
data是一个struct, 每个变量分开占用空间. 依次为int4 + DATE20 + double8 = 32.
所以结果是 20 + 32 = 52.
当然...在某些16位编辑器下, int可能是2字节,那么结果是 int2 + DATE10 + double8 = 20
4、队列和栈有什么区别？
队列先进先出，栈后进先出
5、写出下列代码的输出内容

```
#include<stdio.h>
int inc(int a)
{
return(++a);
}
int multi(int*a,int*b,int*c)
{
return(*c=*a**b);
}
typedef int(FUNC1)(int in);
typedef int(FUNC2) (int*,int*,int*);

void show(FUNC2 fun,int arg1, int*arg2)
{
INCp=&inc;
int temp =p(arg1);
fun(&temp,&arg1, arg2);
printf("%d\n",*arg2);
}

main()
{
int a;
show(multi,10,&a);
return 0;
}
```

答：110
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 ”

```
1、#include"string.h"
2、main()
```

```
3、{
4、 char*src="hello,world";
5、 char* dest=NULL;
6、 int len=strlen(src);
7、 dest=(char*)malloc(len);
8、 char* d=dest;
9、 char* s=src[len];
10、 while(len--!=0)
11、 d++=s--;
12、 printf("%s",dest);
13、 return 0;
14、}
```

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 <stdio.h>
#include <string.h>
main()
{
char str[]="hello,world";
int len=strlen(str);
char t;
for(int i=0; i<len/2; i++)
{
t=str[i];
str[i]=str[len-i-1]; str[len-i-1]=t;
}
printf("%s",str);
return 0;
}
```

1.-1,2,7,28,,126请问28和126中间那个数是什么？为什么？

第一题的答案应该是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

2.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。

3.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 <stdlib.h>
#include <stdio.h>
int main(void)
{
long l;
char *str = "98765432";

l = atol(lstr);
printf("string = %s integer = %ld\n", str, l);
return(0);
}
```

2.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

3.直接链接两个信令点的一组链路称作什么?
PPP点到点连接

4.接入网用的是什么接口?
DL:接入网在接入这些网络时，一般采用E1、V.24、V.35、2B1Q“U”接口，其余类型的接口使用较少。

5.voip都用了那些协议?
VoIP使用IETF会话发起协议（SIP）和实时传输协议（RTP）提交呼叫信令和语音消息
与VoIP相关的网络技术协议很多，常见的有控制实时数据流应用在IP网络传输的RTP(实时传输协议)和RTCP(实时传输控制协议)；
有保证网络QoS质量服务的RSVP(资源预留协议)和IP different Service等,
还有传统语音数字化编码的一系列协议如G.711、G.728、G.723、G.729等等。
但目前VoIP技术最常用的话音建立和控制信令是H.323和SIP(会话初始协议)。

6.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

7.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

8.enum string
{
x1,
x2,
x3=10,
x4,
x5,
}x;
问x= 0x801005，0x8010f4 ;

9.unsigned char *p1;
unsigned long *p2;
p1=(unsigned char *)0x801000;
p2=(unsigned long *)0x810000;
请问p1+5= 0x801005;
p2+5= 0x810014(加5*4=20字节，16进制为0x14);

三.选择题:
1.Ethternet链接到Internet用到以下那个协议?
A.HDLC;B.ARP;C.UDP;D.TCP;E.ID
2.属于网络层协议的是:
A.TCP;B.IP;C.ICMP;D.X.25
3.Windows消息调度机制是:
A.指令队列;B.指令堆栈;C.消息队列;D.消息堆栈;
4.unsigned short hash(unsigned short key)
{
return (key>>)%256
}
请问hash(16),hash(256)的值分别是:
A.1.16;B.8.32;C.4.16;D.1.32

四.找错题:
1.请问下面程序有什么错误?
int a[60][250][1000],i,j,k;
for(k=0;k<=1000;k++)
for(j=0;j<250;j++)
for(i=0;i<60;i++)
a[i][j][k]=0;
把循环语句内外换一下(DL：换一下好像还是有错)

2.#define Max_CB 500
void LmiQueryCSmd(Struct MSgCB * pmsg)
{
unsigned char ucCmdNum;
.....

for(ucCmdNum=0;ucCmdNum<Max_CB;ucCmdNum++)
{
.....;

```
}
```
死循环

3.以下是求一个数的平方的程序,请找出错误:
```
#define SQUARE(a)((a)*(a))
int a=5;
int b;
b=SQUARE(a++);
```

4.typedef unsigned char BYTE
```
int examply_fun(BYTE gt_len; BYTE *gt_code)
{
BYTE *gt_buf;
gt_buf=(BYTE *)MALLOC(Max_GT_Length);
.....
if(gt_len>Max_GT_Length)
{
return GT_Length_ERROR;
}
.....
}
```
五.问答题:
1.IP Phone的原理是什么?
IPV6
2.TCP/IP通信建立的过程怎样，端口有什么作用？
三次握手，确定是哪个应用程序使用该协议
3.1号信令和7号信令有什么区别，我国某前广泛使用的是那一种？
七号信令网是电话网、智能网以及各种新业务的神经和支撑网，是通信网建设维护的重要部分。根据我国七号信令技术体制要求，我国七号信令网最终采用三级准直联结构方式。
1号信令利用TS16传送时。每个TS16负责传送两个话路的线路信令，TS16和话路有着固定的一一对应关系。
而7号信令利用TS16来传送时，只是将组成信令单元的若干个8位位组，依次插入TS16，TS16并不知道传送的内容，
即信令和话路没有固定关系，只不过利用TS16作为传送信令的载体，时传送信令消息的数据链路，
因此，选用哪个时隙做数据链路均可。 --- 这也是随路信令和公共信道信令的一个本质区别。

4.列举5种以上的电话新业务？

微软亚洲技术中心的面试题！！！
1. 进程和线程的差别。
线程是指进程内的一个执行单元,也是进程内的可调度实体.
与进程的区别:
(1)调度：线程作为调度和分配的基本单位，进程作为拥有资源的基本单位
(2)并发性：不仅进程之间可以并发执行，同一个进程的多个线程之间也可并发执行
(3)拥有资源：进程是拥有资源的一个独立单位，线程不拥有系统资源，但可以访问隶属于进程的资源.
(4)系统开销：在创建或撤消进程时，由于系统都要为之分配和回收资源，导致系统的开销明显大于创建或撤消线程时的开销。
2.测试方法

