1、int a[10]={1,2,3,4,5,6,7,8,9,0};  
   int \*p=&a[1];  
   则p[6]等于8

|  |
| --- |
| 下标的计数从0开始； |

2、整数数组清零：bzero(),memset()。

|  |  |
| --- | --- |
| 去c语言库里查一下这两个函数的使用 memset *语法:*   |  | | --- | | #include <string.h>  void \*memset( void \*buffer, int ch, size\_t count ); |   功能: 函数拷贝*ch* 到*buffer* 从头开始的*count* 个字符里, 并返回*buffer*指针。 memset() 可以应用在将一段内存初始化为某个值。例如：  memset( the\_array, '\0', sizeof(the\_array) );  memset(chk,0,NUMBER);（使用过，正在使用）  这是将一个数组的所以分量设置成零的很便捷的方法。  **bzero函数**  函数原型：void bzero（void \*s, int n）；  [头文件](https://so.csdn.net/so/search?q=%E5%A4%B4%E6%96%87%E4%BB%B6&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/weixin_42235488/article/details/_blank)：#include <string.h>  功能：将[字符串](https://so.csdn.net/so/search?q=%E5%AD%97%E7%AC%A6%E4%B8%B2&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/weixin_42235488/article/details/_blank)s的前n个字节置为0，一般来说n通常取sizeof(s),将整块空间清零。  返回值：无返回值  bzero( the\_array, sizeof(the\_array) );  bzero(chk,NUMBER); |

3、sizeof();[测试](http://lib.csdn.net/base/softwaretest" \o "软件测试知识库)变量所占地址的字节数

|  |
| --- |
| Char占一个字节  Int占4个字节  对于指针的话，只测试指针变量地址的字节数  对于数组，可以算整个数组所占的地址的字节数  复习变量，以及它们所占的地址。  测试结构体的sizeof(); |

4、 main()   
 {   
  char \*str[]={"ab","cd","ef","gh","ij","kl"};   
  char \*t;   
  t=(str+4)[-1];   
  printf("%s",t);   
 }则显示"gh"

|  |
| --- |
| 数组从0开始算 |

5、小端：低位字节数据存储在低地址  
 大端：高位字节数据存储在低地址  
 例如：int a=0x12345678;(a首地址为0x2000)  
 0x2000  0x2001  0x2002  0x2003  
  0x12    0x34    0x56    0x78      大端格式

|  |
| --- |
| int a=0x12345678;  大端格式：高位字节数据存储在低地址  低地址-->高地址  0x2000  0x2001  0x2002  0x2003  0x12 ox34 0x56 0x78  小端格式：低位字节数据存储在低地址  低地址-->高地址  0x2000  0x2001  0x2002  0x2003  0x78 0x56 ox34 0x12 |

6、异步IO和同步IO区别  
 如果是同步IO，当一个IO操作执行时，应用程序必须等待，直到此IO执行完，相反，异步IO操作在后台运行，  
 IO操作和应用程序可以同时运行，提高系统性能，提高IO流量； 在同步文件IO中，线程启动一个IO操作然后就立即进入等待状态，直到IO操作完成后才醒来继续执行，而异步文件IO中，  
 线程发送一个IO请求到内核，然后继续处理其他事情，内核完成IO请求后，将会通知线程IO操作完成了。

|  |
| --- |
| 异步IO：类似标志位，另一个线程接收标志位，等主线程执行完，再运行  同步IO:主线程没执行完，再发数据，主线程接收不到。 |

7、用变量a定义  
 一个整型数   int a;  
 一个指向整型数的指针  int \*a;  
 一个指向指针的指针，它指向的指针是指向一个整型数  int \*\*a;  
 一个有10个整型数的数组   int a[10];  
 一个有10指针的数组，该指针是指向一个整型数  int \*a[10];//这个数组里面有十个指针  
 一个指向有10个整型数数组的指针   int (\*a)[10];//这个指针指向一个数组  
 一个指向函数的指针，该函数有一个整型数参数并返回一个整型数   int (\*a)(int);  
 一个有10个指针的数组，该指针指向一个函数，该函数有一个整型数参数并返回一个整型  int (\*a[10])(int);