人工测试：个人复查、抽查和会审
机器测试：黑盒测试和白盒测试

2．Heap与stack的差别。
Heap是堆，stack是栈。
Stack的空间由操作系统自动分配/释放，Heap上的空间手动分配/释放。
Stack空间有限，Heap是很大的自由存储区
C中的malloc函数分配的内存空间即在堆上,C++中对应的是new操作符。
程序在编译期对变量和函数分配内存都在栈上进行,且程序运行过程中函数调用时参数的传递也在栈上进行
3．Windows下的内存是如何管理的？
4．介绍.Net和.Net的安全性。
5．客户端如何访问.Net组件实现Web Service？
6．C/C++编译器中虚表是如何完成的？
7．谈谈COM的线程模型。然后讨论进程内/外组件的差别。
8．谈谈IA32下的分页机制
小页(4K)两级分页模式，大页(4M)一级
9．给两个变量，如何找出一个带环单链表中是什么地方出现环的？
一个递增一，一个递增二，他们指向同一个接点时就是环出现的地方
10．在IA32中一共有多少种办法从用户态跳到内核态？
通过调用门，从ring3到ring0，中断从ring3到ring0，进入vm86等等
11．如果只想让程序有一个实例运行，不能运行两个。像winamp一样，只能开一个窗口，怎样实现？
用内存映射或全局原子（互斥变量）、查找窗口句柄..
FindWindow，互斥，写标志到文件或注册表,共享内存。.
12．如何截取键盘的响应，让所有的‘a’变成‘b’？
键盘钩子SetWindowsHookEx
13．Apartment在COM中有什么用？为什么要引入？
14．存储过程是什么？有什么用？有什么优点？
我的理解就是一堆sql的集合，可以建立非常复杂的查询，编译运行，所以运行一次后，以后再运行速度比单独执行SQL快很多
15．Template有什么特点？什么时候用？
16．谈谈Windows DNA结构的特点和优点。

网络编程中设计并发服务器，使用多进程 与 多线程 ，请问有什么区别？
1，进程：子进程是父进程的复制品。子进程获得父进程数据空间、堆和栈的复制品。
2，线程：相对与进程而言，线程是一个更加接近与执行体的概念，它可以与同进程的其他线程共享数据，但拥有自己的栈空间，拥有独立的执行序列。
两者都可以提高程序的并发度，提高程序运行效率和响应时间。
线程和进程在使用上各有优缺点：线程执行开销小，但不利于资源管理和保护；而进程正相反。同时，线程适合于在SMP机器上运行，而进程则可以跨机器迁移。

思科
1.用宏定义写出swap（x，y）
```
#define swap(x, y)\
x = x + y;\
y = x - y;\
x = x - y;
```

2.数组a[N]，存放了1至N-1个数，其中某个数重复一次。写一个函数，找出被重复的数字
.时间复杂度必须为o（N）函数原型：
注意a[N]中存放的是1-〉N-1个数
```
int do_dup(int a[],int N)
{
   int temp[N]={0};
   for(int i=0;i<N;i++)
     if(temp[a[i]]!=0)
        return i;
}
```

3 一语句实现x是否为2的若干次幂的判断
```
int i = 512;
cout << boolalpha << ((i & (i - 1)) ? false : true) << endl;
```
按位与运算符a&b，对b中为1的位如果a中也为1则保留，a中其余位全部置0，剩下的a即为结果
也可以理解为保留a中与b中位1对应的位，其余置0
其余按位运算符类似，将a与b按位做相应运算，所得值即结果

4.unsigned int intvert(unsigned int x,int p,int n)实现对x的进行转换,p为起始转化位,n为需要转换的长度,假设起始点在右边.如x=0b0001 0001,p=4,n=3转换后x=0b0110 0001
```
unsigned int intvert(unsigned int x,int p,int n){
unsigned int _t = 0;
unsigned int _a = 1;
for(int i = 0; i < n; ++i){
_t |= _a;
_a = _a << 1;
}
//_t包含n个1
_t = _t << p;//将n个1左移p位
x ^= _t;
return x;
}
```

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

```
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
```

```
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。

12. 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 )//sizeof(str)=4，为指针大小
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，因此其大小为数组大小

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。

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。

```
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].

1.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。

1、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;------语法上并没有限制宏后面必须没有分号，宏只是简单的字符替换，这里是因为使用Min的地方通常会在后面加分号
```

2、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(;;)
```
3、关键字static的作用是什么？
定义静态变量

4、关键字const有什么含意？
表示常量不可以修改的变量。
5、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。
volatile关键字是一种类型修饰符，用它声明的类型变量表示可以被某些编译器未知的因素更改，
比如：操作系统、硬件或者其它线程等。遇到这个关键字声明的变量，
编译器对访问该变量的代码就不再进行优化，从而可以提供对特殊地址的稳定访问。
当要求使用volatile 声明的变量的值的时候，系统总是重新从它所在的内存读取数据，
即使它前面的指令刚刚从该处读取过数据。而且读取的数据立刻被保存。
volatile 指出 i是随时可能发生变化的，每次使用它的时候必须从i的地址中读取，
因而编译器生成的汇编代码会重新从i的地址读取数据放在b中。
而优化做法是，由于编译器发现两次从i读数据的代码之间的代码没有对i进行过操作，
它会自动把上次读的数据放在b中。而不是重新从i里面读。
这样以来，如果i是一个寄存器变量或者表示一个端口数据就容易出错，
所以说volatile可以保证对特殊地址的稳定访问。
vc6调试模式下没有优化，关键字的作用看不出来，但发行模式则会起作用，故对于多线程共享的变量最好用volatile修饰

int (*s[10])(int) 表示的是什么啊
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。

1.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
2.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;

3.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

4.#include <stdio.h>

```
#include <stdlib.h>
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

5.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

6.列举几种进程的同步机制，并比较其优缺点。
在主流的Linux内核中包含了几乎所有现代的操作系统具有的同步机制，这些同步机制包括：原子操作、信号量（semaphore）、读写信号量（rw_semaphore）、spinlock、BKL(Big Kernel Lock)、rwlock、brlock（只包含在2.4内核中）、RCU（只包含在2.6内核中）和seqlock（只包含在2.6内核中）。

7.进程之间通信的途径
管道(pipe)和有名管道(named pipe)、消息队列(mesage queue)、信号(signal)、
信号量(semaphore)、共享存储区(shared memory)、套接口(socket)等

11.进程死锁的原因
资源竞争及进程推进顺序非法
12.死锁的4个必要条件
互斥、请求保持、不可剥夺、环路
13.死锁的处理
鸵鸟策略、预防策略、避免策略、检测与解除死锁
15. 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
8.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
9.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
10.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

12.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

1：(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

2：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

1，要对绝对地址0x100000赋值，我们可以用
(unsigned int*)0x100000 = 1234;
那么要是想让程序跳转到绝对地址是0x100000去执行，应该怎么做？
*((void (*)( ))0x100000) ( );
首先要将0x100000强制转换成函数指针,即:
(void (*)())0x100000
然后再调用它:
*((void (*)())0x100000)();
用typedef可以看得更直观些:
typedef void(*)() voidFuncPtr;
*((voidFuncPtr)0x100000)();

2，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]))

面试题: 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

```
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＝0x80,输出的是int，最高位为1，是负数，所以它的值就是0x00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。

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);
getchar();
}
```

问输出结果是什么？希望大家能说说原因，先谢谢了
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，只不过。。。。。。。楼上都说过了，最好别这么干。
DL:在C++中使用指针的引用也可以在其他函数内部申请空间，格式为int *&p,然后用法一样，c不行

char a[10],strlen(a)为什么等于15？运行的结果

```
#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。

```
char (*str)[20];/*str是一个数组指针，即指向数组的指针．*/
char *str[20];/*str是一个指针数组，其元素为指针型数据．*/
```

1)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
```
2)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 s
tatic上？

```
// 帮忙分析一下
#include<iostream.h>
#include <string.h>
#include <malloc.h>
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
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

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；
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 &&(*c<a || *c<b) || (a<0 && b<0 &&(*c>a || *c>b)));
}
```

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=0x99;
cout << s.a <<endl <<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0x99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

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：

1. 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 /*空域*/
unsigned b:4 /*从下一单元开始存放*/
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。

2. 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。

3. 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 /*该2位不能使用*/
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。

二、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
```

改错：

```
#include <stdio.h>

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 <stdio.h>
int main(void) {
int **p, *q;
int arr[100];
q = arr;
p = &q;
return 0;
}
```

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;
}
```
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.

```
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。

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。

```
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

1）读文件file1.txt的内容（例如）：
12
34
56
输出到file2.txt：
56
34
12
（逆序）
2）输出和为一个给定整数的所有组合
例如n=5
5=1+4；5=2+3（相加的数不能重复）
则输出
1，4；2，3。
望高手赐教！！

第一题,注意可增长数组的应用.
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
int MAX = 10;
int *a = (int *)malloc(MAX * sizeof(int));
int *b;

FILE *fp1;
FILE *fp2;

fp1 = fopen("a.txt","r");
if(fp1 == NULL)

```
{printf("error1");
exit(-1);
}

fp2 = fopen("b.txt","w");
if(fp2 == NULL)
{printf("error2");
exit(-1);
}

int i = 0;
int j = 0;

while(fscanf(fp1,"%d",&a[i]) != EOF)
{
i++;
j++;
if(i >= MAX)
{
MAX = 2 * MAX;
b = (int*)realloc(a,MAX * sizeof(int));
if(b == NULL)
{
printf("error3");
exit(-1);
}
a = b;
}
}

for(;--j >= 0;)
fprintf(fp2,"%d\n",a[j]);

fclose(fp1);
fclose(fp2);

return 0;


}
```

第二题.

```
#include <stdio.h>

int main(void)
{
unsigned long int i,j,k;

printf("please input the number\n");
scanf("%d",&i);
```

```
if( i % 2 == 0)
j = i / 2;
else
j = i / 2 + 1;

printf("The result is \n");
for(k = 0; k < j; k++)
printf("%d = %d + %d\n",i,k,i - k);
return 0;
}

#include <stdio.h>
void main()
{
unsigned long int a,i=1;
scanf("%d",&a);
if(a%2==0)
{
for(i=1;i<a/2;i++)
printf("%d",a,a-i);
}
else
for(i=1;i<=a/2;i++)
printf(" %d, %d",i,a-i);
}
```

兄弟,这样的题目若是做不出来实在是有些不应该, 给你一个递规反向输出字符串的例子,
可谓是反序的经典例程.

```
void inverse(char *p)
{
if( *p = = '\0' )
return;
inverse( p+1 );
printf( "%c", *p );
}

int main(int argc, char *argv[])
{
inverse("abc\0");

return 0;
}
```

借签了楼上的“ 递规反向输出 ”

```
#include <stdio.h>
void test(FILE *fread, FILE *fwrite)
{
char buf[1024] = {0};
if (!fgets(buf, sizeof(buf), fread))
```

```
return;
test( fread, fwrite );
fputs(buf, fwrite);
}
int main(int argc, char *argv[])
{
FILE *fr = NULL;
FILE *fw = NULL;
fr = fopen("data", "rb");
fw = fopen("dataout", "wb");
test(fr, fw);
fclose(fr);
fclose(fw);
return 0;
}
```

在对齐为4的情况下

```
struct BBB
{
long num ;
char *name;
short int data;
char ha;
short ba[5];
}*p;
p=0x1000000;
p+0x200=____;
(Ulong)p+0x200=____;
(char*)p+0x200=____;
```

希望各位达人给出答案和原因，谢谢拉

解答：假设在32位CPU上，

```
sizeof(long) = 4 bytes
sizeof(char *) = 4 bytes
sizeof(short int) = sizeof(short) = 2 bytes
sizeof(char) = 1 bytes
```

由于是4字节对齐，

```
sizeof(struct BBB) = sizeof(*p)
= 4 + 4 + 2 + 1 + 1/*补齐*/ + 2*5 + 2/*补齐*/ = 24 bytes (经Dev-C++验证)

p=0x1000000;
p+0x200=____;
= 0x1000000 + 0x200*24

(Ulong)p+0x200=____;
= 0x1000000 + 0x200

(char*)p+0x200=____;
= 0x1000000 + 0x200*4
```

你可以参考一下指针运算的细节

写一段程序，找出数组中第k大小的数，输出数所在的位置。例如{2，4，3，4，7}中，第一大的数是7，位置在4。第二大、第三大的数都是4，位置在1、3随便输出哪一个均可。函数接口为：int find_orderk(const int* narry,const int n,const int k)
要求算法复杂度不能是O(n^2)
谢谢！
可以先用快速排序进行排序，其中用另外一个进行地址查找
代码如下，在VC++6.0运行通过。给分吧^-^

```
//快速排序

#include<iostream>

usingnamespacestd;

intPartition (int*L,intlow,int high)
{
inttemp = L[low];
intpt = L[low];

while (low < high)
{
while (low < high && L[high] >= pt)
--high;
L[low] = L[high];
while (low < high && L[low] <= pt)
++low;
L[low] = temp;
}
L[low] = temp;

returnlow;
}

voidQSort (int*L,intlow,int high)
{
if (low < high)
{
intpl = Partition (L,low,high);

QSort (L,low,pl - 1);
QSort (L,pl + 1,high);
}
}

intmain ()
{
intnarry[100],addr[100];
```

```
intsum = 1,t;

cout << "Input number:" << endl;
cin >> t;

while (t != -1)
{
narry[sum] = t;
addr[sum - 1] = t;
sum++;

cin >> t;
}

sum -= 1;
QSort (narry,1,sum);

for (int i = 1; i <= sum;i++)
cout << narry[i] << '\t';
cout << endl;

intk;
cout << "Please input place you want:" << endl;
cin >> k;

intaa = 1;
intkk = 0;
for (;;)
{
if (aa == k)
break;
if (narry[kk] != narry[kk + 1])
{
aa += 1;
kk++;
}

}

cout << "The NO." << k << "number is:" << narry[sum - kk] << endl;
cout << "And it's place is:" ;
for (i = 0;i < sum;i++)
{
if (addr[i] == narry[sum - kk])
cout << i << '\t';
}


return0;
}
```

1、找错

```
Void test1()
{
char string[10];
char* str1="0123456789";
strcpy(string, str1);// 溢出，应该包括一个存放'\0'的字符string[11]
}


Void test2()
{
char string[10], str1[10];
for(I=0; I<10;I++)
{
str1[i] ='a';
}
strcpy(string, str1);// I，i没有声明。
}

Void test3(char* str1)
{
char string[10];
if(strlen(str1)<=10)// 改成<10,字符溢出，将strlen改为sizeof也可以
{
strcpy(string, str1);
}
}
```

2.