8、int foo(void)  
{  
 int i;  
 char c=0x80;  
 i=c;  
 if(i>0)  
  return 1;  
 return 2;  
}返回值为2；因为i=c=-128;如果c=0x7f,则i=c=127。

|  |
| --- |
|  |

9、a=b\*2;a=b/4;a=b%8;a=b/8\*8+b%4;a=b\*15;效率最高的[算法](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库)  
 a=b\*2 -> a=b<<1;  
 a=b/4 -> a=b>>2;  
 a=b%8 -> a=b&7;  
 a=b/8\*8+b%4 -> a=((b>>3)<<3)+(b&3)  
 a=b\*15 -> a=(b<<4)-b  
10、c关键字

 c的关键字共32个  
 \*数据类型关键字（12）  
 char,short,int,long,float,double,unsigned,signed,union,enum,void,struct  
 \*控制语句关键字（12）  
 if,else,switch,case,default,for,do,while,break,continue,goto,return  
 \*存储类关键字（5）  
 auto,extern,register,static,const  
 \*其他关键字（3）  
 sizeof,typedef,volatile  
11、int main(void)  
 {  
  unsigned int a = 6;  
  int b = -20;  
  char c;  
  (a+b>6)?(c=1):(c=0);  
 }则c=1，但a+b=-14;如果a为int类型则c=0。  
 原来有符号数和无符号数进行比较运算时（==，<,>,<=,>=）,有符号数隐式转换成了无符号数（即底层的补码不变，但是此数从有符号数变成了无符号数），  
 比如上面 (a+b)>6这个比较运算，a+b=-14，-14的补码为1111111111110010。此数进行比较运算时，  
 被当成了无符号数，它远远大于6，所以得到上述结果。  
12、给定一个整型变量a，写两段代码，第一个设置a的bit3，第二个清除a的bit，在以上两个操作中，  
 要保持其它位不变。  
 #define BIT3 (0x1<<3)  
　　static int a;  
　　void set\_bit3(void)  
　　{  
　　 a |= BIT3;  
　　}  
　　void clear\_bit3(void)  
　　{  
　　 a &= ~BIT3;  
　　}  
13、要求设置一绝对地址为0x67a9的整型变量的值为0xaa66。  
 　int \*ptr;  
　　  ptr = (int \*)0x67a9;  
　    \*ptr = 0xaa66;（建议用这种）  
　　  
　　  一个较晦涩的方法是：  
　　  \*(int \* const)(0x67a9) = 0xaa66;  
14、中断是[嵌入式](http://lib.csdn.net/base/embeddeddevelopment" \o "嵌入式开发知识库)系统中重要的组成部分，这导致了很多编译开发商提供一种扩展—让标准C支持中断。  
 具代表性的是，产生了一个新的关键字\_\_interrupt。下面的代码就使用了\_\_interrupt关键字去定义了一个中断服务子程序(ISR)，请评论一下这段代码的。  
 \_\_interrupt void compute\_area (void)   
 {   
 　　double area = PI \* radius \* radius;   
 　　printf(" Area = %f", area);   
 　　return area;   
 }   
 ISR不可能有参数和返回值的！  
 ISR尽量不要使用浮点数处理程序，浮点数的处理程序一般来说是不可重入的，而且是消耗大量CPU时间的！！  
 printf函数一般也是不可重入的，UART属于低速设备，printf函数同样面临大量消耗CPU时间的问题！  
15、评价下面的代码片断：  
　　  
　　unsigned int zero = 0;  
　　unsigned int compzero = 0xFFFF;  
　　/\*1's complement of zero \*/  
　　对于一个int型不是16位的处理器为说，上面的代码是不正确的。应编写如下：  
　　unsigned int compzero = ~0;  