```
void g(int**);
int main()
{
int line[10],i;
int *p=line; //p是地址的地址
for (i=0;i<10;i++)
{
*p=i;
g(&p);//数组对应的值加1
}
for(i=0;i<10;i++)
printf("%d\n",line[i]);
return 0;
}

void g(int**p)
{
(**p)++;
(*p)++;// 无效
}
```

输出：
1
2
3
4
5
6
7
8
9
10
3. 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));
}
}
```

// static会保存上次结果，记住这一点，剩下的自己写
输出：8,10,12,14,16,

4.

```
int func(int a)
{
int b;
switch(a)
{
case 1: 30;
case 2: 20;
case 3: 16;
default: 0
}
return b;
}
```

则func(1)=?

// b定义后就没有赋值。

5:
int a[3];
a[0]=0; a[1]=1; a[2]=2;
int *p, *q;
p=a;
q=&a[2];
则a[q-p]=a[2]
解释：指针一次移动一个int但计数为1

今天早上的面试题9道，比较难，向牛人请教，国内的一牛公司，坐落在北京北四环某大厦：
1、线形表a、b为两个有序升序的线形表，编写一程序，使两个有序线形表合并成一个有序升序线形表h；
答案在 请化大学 严锐敏《数据结构第二版》第二章例题，数据结构当中，这个叫做：两路归并排序
```
Linklist *unio(Linklist *p,Linklist *q){
linklist *R,*pa,*qa,*ra;
pa=p;
qa=q;
R=ra=p;
while(pa->next!=NULL&&qa->next!=NULL){
if(pa->data>qa->data){
ra->next=qa;
qa=qa->next;
}
else{
ra->next=pa;
pa=pa->next;
}
}
if(pa->next!=NULL)
ra->next=pa;
if(qa->next!=NULL)
ra->next==qa;
return R;
}
```
2、运用四色定理，为N个局域举行配色，颜色为1、2、3、4四种，另有数组adj[][N]，如adj[i][j]=1则表示i区域与j区域相邻，数组color[N]，如color[i]=1,表示i区域的颜色为1号颜色。
四色填充
3、用递归算法判断数组a[N]是否为一个递增数组。
递归的方法，记录当前最大的，并且判断当前的是否比这个还大，大则继续，否则返回false结束：
```
bool fun( int a[], int n )
{
if( n= =1 )
return true;
if( n= =2 )
```

```
return a[n-1] >= a[n-2];
return fun( a,n-1) && ( a[n-1] >= a[n-2] );
}
```

4、编写算法，从10亿个浮点数当中，选出其中最大的10000个。
用外部排序，在《数据结构》书上有
《计算方法导论》在找到第n大的数的算法上加工
5、编写一unix程序，防止僵尸进程的出现.

Top

回复人：free131(白日?做梦!) ( ) 信誉：100 2006-4-17 10:17:34 得分:0

?
同学的4道面试题，应聘的职位是搜索引擎工程师，后两道超级难，（希望大家多给一些算发）
1.给两个数组和他们的大小，还有一动态开辟的内存，求交集，把交集放到动态内存dong tai，并且返回交集个数

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

2.单连表的建立，把'a'--'z'26个字母插入到连表中，并且倒叙，还要打印！
方法1：

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

void main(void)
{ char c;

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

p.next=NULL;
}
}
```

方法2：

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

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

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

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

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

print(head);
```

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

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

找出错误！！

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

1. 简述一个Linux驱动程序的主要流程与功能。

2. 请列举一个软件中时间换空间或者空间换时间的例子。

```
void swap(int a,int b)
{
int c; c=a;a=b;b=a;
}
```

--->空优

```
void swap(int a,int b)
{
a=a+b;b=a-b;a=a-b;
}
```

6. 请问一下程序将输出什么结果？

```
char *RetMenory(void)
{
char p[] = “ hellow world ”;
return p;
}
void Test(void)
{
char *str = NULL;
str = RetMemory();
printf(str);
}
```

RetMenory执行完毕，p资源被回收，指向未知地址。返回地址，str的内容应是不可预测的
, 打印的应该是str的地址

写一个函数,它的原形是int continumax(char *outputstr,char *intputstr)
功能：
在字符串中找出连续最长的数字串，并把这个串的长度返回，并把这个最长数字串付给其
中一个函数参数outputstr所指内存。例如："abcd12345ed125ss123456789"的首地址传给
intputstr后，函数将返回
9，outputstr所指的值为123456789

```
int continumax(char *outputstr, char *inputstr)
{
char *in = inputstr, *out = outputstr, *temp, *final;
int count = 0, maxlen = 0;

while( *in != '\0' )
{
if( *in > 47 && *in < 58 )
{
for(temp = in; *in > 47 && *in < 58 ; in++ )
count++;
}
else
in++;
```

```
if( maxlen < count )
{
maxlen = count;
count = 0;
final = temp;
}
}
for(int i = 0; i < maxlen; i++)
{
*out = *final;
out++;
final++;
}
*out = '\0';
return maxlen;
}
```

不用库函数,用C语言实现将一整型数字转化为字符串
方法1：

```
int getlen(char *s){
int n;
for(n = 0; *s != '\0'; s++)
n++;
return n;
}
void reverse(char s[])
{
int c,i,j;
for(i = 0,j = getlen(s) - 1; i < j; i++,j--){
c = s[i];
s[i] = s[j];
s[j] = c;
}
}
void itoa(int n,char s[])
{
int i,sign;
if((sign = n) < 0)
n = -n;
i = 0;
do{/*以反序生成数字*/
s[i++] = n%10 + '0';/*get next number*/
}while((n /= 10) > 0);/*delete the number*/

if(sign < 0)
s[i++] = '-';

s[i] = '\0';
reverse(s);
}
```

方法2:

```
#include <iostream>
using namespace std;

void itochar(int num);

void itochar(int num)
{
int i = 0;
int j ;
char stra[10];
char strb[10];
while ( num )
{
stra[i++]=num%10+48;
num=num/10;
}
stra[i] = '\0';
for( j=0; j < i; j++)
{
strb[j] = stra[i-j-1];
}
strb[j] = '\0';
cout<<strb<<endl;

}
int main()
{
int num;
cin>>num;
itochar(num);
return 0;
}
```

前几天面试，有一题想不明白,请教大家！

```
typedef struct
{
int a:2;
int b:2;
int c:1;
}test;

test t;
t.a = 1;
t.b = 3;
t.c = 1;

printf("%d",t.a);
printf("%d",t.b);
printf("%d",t.c);
```

谢谢!
t.a为01,输出就是1
t.b为11，输出就是－1
t.c为1，输出也是-1
3个都是有符号数int嘛。
这是位扩展问题
01
11
1
编译器进行符号扩展

求组合数： 求n个数（1....n）中k个数的组合....
如：combination(5,3)
要求输出：543，542，541，532，531，521，432，431，421，321，
#include<stdio.h>

```
int pop(int *);
int push(int );
void combination(int ,int );

int stack[3]={0};
top=-1;

int main()
{
int n,m;
printf("Input two numbers:\n");
while( (2!=scanf("%d%*c%d",&n,&m)) )
{
fflush(stdin);
printf("Input error! Again:\n");
}
combination(n,m);
printf("\n");
}
void combination(int m,int n)
{
int temp=m;
push(temp);
while(1)
{
if(1==temp)
{
if(pop(&temp)&&stack[0]==n) //当栈底元素弹出&&为可能取的最小值，循环退出
break;
}
else if( push(--temp))
{
```

```
printf("%d%d%d ",stack[0],stack[1],stack[2]);// § &auml; ¨ ì ¤ @?
pop(&temp);
}
}
}
int push(int i)
{
stack[++top]=i;
if(top<2)
return 0;
else
return 1;
}
int pop(int *i)
{
*i=stack[top--];
if(top>=0)
return 0;
else
return 1;
}
```

1、用指针的方法，将字符串“ ABCD1234efgh ”前后对调显示

```
#include <stdio.h>
#include <string.h>
#include <dos.h>
int main()
{
char str[] = "ABCD1234efgh";
int length = strlen(str);
char * p1 = str;
char * p2 = str + length - 1;
while(p1 < p2)
{
char c = *p1;
*p1 = *p2;
*p2 = c;
++p1;
--p2;
}
printf("str now is %s\n",str);
system("pause");
return 0;
}
```

2、有一分数序列：1/2,1/4,1/6,1/8……，用函数调用的方法，求此数列前20项的和

```
#include <stdio.h>
double getValue()
{
double result = 0;
int i = 2;
```

```
while(i < 42)
{
result += 1.0 / i;//一定要使用1.0做除数，不能用1，否则结果将自动转化成整数，即
0.000000
i += 2;
}
return result;
}
int main()
{
printf("result is %f\n", getValue());
system("pause");
return 0;
}
```

Top

回复人：free131(白日?做梦!) ( ) 信誉：100 2006-4-17 10:18:33 得分:0

?
有一个数组a[1000]存放0--1000;要求每隔二个数删掉一个数，到末尾时循环至开头继续进行，求最后一个被删掉的数的原始下标位置。
以7个数为例：
{0,1,2,3,4,5,6,7} 0-->1-->2（删除）-->3-->4-->5(删除)-->6-->7-->0（删除），如此循环直到最后一个数被删除。
方法1：数组

```
#include <iostream>
using namespace std;
#define null 1000

int main()
{
int arr[1000];
for (int i=0;i<1000;++i)
arr[i]=i;
int j=0;
int count=0;
while(count<999)
{
while(arr[j%1000]==null)
j=(++j)%1000;
j=(++j)%1000;
while(arr[j%1000]==null)
j=(++j)%1000;
j=(++j)%1000;
while(arr[j%1000]==null)
j=(++j)%1000;
```

```
arr[j]=null;
++count;
}
while(arr[j]==null)
j=(++j)%1000;

cout<<j<<endl;
return 0;
}
```

方法2：链表

```
#include<iostream>
using namespace std;
#define null 0
struct node
{
int data;
node* next;
};
int main()
{
node* head=new node;
head->data=0;
head->next=null;
node* p=head;
for(int i=1;i<1000;i++)
{
node* tmp=new node;
tmp->data=i;
tmp->next=null;
head->next=tmp;
head=head->next;
}
head->next=p;
while(p!=p->next)
{
p->next->next=p->next->next->next;
p=p->next->next;
}
cout<<p->data;
return 0;
}
```

方法3：通用算法

```
#include <stdio.h>
#define MAXLINE 1000 //元素个数
/*
MAXLINE 元素个数
a[] 元素数组
R[] 指针场
suffix 下标
index 返回最后的下标序号
values 返回最后的下标对应的值
```

```
start 从第几个开始
K 间隔
*/
int find_n(int a[],int R[],int K,int& index,int& values,int s=0) {
int suffix;
int front_node,current_node;
suffix=0;
if(s==0) {
current_node=0;
front_node=MAXLINE-1;
}
else {
current_node=s;
front_node=s-1;
}
while(R[front_node]!=front_node) {
printf("%d\n",a[current_node]);
R[front_node]=R[current_node];
if(K==1) {
current_node=R[front_node];
continue;
}
for(int i=0;i<K;i++){
front_node=R[front_node];
}
current_node=R[front_node];
}
index=front_node;
values=a[front_node];

return 0;
}
int main(void) {
int a[MAXLINE],R[MAXLINE],suffix,index,values,start,i,K;
suffix=index=values=start=0;
K=2;

for(i=0;i<MAXLINE;i++) {
a[i]=i;
R[i]=i+1;
}
R[i-1]=0;
find_n(a,R,K,index,values,2);
printf("the value is %d,%d\n",index,values);
return 0;
}
```

试题：

```
void test2()
{
```

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

解答：对试题2，如果面试者指出字符数组str1不能在数组内结束可以给3分；如果面试者指出strcpy(string, str1)调用使得从str1内存起复制到string内存起所复制的字节数具有不确定性可以给7分，在此基础上指出库函数strcpy工作方式的给10分；

str1不能在数组内结束:因为str1的存储为：{a,a,a,a,a,a,a,a,a,a},没有'\0'(字符串结束符)，所以不能结束

strcpy(char *s1,char *s2)他的工作原理是，扫描s2指向的内存，逐个字符付到s1所指向的内存，直到碰到'\0',因为str1结尾没有'\0'，所以具有不确定性，不知道他后面还会付什么东东。

正确应如下

```
void test2()
{
char string[10], str1[10];
int i;
for(i=0; i<9; i++)
{
str1[i] = 'a'+i; //把abcdefghi赋值给字符数组
}
str[i]='\0';//加上结束符
strcpy( string, str1 );
}
```

第二个code题是实现strcmp

```
int StrCmp(const char *str1, const char *str2)
```

做是做对了，没有抄搞，比较乱

```
int StrCmp(const char *str1, const char *str2)
{
assert(str1 && srt2);
while (*str1 && *str2 && *str1 == *str2) {
str1++, str2++;
}
if (*str1 && *str2)
return (*str1-*str2);
elseif (*str1 && *str2==0)
return 1;
elseif (*str1 = = 0 && *str2)
return -1;
else
return 0;
}

int StrCmp(const char *str1, const char *str2)
{
```

```
//省略判断空指针(自己保证)
while(*str1 && *str1++ = = *str2++);
return *str1-*str2;
}
```

第三个code题是实现子串定位

```
int FindSubStr(const char *MainStr, const char *SubStr)
```

做是做对了，没有抄搞，比较乱

```
int MyStrstr(const char* MainStr, const char* SubStr)
{
const char *p;
const char *q;
const char * u = MainStr;

//assert((MainStr!=NULL)&&( SubStr!=NULL));//用断言对输入进行判断
while(*MainStr) //内部进行递增
{
p = MainStr;
q = SubStr;
while(*q && *p && *p++ == *q++);
if(!*q )
{
return MainStr - u +1 ;//MainStr指向当前起始位，u指向
}
MainStr ++;
}
return -1;
}
```

分析：

```
int arr[] = {6,7,8,9,10};
int *ptr = arr;
*(ptr++)+=123;
printf( “ %d %d ” , *ptr, *(++ptr));
```

输出：8 8

过程：对于*(ptr++)+=123;先做加法6+123，然后++，指针指向7；对于printf(“ %d %d ” , *ptr, *(++ptr));从后往前执行，指针先++，指向8，然后输出8，紧接着再输出8

华为全套完整试题

高级题

6、已知一个单向链表的头，请写出删除其某一个结点的算法，要求，先找到此结点，然后删除。

```
slnodetype *Delete(slnodetype *Head,int key){}中if(Head->number==key)
{
Head=Pointer->next;
free(Pointer);
break;
}
Back = Pointer;
Pointer=Pointer->next;
if(Pointer->number==key)
```

```
{
Back->next=Pointer->next;
free(Pointer);
break;
}
void delete(Node* p)
{
if(Head = Node)

while(p)
}
```