16、main()  
 {  
  char \*ptr;  
 　　if ((ptr = (char \*)malloc(0)) == NULL)  
 　　puts("Got a null pointer");  
 　　else  
 　　puts("Got a valid pointer");  
 }  
 该代码的输出是“Got a valid pointer”。还可以\*ptr='a';不出现段错误  
17、Typedef 在[C语言](http://lib.csdn.net/base/c" \o "C语言知识库)中频繁用以声明一个已经存在的数据类型的同义字。也可以用预处理器做类似的事。  
    例如，思考一下下面的例子：  
　　#define dPS struct s \*  
　　typedef struct s \* tPS;  
　　  
　　以上两种情况的意图都是要定义dPS 和 tPS 作为一个指向结构s指针。哪种方法更好呢？（如果有的话）为什么？  
　　这是一个非常微妙的问题，任何人答对这个问题（正当的原因）是应当被恭喜的。答案是：typedef更好。思考下面的例子：  
　　dPS p1,p2;  
　　tPS p3,p4;  
　　  
　　第一个扩展为  
　　struct s \* p1, p2;  
　　  
　　上面的代码定义p1为一个指向结构的指，p2为一个实际的结构，这也许不是你想要的。第二个例子正确地定义了p3 和p4 两个指针。  
18、int a = 5, b = 7, c;  
　　c = a+++b;  
 则c=12。  
19、int main()  
 {  
  int j=2;  
  int i=1;  
  if(i = 1) j=3;  
  if(i = 2) j=5;  
  printf("%d",j);  
 }   
 输出为5；如果再加上if(i=3)j=6;则输出6。  
20、宏定义是在预编译阶段被处理的。  
21、Norflash与Nandflash的区别  
 (1)、NAND闪存的容量比较大  
 (2)、由于NandFlash没有挂接在地址总线上，所以如果想用NandFlash作为系统的启动盘，就需要CPU具备特殊的功能，  
 如s3c2410在被选择为NandFlash启动方式时会在上电时自动读取NandFlash的4k数据到地址0的SRAM中。  
 (3)、NAND Flash一般地址线和数据线共用，对读写速度有一定影响。NOR Flash闪存数据线和地址线分开，  
    所以相对而言读写速度快一些。  
22、反码：对原码除符号位外的其余各位逐位取反就是反码  
 补码：负数的补码就是对反码加1  
 正数的原码、反码、补码都一样  
23、pthread\_t tid;  
 pthread\_create(&tid,NULL,pthread\_func,NULL);//创建线程  
 pthread\_join(tid,NULL);//等待子线程结束，并回收资源  
 pthread\_detach(tid);//与当前进程分离  
 pthread\_exit(NULL);//退出调用线程  
 pthread\_cancel(tid);//取消线程  
 pthread\_mutex mutex=PTHREAD\_MUTEX\_INITIALIZER;  
 pthread\_mutex\_init(&mutex,NULL);//初始化一个互斥锁  
 pthread\_mutex\_lock(&mutex);//对互斥锁上锁  
 pthread\_mutex\_unlock(&mutex);//对互斥锁解锁  
 sem\_t sem;  
 sem\_init(&sem,0,1);//创建信号量并初始化它的值  
 sem\_wait(&sem);//信号量的值减1  
 sem\_post(&sem);//信号量的值加1  
24、内存管理MMU的作用  
  \*内存分配和回收  
  \*内存保护  
  \*内存扩充  
  \*地址映射   
25、ROM是只读存储器，掉电不丢失  
 RAM是读写存储器，掉电丢失  
26、SRAM：CPU的缓存就是SRAM，静态的随机存取存储器，加电情况下，不需要刷新，数据不会丢失  
 DRAM，动态随机存取存储器最为常见的系统内存，需要不断刷新，才能保存数据  
 SDRAM：同步动态随机存储器，即数据的读取需要时钟来同步。  
27、signed char 的取值范围-128~127.  
28、编译和链接有什么不同？（如外部符号的处理）  
 编译生成的是目标文件(object  \*.o);  
 编译过程中对于外部符号不做任何解释和处理。外部符号对应的就是“符号”

 链接生成的是可执行程序  
 链接将会解释和处理外部符号。外部符号对应的是地址  
29、已知strcpy函数的函数原型是：  
 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  
 char \*strcpy(char \*strDest, const char \*strSrc)  
 {  
  int i=0;  
  if(!(strDest && strSrc))  
   return;  
  while(strDest[i++] = \*strSrc++);  
  return strDest;  
 }  
30、strcpy能把strSrc的内容复制到strDest，为什么还要char \*类型的返回值？  
 为了实现链式表达式  
 int len = strlen(strcpy(strDest, strSrc));  
31、写一个“标准”宏MIN，这个宏输入两个参数并返回较小的一个  
 #define MIN(a, b) (a) <= (b) ? (a) : (b)  
32、关键字static的作用是什么  
 static用来修饰一个局部的变量的时候，  
  生命域是全局的  
  作用域是局部的

 static用来修饰一个模块内的（某一个C的源程序文件）全局变量的时候  
  生命域不变  
  作用域减小，只在本模块内有效

 static用来修饰一个函数的时候  
  作用域减小，只在本模块内有效

33、说明下面的声明的含义：  
　　A.  
　　const int a;     // a是一个常数  
　　int const a;     // a是一个常数  
　　B.  
　　const int \*a;     // a是一个指向整型常数的指针  
　　int \* const a;     // a是一个指向整型变量的常指针  
　　int const \* a const;   // a是一个指向整型常数的常指针  
 C.  
 char \*strcpy(char \*strDest, const char \*strSrc);  
       // 参数在函数内部不会被修改  
 const int strcmp(char \*source, char \*dest);  
       // 函数的返回值不能被修改  
 const int a = strcmp(xx, yy);  
 if(strcmp(xx,yy) != 0)  
34、说明关键字volatile有什么含意，并给出例子。  
 volatile表示被修饰的符号是易变的。告诉编译器不要随便优化我的代码！！  
 \*一个硬件寄存器  
 \*中断中用到的变量  
 \*线程之间共享变量  
 volatile int a = 10;  
 while((a & 0x01) == 0);  
 #define P\_UART\_STATUS ((const volatile unsigned int \*)0x88000000);  
       // volatile表示硬件会修改这个寄存器的内容  
       // const表示该寄存器只读，写无意义  
35、printf可以接受多个参数，为什么，请写出printf的原型。  
 int printf(const char \*fmt, ...);  
36、什么是堆栈，简述为什么需要堆栈？  
 堆栈是计算机中最常用的一种[数据结构](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库)，保存数据的一块连续内存；比如函数的调用是用堆栈实现的。  
37、请列举常用的串行通信方式（两种以上），并简述串行通信和并行通信不同之处、优缺点。  
 异步通信和同步通信；并行速度快，串行口线间干扰小  
38、列举一下你熟悉7层OSI协议中的几层。说说你最熟悉的一层协议的功能。  
    应用层，表示层，会话层，传输层，网络层，数据链路层，物理层。  
39、路由协议：网关-网关协议，外部网关协议，内部网关协议（RIP-1、RIP-IGRP、EIGRP、IS-IS和OSPF）  
40、位转换  
 位 8   7   6   5   4   3   2    1  
 数 v8  v7  v6  v5  v4  v3  v2   v1