有一个16位的整数，每4位为一个数，写函数求他们的和。
解释：
整数1101010110110111
和 1101+0101+1011+0111
感觉应该不难，当时对题理解的不是很清楚，所以写了一个函数，也不知道对不对。
疑问：
既然是16位的整数，1101010110110111是2进制的，那么函数参数怎么定义呢，请大虾指教
。
答案：用十进制做参数，计算时按二进制考虑。

```
/* n就是16位的数，函数返回它的四个部分之和 */
char SumOfQuaters(unsigned short n)
{
char c = 0;
int i = 4;
do
{
c += n & 15;
n = n >> 4;
} while (--i);

return c;
}
```

有1,2,....一直到n的无序数组,求排序算法,并且要求时间复杂度为O(n),空间复杂度O(1)
,使用交换,而且一次只能交换两个数.（华为）

```
#include<iostream.h>

int main()
{
int a[] = {10,6,9,5,2,8,4,7,1,3};
int len = sizeof(a) / sizeof(int);
int temp;

for(int i = 0; i < len; )
{
temp = a[a[i] - 1];
```

```
a[a[i] - 1] = a[i];
a[i] = temp;

if ( a[i] == i + 1)
i++;
}
for (int j = 0; j < len; j++)
cout<<a[j]<<",";

return 0;
}
```

（慧通）

1 写出程序把一个链表中的接点顺序倒排

```
typedef struct linknode
{
int data;
struct linknode *next;
}node;
//将一个链表逆置
node *reverse(node *head)
{
node *p,*q,*r;
p=head;
q=p->next;
while(q!=NULL)
{
r=q->next;
q->next=p;
p=q;
q=r;
}

head->next=NULL;
head=p;
return head;
}
```

2 写出程序删除链表中的所有接点

```
void del_all(node *head)
{
node *p;
while(head!=NULL)
{
p=head->next;
free(head);
head=p;
}
cout<<"释放空间成功!"<<endl;
}
```

3两个字符串，s,t;把t字符串插入到s字符串中，s字符串有足够的空间存放t字符串

```
void insert(char *s, char *t, int i)
{
char *q = t;
char *p =s;
if(q == NULL)return;
while(*p!='\0')
{
p++;
}
while(*q!=0)
{
*p=*q;
p++;
q++;
}
*p = '\0';
}
```

分析下面的代码：

```
char *a = "hello";
char *b = "hello";
if(a= =b)
printf("YES");
else
printf("NO");
```

这个简单的面试题目,我选输出 no(对比的应该是指针地址吧),可在VC是YES 在C是NO
lz的呢，是一个常量字符串。位于静态存储区，它在程序生命期内恒定不变。如果编译器优化的话，会有可能a和b同时指向同一个hello的。则地址相同。如果编译器没有优化，那么就是两个不同的地址，则不同

写一个函数，功能：完成内存之间的拷贝

memcpy source code:

```
270 void* memcpy( void *dst, const void *src, unsigned int len )
271 {
272 register char *d;
273 register char *s;
27
275 if (len == 0)
276 return dst;
277
278 if (is_overlap(dst, src, len, len))
279 complain3("memcpy", dst, src, len);
280
281 if ( dst > src ) {
282 d = (char *)dst + len - 1;
283 s = (char *)src + len - 1;
284 while ( len >= 4 ) {
285 *d-- = *s--;
286 *d-- = *s--;
```

```
287 *d-- = *s--;
288 *d-- = *s--;
289 len -= 4;
290 }
291 while ( len-- ) {
292 *d-- = *s--;
293 }
294 } else if ( dst < src ) {
295 d = (char *)dst;
296 s = (char *)src;
297 while ( len >= 4 ) {
298 *d++ = *s++;
299 *d++ = *s++;
300 *d++ = *s++;
301 *d++ = *s++;
302 len -= 4;
303 }
304 while ( len-- ) {
305 *d++ = *s++;
306 }
307 }
308 return dst;
309 }
```

公司考试这种题目主要考你编写的代码是否考虑到各种情况，是否安全（不会溢出）
各种情况包括：
１、参数是指针，检查指针是否有效
２、检查复制的源目标和目的地是否为同一个，若为同一个，则直接跳出
３、读写权限检查
４、安全检查，是否会溢出
memcpy拷贝一块内存，内存的大小你告诉它
strcpy是字符串拷贝，遇到'\0'结束

```
/* memcpy ——— 拷贝不重叠的内存块 */
void memcpy(void* pvTo, void* pvFrom, size_t size)
{
void* pbTo = (byte*)pvTo;
void* pbFrom = (byte*)pvFrom;
ASSERT(pvTo != NULL && pvFrom != NULL); //检查输入指针的有效性
ASSERT(pbTo>=pbFrom+size || pbFrom>=pbTo+size);//检查两个指针指向的内存是否重叠

while(size-->0)
*pbTo++ == *pbFrom++;
return(pvTo);
}
```

华为面试题：怎么判断链表中是否有环？

```
bool CircleInList(Link* pHead)
{
if(pHead = = NULL || pHead->next = = NULL)//无节点或只有一个节点并且无自环
```

```
return (false);
if(pHead->next = = pHead)//自环
return (true);
Link *pTemp1 = pHead;//step 1
Link *pTemp = pHead->next;//step 2
while(pTemp != pTemp1 && pTemp != NULL && pTemp->next != NULL)
{
pTemp1 = pTemp1->next;
pTemp = pTemp->next->next;
}
if(pTemp = = pTemp1)
return (true);
return (false);
}
```

两个字符串，s,t;把t字符串插入到s字符串中，s字符串有足够的空间存放t字符串

```
void insert(char *s, char *t, int i)
{
memcpy(&s[strlen(t)+i],&s[i],strlen(s)-i);
memcpy(&s[i],t,strlen(t));
s[strlen(s)+strlen(t)]='\0';
}
```

1。编写一个 C 函数，该函数在一个字符串中找到可能的最长的子字符串，且该字符串是由同一字符组成的。

```
char * search(char *cpSource, char ch)
{
char *cpTemp=NULL, *cpDest=NULL;
int iTemp, iCount=0;
while(*cpSource)
{
if(*cpSource == ch)
{
iTemp = 0;
cpTemp = cpSource;
while(*cpSource == ch)
++iTemp, ++cpSource;
if(iTemp > iCount)
iCount = iTemp, cpDest = cpTemp;
if(!*cpSource)
break;
}
++cpSource;
}
return cpDest;
}
```

2。请编写一个 C 函数，该函数在给定的内存区域搜索给定的字符，并返回该字符所在位置索引值。

```
int search(char *cpSource, int n, char ch)
{
```

```
int i;
for(i=0; i<n && *(cpSource+i) != ch; ++i);
return i;
}
```

一个单向链表，不知道头节点,一个指针指向其中的一个节点，问如何删除这个指针指向的节点？
将这个指针指向的next节点值copy到本节点，将next指向next->next,并随后删除原next指向的节点。

```
#include <stdio.h>
void foo(int m, int n)
{
printf("m=%d, n=%d\n", m, n);
}

int main()
{
int b = 3;
foo(b+=3, ++b);
printf("b=%d\n", b);
return 0;
}
```