 转换后：  
 位 8   7   6   5   4    3   2   1  
 数 v1  v2  v3  v4  v5   v6  v7  v8  
 unsigned char  bit\_reverse(unsigned char  c)  
 {  
 　　unsigned char buf = 0;  
 　　int bit = 8;  
 　　while(bit)  
 　　{  
 　　bit--;  
 　　buf |= ((c & 1) << bit);  
 　　c >>=1;  
 　　}  
 　　return buf;  
 }  
41、字符串倒序  
 1)、inverted\_order(char \*p)  
  {  
   char \*s1,\*s2,tem;  
   s1=p;  
   s2=s1+strlen(p)-1;  
   while(s1<s2)  
   {  
    tem=\*s1;  
    \*s1=\*s2;  
    \*s2=tem;  
    s1++;  
    s2--;  
   }  
  }  
 2)、inverted\_order(char \*p)  
  {  
   int len = strlen(src);  
   char \*des = (char \*)malloc(len + 1);  
   char \*s = &src[len -1];  
   char \*d = des;  
   while(len-- != 0)  
    \*d++ = \*s--;  
   \*d = 0;  
   free(des);  
  }  
42、引用和指针的区别  
  (1). 指针是一个实体，而引用仅是个别名；  
　　 (2). 引用使用时无需解引用（\*），指针需要解引用；  
　　 (3). 引用只能在定义时被初始化一次，之后不可变；指针可变；  
　　 (4). 引用没有 const，指针有 const，const 的指针不可变；  
　　 (5). 引用不能为空，指针可以为空；  
　　 (6). “sizeof 引用”得到的是所指向的变量（对象）的大小，  
    而“sizeof指针”得到的是指针本身（所指向的变量或对象的地址）的大小；  
　　 (7). 指针和引用的自增（++）运算意义不一样；  
43、队列和栈的区别  
 队列是先进先出，只能在一端插入另一端删除，可以从头或尾进行遍历（但不能同时遍历），  
 栈是先进后出，只能在同一端插入和删除，只能从头部取数据  
44、四层模型？七层模型？TCP/IP协议包括？  
 这7层是:物理层、数据链路层、网路层、传输层、话路层、表示层和应用层  
 这4层分别为：应用层、传输层、网络层、链路层。   
 TCP/IP协议族包括(IP)、(ARP)、(RARP)、(ICMP)、(UDP)、(TCP)、(RIP)、Telnet、(SMTP)、DNS等协议。  
45、TCP通信建立和结束的过程？端口的作用  
 三次握手和四次挥手；端口是一个软件结构，被客户程序或服务进程用来发送和接收信息。一个端口对应一个16比特的数。服务进程通常使用一个固定的端口。  
    21端口：21端口主要用于FTP（File Transfer Protocol，文件传输协议）服务。    23端口：23端口主要用于Telnet（远程登录）服务，是Internet上普遍采用的登录和仿真程序。　　  
    25端口：25端口为SMTP（Simple Mail TransferProtocol，简单邮件传输协议）服务器所开放，  
    主要用于发送邮件，如今绝大多数邮件服务器都使用该协议。　　  
    53端口：53端口为DNS（Domain Name Server，域名服务器）服务器所开放，  
    主要用于域名解析，DNS服务在NT系统中使用的最为广泛。　　  
    67、68端口：67、68端口分别是为Bootp服务的Bootstrap Protocol Server  
    （引导程序协议服务端）和Bootstrap Protocol Client（引导程序协议客户端）开放的端口。　　  
    69端口：TFTP是Cisco公司开发的一个简单文件传输协议，类似于FTP。　　  
    79端口：79端口是为Finger服务开放的，主要用于查询远程主机在线用户、[操作系统](http://lib.csdn.net/base/operatingsystem" \o "操作系统知识库)类型以及是否缓冲区溢出等用户的详细信息。880端口：80端口是为HTTP（HyperText Transport Protocol，超文本传输协议）开放的，  
    这是上网冲浪使用最多的协议，主要用于在WWW（World Wide Web，万维网）服务上传输信息的协议。　　  
    99端口：99端口是用于一个名为“Metagram Relay”（亚对策延时）的服务，  
    该服务比较少见，一般是用不到的。　　  
    109、110端口：109端口是为POP2（Post Office Protocol Version 2，邮局协议2）服务开放的，  
    110端口是为POP3（邮件协议3）服务开放的，POP2、POP3都是主要用于接收邮件的。　  
    111端口：111端口是SUN公司的RPC（Remote Procedure Call，远程过程调用）服务所开放的端口，  
     主要用于分布式系统中不同计算机的内部进程通信，RPC在多种网络服务中都是很重要的组件。　　  
    113端口：113端口主要用于Windows的“Authentication Service”（验证服务）。　　  
    119端口：119端口是为“Network News Transfer Protocol”（网络新闻组传输协议，简称NNTP）开放的。　　  
    135端口：135端口主要用于使用RPC（Remote Procedure Call，远程过程调用）协议并提供DCOM（分布式组件对象模型）服务。　　  
    137端口：137端口主要用于“NetBIOS Name Service”（NetBIOS名称服务）。　　  
    139端口：139端口是为“NetBIOS Session Service”提供的，主要用于提供Windows文件和打印机共享以及Unix中的Samba服务。　　  
    143端口：143端口主要是用于“Internet Message Access Protocol”v2（Internet消息访问协议，简称IMAP）。　　  
    161端口：161端口是用于“Simple Network Management Protocol”（简单网络管理协议，简称SNMP）。　　  
    443端口：43端口即网页浏览端口，主要是用于HTTPS服务，是提供加密和通过安全端口传输的另一种HTTP。　　  
    554端口：554端口默认情况下用于“Real Time Streaming Protocol”（实时流协议，简称RTSP）。　　  
    1024端口：1024端口一般不固定分配给某个服务，在英文中的解释是“Reserved”（保留）。　　  
    1080端口：1080端口是Socks代理服务使用的端口，大家平时上网使用的WWW服务使用的是HTTP协议的代理服务。　　  
    1755端口：1755端口默认情况下用于“Microsoft Media Server”（微软媒体服务器，简称MMS）。　　  
    4000端口：4000端口是用于大家经常使用的QQ聊天工具的，再细说就是为QQ客户端开放的端口，QQ服务端使用的端口是8000。　　  
    5554端口：在今年4月30日就报道出现了一种针对微软lsass服务的新蠕虫病毒——震荡波（Worm.Sasser），该病毒可以利用TCP 5554端口开启一个FTP服务，主要被用于病毒的传播。　　  
    5632端口：5632端口是被大家所熟悉的远程控制软件pcAnywhere所开启的端口。　　  
    8080端口：8080端口同80端口，是被用于WWW代理服务的，可以实现网页浏览。  
46、物理地址转换成IP地址的协议？反之？  
 地址解析协议（ARP）的作用是将IP地址转换成物理地址；反地址解析协议（RARP）则负责将物理地址转换成IP地址。  
47、WLAN：无线局域网络：利用射频技术进行数据传输的系统。  
 　　WLAN是指应用无线通信技术将计算机设备互联起来，构成可以互相通信和实现资源共享的网络体系。无线局域网本质的特点是不再使用通信电缆将计算机与网络连接起来，而是通过无线的方式连接，从而使网络的构建和终端的移动更加灵活。

48、已知数组table，用宏求元素个数。  
　　COUNT(table)  (sizeof(table)/sizeof(table[0]));

49、定义一个两个参数的标准宏MAX，总是输出最大值。  
　　#define  MAX(a,b)  ((a)>(b)?(a):(b))  
50、什么是平衡二叉树？  
 当且仅当两个子树的高度差不超过1时，这个树是平衡二叉树。  
51、全局变量和局部变量的区别。  
　　全局变量,储存在静态区.进入main函数之前就被创建.生命周期为整个源程序;   
 局部变量,在栈中分配.在函数被调用时才被创建.生命周期为函数内。  
52、数组与链表的区别。  
　　数组中的数据在内存中的按顺序存储的，而链表是随机存储的！  
 要访问数组中的元素可以按下标索引来访问，速度比较快，如果对他进行插入操作的话， 就得移动很多元素，所以对数组进行插入操作效率很低！由于连表是随机存储的，  
 链表在插入，删除操作上有很高的效率（相对数组），如果要访问链表中的某个元素的话，  
 那就得从链表的头逐个遍历，直到找到所需要的元素为止，  
 所以链表的随机访问的效率就比数组要低    
53、死锁的四个条件及处理方法。  
　　(1)互斥条件：一个资源每次只能被一个进程使用。      
 (2) 请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放。   
    (3)不剥夺条件:进程已获得的资源，在末使用完之前，不能强行剥夺。     
    (4)循环等待条件:若干进程之间形成一种头尾相接的循环等待资源关系。    
　　    解决死锁的方法分为死锁的预防，避免，检测与恢复三种   
54、进程调度策略。  
　　先进先出算法，最短CPU运行期优先调度算法，轮转法，多级队列方法  
55、[Linux](http://lib.csdn.net/base/linux" \o "Linux知识库)驱动程序流程及功能。  
　　设备驱动程序的功能：  
　　对设备初始化和释放  
　　把数据从内核传送到硬件和从硬件读取数据  
　　读取应用程序传送给设备文件的数据和回送应用程序请求的数据  
　　检测和处理设备出现的错误  
56、时间换空间、空间换时间的例子。  
　　冒泡排序 --- 时间换空间  
　　快速排序，堆排序 --- 空间换时间    
57、MAC层通信协议有哪些？  
　　ISO2110，IEEE802，IEEE802.2    
58、关键字static的作用是什么？  
 \*在函数体内，一个被声明为静态的变量在这一函数被调用过程中维持其值不变（该变量存放在静态变量区）。  
    \*在模块内（但在函数体外），一个被声明为静态的变量可以被模块内所用函数访问，但不能被模块外其它函数访问。  
     它是一个本地的全局变量。  
    \*在模块内，一个被声明为静态的函数只可被这一模块内的其它函数调用。  
  那就是，这个函数被限制在声明它的模块的本地范围内使用。  
59、参数的传递方式有几种？  
 1、值传递  
    2、指针传递  
    严格来看，只有一种传递，值传递，指针传递也是按值传递的，复制的是地址。  
60、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。   
61、如何引用一个已经定义过的全局变量？  　　  
   答：extern 可以用引用头文件的方式，也可以用extern关键字，如果用引用头文件方式来引用某个在头文件中声明的全局变理，  
   假定你将那个变量写错了，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。   
62、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    
63、语句for(   ；1   ；)有什么问题？它是什么意思？  　　答：和while(1)相同。

64、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)的说明之前再加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。 从以上分析可以看出，把局部变量改变为静态变量后是改变了它的存储  
 方式即改变了它的生存期。把全局变量改变为静态变量后是改变了它的作用域，     
 限制了它的使用范围。static函数与普通函数作用域不同。仅在本文件。只在当前源文件中使用的函数应该说明为内部函数(static)  
 ，内部函数应该在当前源文件中说明和定义。对于可在当前源文件以外使用的函数，应该在一个头文件中说明，  
 要使用这些函数的源文件要包含这个头文件 static全局变量与普通的全局变量有什么区别：  
 static全局变量只初使化一次，防止在其他文件单元中被引用static局部变量和普通局部变量有什么区别：  
 static局部变量只被初始化一次，下一次依据上一次结果值； static函数与普通函数有什么区别：  
 static函数在内存中只有一份，普通函数在每个被调用中维持一份拷贝  
65、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。  
66、-1,2,7,28,,126请问28和126中间那个数是什么？为什么？  　　  
    答案应该是4^3-1=63 规律是n^3-1(当n为偶数0，2，4)n^3+1(当n为奇数1，3，5)   
67、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；  
68、.[软件测试](http://lib.csdn.net/base/softwaretest" \o "软件测试知识库)都有那些种类?    
  人工测试：个人复查、抽查和会审   
  机器测试：黑盒测试（针对系统功能的测试 ）和白盒测试 （测试函数功能，各函数接口）　  
69、堆栈溢出一般是由什么原因导致的？  
      没有回收垃圾资源。  
70、写出float x 与“零值”比较的if语句。  
      if(x>0.000001&&x<-0.000001)  
71、P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  
72、写一个程序, 要求功能：求出用1，2，5这三个数不同个数组合的和为100的组合个数。  
   如：100个1是一个组合，5个1加19个5是一个组合。。。。 请用C++语言写。