输出：m=7,n=4,b=7(VC6.0)
这种方式和编译器中得函数调用关系相关即先后入栈顺序。不过不同编译器得处理不同。也是因为C标准中对这种方式说明为未定义，所以各个编译器厂商都有自己得理解，所以最后产生得结果完全不同。
因为这样，所以遇见这种函数，我们首先要考虑我们得编译器会如何处理这样得函数，其次看函数得调用方式，不同得调用方式，可能产生不同得结果。最后是看编译器优化。

2.写一函数，实现删除字符串str1中含有的字符串str2.
第二个就是利用一个KMP匹配算法找到str2然后删除（用链表实现的话，便捷于数组）

```
/*雅虎笔试题(字符串操作)
给定字符串A和B,输出A和B中的最大公共子串。
比如A="aocdfe" B="pmcdfa" 则输出"cdf"
*/
//Author: azhen
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

char *commanstring(char shortstring[], char longstring[])
{
int i, j;
```

```
char *substring=malloc(256);

if(strstr(longstring, shortstring)!=NULL) //如果……，那么返回shortstring
return shortstring;

for(i=strlen(shortstring)-1;i>0; i--) //否则，开始循环计算
{
for(j=0; j<=strlen(shortstring)-i; j++){
memcpy(substring, &shortstring[j], i);
substring[i]='\0';
if(strstr(longstring, substring)!=NULL)
return substring;
}
}
return NULL;
}


main()
{
char *str1=malloc(256);
char *str2=malloc(256);
char *comman=NULL;

gets(str1);
gets(str2);

if(strlen(str1)>strlen(str2)) //将短的字符串放前面
comman=commanstring(str2, str1);
else
comman=commanstring(str1, str2);

printf("the longest comman string is: %s\n", comman);
}
```

11.写一个函数比较两个字符串str1和str2的大小，若相等返回0，若str1大于str2返回1，若str1小于str2返回－1

```
int strcmp ( const char * src,const char * dst)
{
int ret = 0 ;
while( ! (ret = *(unsigned char *)src - *(unsigned char *)dst) && *dst)
{
++src;
++dst;
}
if ( ret < 0 )
ret = -1 ;
else if ( ret > 0 )
ret = 1 ;
```

```
return( ret );
}
```

3,求1000！的未尾有几个0（用素数相乘的方法来做，如72=2*2*2*3*3）；
求出1->1000里,能被5整除的数的个数n1,能被25整除的数的个数n2,能被125整除的数的个数n3,
能被625整除的数的个数n4.
1000!末尾的零的个数=n1+n2+n3+n4;

```
#include<stdio.h>
#define NUM 1000

int find5(int num){
int ret=0;
while(num%5==0){
num/=5;
ret++;
}
return ret;
}
int main(){
int result=0;
int i;
for(i=5;i<=NUM;i+=5)
{
result+=find5(i);
}
printf(" the total zero number is %d\n",result);
return 0;
}
```

1. 有双向循环链表结点定义为：

```
struct node
{ int data;
struct node *front,*next;
};
```

有两个双向循环链表A，B，知道其头指针为：pHeadA,pHeadB，请写一函数将两链表中data值相同的结点删除

```
BOOL DeteleNode(Node *pHeader, DataType Value)
{
if (pHeader == NULL) return;

BOOL bRet = FALSE;
Node *pNode = pHead;
while (pNode != NULL)
{
if (pNode->data == Value)
{
```

```
if (pNode->front == NULL)
{
pHeader = pNode->next;
pHeader->front = NULL;
}
else
{
if (pNode->next != NULL)
{
pNode->next->front = pNode->front;
}
pNode->front->next = pNode->next;
}

Node *pNextNode = pNode->next;
delete pNode;
pNode = pNextNode;

bRet = TRUE;
//不要break或return, 删除所有
}
else
{
pNode = pNode->next;
}
}

return bRet;
}

void DE(Node *pHeadA, Node *pHeadB)
{
if (pHeadA == NULL || pHeadB == NULL)
{
return;
}

Node *pNode = pHeadA;
while (pNode != NULL)
{
if (DeteleNode(pHeadB, pNode->data))
{
if (pNode->front == NULL)
{
pHeadA = pNode->next;
pHeadA->front = NULL;
}
else
{
pNode->front->next = pNode->next;
```

```
if (pNode->next != NULL)
{
pNode->next->front = pNode->front;
}
}
Node *pNextNode = pNode->next;
delete pNode;
pNode = pNextNode;
}
else
{
pNode = pNode->next;
}
}
}
```

2. 编程实现：找出两个字符串中最大公共子字符串,如"abccade","dgcadde"的最大子串为"cad"

```
int GetCommon(char *s1, char *s2, char **r1, char **r2)
{
int len1 = strlen(s1);
int len2 = strlen(s2);
int maxlen = 0;

for(int i = 0; i < len1; i++)
{
for(int j = 0; j < len2; j++)
{
if(s1[i] == s2[j])
{
int as = i, bs = j, count = 1;
while(as + 1 < len1 && bs + 1 < len2 && s1[++as] == s2[++bs])
count++;

if(count > maxlen)
{
maxlen = count;
*r1 = s1 + i;
*r2 = s2 + j;
}
}
}
}
```

3. 编程实现：把十进制数(long型)分别以二进制和十六进制形式输出，不能使用printf系列库函数

```
char* test3(long num) {
char* buffer = (char*)malloc(11);
buffer[0] = '0';
buffer[1] = 'x';
buffer[10] = '\0';
```

```
char* temp = buffer + 2;
for (int i=0; i < 8; i++) {
temp[i] = (char)(num<<4*i>>28);
temp[i] = temp[i] >= 0 ? temp[i] : temp[i] + 16;
temp[i] = temp[i] < 10 ? temp[i] + 48 : temp[i] + 55;
}
return buffer;
}
```

输入N, 打印 N*N 矩阵
比如 N = 3，打印：

```
1 2 3
8 9 4
7 6 5
```

N = 4，打印：

```
1 2 3 4
12 13 14 5
11 16 15 6
10 9 8 7
```

解答：

```
1 #define N 15
int s[N][N];
void main()
{
int k = 0, i = 0, j = 0;
int a = 1;
for( ; k < (N+1)/2; k++ )
{
while( j < N-k ) s[i][j++] = a++; i++; j--;
while( i < N-k ) s[i++][j] = a++; i--; j--;
while( j > k-1 ) s[i][j--] = a++; i--; j++;
while( i > k ) s[i--][j] = a++; i++; j++;
}
for( i = 0; i < N; i++ )
{
for( j = 0; j < N; j++ )
cout << s[i][j] << '\t';
cout << endl;
}
}
2 define MAX_N 100
int matrix[MAX_N][MAX_N];

/*
```

```
* （x,y）：第一个元素的坐标
* start：第一个元素的值
* n：矩阵的大小
*/
void SetMatrix(int x, int y, int start, int n) {
int i, j;

if (n <= 0) //递归结束条件
return;
if (n == 1) { //矩阵大小为1时
matrix[x][y] = start;
return;
}
for (i = x; i < x + n-1; i++) //矩阵上部
matrix[y][i] = start++;

for (j = y; j < y + n-1; j++) //右部
matrix[j][x+n-1] = start++;

for (i = x+n-1; i > x; i--) //底部
matrix[y+n-1][i] = start++;

for (j = y+n-1; j > y; j--) //左部
matrix[j][x] = start++;

SetMatrix(x+1, y+1, start, n-2); //递归
}

void main() {
int i, j;
int n;

scanf("%d", &n);
SetMatrix(0, 0, 1, n);

//打印螺旋矩阵
for(i = 0; i < n; i++) {
for (j = 0; j < n; j++)
printf("%4d", matrix[i][j]);
printf("\n");
}
}
```