 答案：最容易想到的算法是：  
 设x是1的个数，y是2的个数，z是5的个数，number是组合数  
 注意到0<=x<=100，0<=y<=50，0<=z=20，所以可以编程为：  
 number=0;  
 for (x=0; x<=100; x++)  
  for (y=0; y<=50; y++)  
   for (z=0; z<=20; z++)  
    if ((x+2\*y+5\*z)==100)  
     number++;  
73、内存对齐问题的原因?  
 平台原因（移植原因）：不是所有的硬件平台都能访问任意地址上的任意数据；  
 性能原因：数据结构（尤其是栈）应该尽可能地在自然边界上对齐，因为为了访问未对齐的内存，处理器需要做两次内存访问，  
     而对齐的内存访问仅需要一次。  
74、比较一下进程和线程的区别？  
 （1）、调度：线程是CPU调度和分派的基本单位  
 （2）、拥有资源：  
  \*  进程是系统中程序执行和资源分配的基本单位  
  \*  线程自己一般不拥有资源（除了必不可少的程序计数器，一组寄存器和栈），但他可以去访问其所属进程的资源，  
     如进程代码，数据段以及系统资源（已打开的文件，I/O设备等）。  
 （3）系统开销：  
  \*  同一进程中的多个线程可以共享同一地址空间，因此它们之间的同步和通信的实现也比较简单  
  \*  在进程切换的时候，涉及到整个当前进程CPU环境的保存以及新被调度运行的进程的CPU环境的设置；  
     而线程切换只需要保存和设置少量寄存器的内容，并不涉及存储器管理方面的操作，从而能更有效地使用系统资源和  
     提高系统吞吐量。  
75、main()  
 {  
   int a[5]={1,2,3,4,5};  
    int \*ptr=(int \*)(&a+1);//&a相当于变成了行指针，加1则变成了下一行首地址  
    printf("%d,%d",\*(a+1),\*(ptr-1));  
 }  
 \*(a+1）就是a[1]，\*(ptr-1)就是a[4],执行结果是2，5  
76、 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  
77、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？  
 答：程序崩溃。因为GetMemory并不能传递动态内存，Test函数中的 str一直都是 NULL。strcpy(str, "hello world");将使程序崩溃。