斐波拉契数列递归实现的方法如下：

```
int Funct( int n )
{
if(n==0) return 1;
if(n==1) return 1;
retrurn Funct(n-1) + Funct(n-2);
```

```
}
```
请问，如何不使用递归，来实现上述函数？
请教各位高手！
解答：
```
int Funct( int n ) // n 为非负整数
{
int a=0;
int b=1;
int c;
if(n==0) c=1;
else if(n==1) c=1;
else for(int i=2;i<=n;i++) //应该n从2开始算起
{
c=a+b;
a=b;
b=c;
}
return c;
}
```
解答：
现在大多数系统都是将低字位放在前面，而结构体中位域的申明一般是先声明高位。
100 的二进制是 001 100 100
低位在前 高位在后
001----s3
100----s2
100----s1
所以结果应该是 1
如果先申明的在低位则：
001----s1
100----s2
100----s3
结果是 4
1、原题跟little-endian，big-endian没有关系
2、原题跟位域的存储空间分配有关，到底是从低字节分配还是从高字节分配，从Dev C++和VC7.1上看，都是从低字节开始分配，并且连续分配，中间不空，不像谭的书那样会留空位
3、原题跟编译器有关，编译器在未用堆栈空间的默认值分配上有所不同，Dev C++未用空间分配为
01110111b，VC7.1下为11001100b,所以在Dev C++下的结果为5，在VC7.1下为1。

注：PC一般采用little-endian，即高高低低，但在网络传输上，一般采用big-endian，即高低低高，华为是做网络的，所以可能考虑big-endian模式，这样输出结果可能为4

判断一个字符串是不是回文
```
int IsReverseStr(char *aStr)
{
int i,j;
int found=1;
if(aStr==NULL)
```

```
return -1;
j=strlen(aStr);
for(i=0;i<j/2;i++)
if(*(aStr+i)!=*(aStr+j-i-1))
{
found=0;
break;
}
return found;
}
```

Josephu 问题为：设编号为1，2，… n的n个人围坐一圈，约定编号为k（1<=k<=n）的人从1开始报数，数到m 的那个人出列，它的下一位又从1开始报数，数到m的那个人又出列，依次类推，直到所有人出列为止，由此产生一个出队编号的序列。

数组实现：

```
#include <stdio.h>
#include <malloc.h>
int Josephu(int n, int m)
{
int flag, i, j = 0;
int *arr = (int *)malloc(n * sizeof(int));
for (i = 0; i < n; ++i)
arr[i] = 1;
for (i = 1; i < n; ++i)
{
flag = 0;
while (flag < m)
{
if (j == n)
j = 0;
if (arr[j])
++flag;
++j;
}
arr[j - 1] = 0;
printf("第%4d个出局的人是：%4d号\n", i, j);
}
free(arr);
return j;
}
int main()
{
int n, m;
scanf("%d%d", &n, &m);
printf("最后胜利的是%d号！\n", Josephu(n, m));
system("pause");
return 0;
}
```

链表实现：

```
#include <stdio.h>
```

```
#include <malloc.h>
typedef struct Node
{
int index;
struct Node *next;
}JosephuNode;
int Josephu(int n, int m)
{
int i, j;
JosephuNode *head, *tail;
head = tail = (JosephuNode *)malloc(sizeof(JosephuNode));
for (i = 1; i < n; ++i)
{
tail->index = i;
tail->next = (JosephuNode *)malloc(sizeof(JosephuNode));
tail = tail->next;
}
tail->index = i;
tail->next = head;

for (i = 1; tail != head; ++i)
{
for (j = 1; j < m; ++j)
{
tail = head;
head = head->next;
}
tail->next = head->next;
printf("第%4d个出局的人是：%4d号\n", i, head->index);
free(head);
head = tail->next;
}
i = head->index;
free(head);
return i;
}
int main()
{
int n, m;
scanf("%d%d", &n, &m);
printf("最后胜利的是%d号！\n", Josephu(n, m));
system("pause");
return 0;
}
```

已知strcpy函数的原型是：
char * strcpy(char * strDest,const char * strSrc);
1.不调用库函数，实现strcpy函数。
2.解释为什么要返回char *。
解说：

1.strcpy的实现代码
```
char * strcpy(char * strDest,const char * strSrc)
{
if ((strDest==NULL)||(strSrc==NULL)) file://[/1]
throw "Invalid argument(s)"; //[2]
char * strDestCopy=strDest; file://[/3]
while ((*strDest++=*strSrc++)!='\0'); file://[/4]
return strDestCopy;
}
```
错误的做法：
[1]
(A)不检查指针的有效性，说明答题者不注重代码的健壮性。
(B)检查指针的有效性时使用((!strDest)||(!strSrc))或(!(strDest&&strSrc))，说明答题者对C语言中类型的隐式转换没有深刻认识。在本例中char *转换为bool即是类型隐式转换，这种功能虽然灵活，但更多的是导致出错概率增大和维护成本升高。所以C++专门增加了bool、true、false三个关键字以提供更安全的条件表达式。
(C)检查指针的有效性时使用((strDest==0)||(strSrc==0))，说明答题者不知道使用常量的好处。直接使用字面常量（如本例中的0）会减少程序的可维护性。0虽然简单，但程序中可能出现很多处对指针的检查，万一出现笔误，编译器不能发现，生成的程序内含逻辑错误，很难排除。而使用NULL代替0，如果出现拼写错误，编译器就会检查出来。
[2]
(A)return new string("Invalid argument(s)");，说明答题者根本不知道返回值的用途，并且他对内存泄漏也没有警惕心。从函数中返回函数体内分配的内存是十分危险的做法，他把释放内存的义务抛给不知情的调用者，绝大多数情况下，调用者不会释放内存，这导致内存泄漏。
(B)return 0;，说明答题者没有掌握异常机制。调用者有可能忘记检查返回值，调用者还可能无法检查返回值（见后面的链式表达式）。妄想让返回值肩负返回正确值和异常值的双重功能，其结果往往是两种功能都失效。应该以抛出异常来代替返回值，这样可以减轻调用者的负担、使错误不会被忽略、增强程序的可维护性。
[3]
(A)忘记保存原始的strDest值，说明答题者逻辑思维不严密。
[4]
(A)循环写成while (*strDest++=*strSrc++);，同[1](B)。
(B)循环写成while (*strSrc!='\0') *strDest++=*strSrc++;，说明答题者对边界条件的检查不力。循环体结束后，strDest字符串的末尾没有正确地加上'\0'。