 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？  
 答：（1）能够输出hello  
     （2）内存泄漏  
78、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：可能是乱码。因为GetMemory返回的是指向“栈内存”的指针，该指针的地址不是 NULL，  
 但其原现的内容已经被清除，新内容不可知。  
79、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：篡改动态内存区的内容，后果难以预料，非常危险。因为free(str);之后，str成为野指针，  
 if(str != NULL)语句不起作用。  
 野指针不是NULL指针，是指向被释放的或者访问受限内存指针。  
 造成原因：指针变量没有被初始化任何刚创建的指针不会自动成为NULL；  
     指针被free或delete之后，没有置NULL；  
     指针操作超越了变量的作用范围，比如要返回指向栈内存的指针或引用，因为栈内存在函数结束时会被释放。  
80、unsigned char \*p=(unsigned char \*)0x0801000  
    unsigned char \*q=(unsigned char \*)0x0810000  
 p+5 =?    0x0801005  
 q+5 =?    0x0810005  
81、进程间通信方式：管道、命名管道、消息队列、共享内存、信号、信号量、套接字。  
 (1)、 管道( pipe )：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程间使用。  
       进程的亲缘关系通常是指父子进程关系。  
 (2)、有名管道 (named pipe) ：有名管道也是半双工的通信方式，但是它允许无亲缘关系进程间的通信。  
 (3)、信号量( semophore ) ：信号量是一个计数器，可以用来控制多个进程对共享资源的访问。  
      它常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源。  
      因此，主要作为进程间以及同一进程内不同线程之间的同步手段。  
 (4)、消息队列( message queue ) ： 消息队列是由消息的链表，存放在内核中并由消息队列标识符标识。  
      消息队列克服了信号传递信息少、管道只能承载无格式字节流以及缓冲区大小受限等缺点。  
 (5)、信号 ( sinal ) ： 信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生。  
 (6)、共享内存( shared memory ) ：共享内存就是映射一段能被其他进程所访问的内存，  
      这段共享内存由一个进程创建，但多个进程都可以访问。共享内存是最快的 IPC 方式，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，  
      如信号两，配合使用，来实现进程间的同步和通信。  
 (7)、套接字( socket ) ： 套接字也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同及其间的进程通信。   
82、宏和函数的优缺点？  
 (1)、函数调用时，先求出实参表达式的值，然后带入形参。而使用带参数的宏只是进行简单的字符替换。  
 (2)、函数调用是在程序运行时处理的，分配临时的内存单元；而宏展开则是在编译时进行的，在展开时并不分配内存单元，  
   不进行值的传递处理，也没有“返回值”的概念。  
 (3)、对函数中的实参和形参都要定义类型，二者的类型要求一致，应进行类型转换；而宏不存在类型问题，宏名无类型，  
   它的参数也是无类型，只是一个符号代表，展开时带入指定的字符即可。宏定义时，字符串可以是任何类型的数据。  
 (4)、调用函数只可得到一个返回值，而宏定义可以设法得到几个结果。  
 (5)、使用宏次数多时，宏展开后源程序长，因为每次展开一次都使程序增长，而函数调用不使源程序变长。  
 (6)、宏替换不占运行时间，只占编译时间；而函数调用则占运行时间（分配单元、保留现场、值传递、返回）。  
83、C和c++的不同  
 c和c++的一些不同点（从语言本身的角度）：  
 1）c++源于c，c++最重要的特性就是引入了面向对象机制，class关键字。  
 2）c++中，变量可以再任何地方声明；c中，局部变量只能在函数开头声明。  
 3）c++中，const型常量是编译时常量；c中，const常量只是只读的变量。  
 4）c++有&引用;c没有  
 5）c++的struct声明自动将结构类型名typedef；c中struct的名字只在结构标签名字空间中，不是作为一种类型出现  
 6）c语言的main函数可以递归调用;c++中则不可以  
 7）c中，void \*可以隐式转换成其他指针类型；c++中要求现实转换，否则编译通不过  
84、6.大小端格式问题。  
 方法一：  
 void checkCpuMode(void)  
 {  
  int i = 0x12345678;  
  char \*cp = (char \*)&i;  
  if(\*cp == 0x78)  
   printf("little endian");  
  else  
   printf("big endian\n");  
 }  
 方法二：  
 void checkCpuMode(void)  
 {  
  int a = 0x12345678;  
  if((char)a == 0x12)  
   printf("big endian\n");  
  else   
   printf("little endian\n");  
 }  
 方法三：  
 void checkCpuMode(void)  
    {  
  union   
  {  
   short s;  
   char c[sizeof(short)];  
  }un;  
  un.s=0x0102;  
  if(un.[0]==1&&un.c[1]==2)  
   printf("big endian\n");  
  else  
   printf("little endian\n");  
 }  
85、由C/C++编译的程序占用的内存分为以下几个部分   
 1、栈区(stack)： 由编译器自动分配释放 ，存放函数的参数值，局部变量的值等。其操作方式类似于数据结构中的栈。   
 2、堆区(heap)： 一般由程序员分配释放，若程序员不释放，程序结束时可能由OS回收。  
                 注意它与数据结构中的堆是两回事，分配方式倒是类似于链表。   
 3、全局区(static)： 全局变量和静态变量的存储是放在一块的，初始化的全局变量和静态变量在一块区域，  
                    未初始化的全局变量和未初始化的静态变量在相邻的另一块区域，程序结束后有系统释放 。   
 4、文字常量区： 常量字符串就是放在这里的， 程序结束后由系统释放。   
 5、程序代码区： 存放函数体的二进制代码。  
87、for(m=5;m--;m<0)  
 {  
  printf("m=%d\n",m);  
 }输出：4、3、2、1、0  
88、5["abcdef"]能够编译通过，请问编译后的结果是什么？  
 printf("%d\n",5["abcdef"]);输出'f'的ACSII值，如果是4["abcdef"]则输出'e'的ACSII的值。  
89、线程同步的方法：信号量、条件变量、互斥锁。  
90、for(i=0;i<2,i<3,i<4;i++)  
  printf("%d \n",i);     输出：0,1,2,3。

100、物理地址，虚拟地址，逻辑地址和总线地址的区别  
     逻辑地址（Logical Address）是指由程序产生的与段相关的偏移地址部分。  
 例如，你在进行C语言指针编程中，可以读取指针变量本身值(&amp;操作)，实际上这个值就是逻辑地址，  
 它是相对于你当前进程数据段的地址，不和绝对物理地址相干。只有在Intel实模式下，  
 逻辑地址才和物理地址相等（因为实模式没有分段或分页机制, Cpu不进行自动地址转换）；  
 逻辑也就是在Intel 保护模式下程序执行代码段限长内的偏移地址（假定代码段、数据段如果完全一样）。  
 应用程序员仅需与逻辑地址打交道，而分段和分页机制对您来说是完全透明的，仅由系统编程人员涉及。  
 应用程序员虽然自己可以直接操作内存，那也只能在操作系统给你分配的内存段操作。  
     线性地址（Linear Address）是逻辑地址到物理地址变换之间的中间层。程序代码会产生逻辑地址，  
 或者说是段中的偏移地址，加上相应段的基地址就生成了一个线性地址。如果启用了分页机制，  
 那么线性地址可以再经变换以产生一个物理地址。若没有启用分页机制，那么线性地址直接就是物理地址。  
 Intel 80386的线性地址空间容量为4G（2的32次方即32根地址总线寻址）。  
     物理地址（Physical Address） 是指出现在CPU外部地址总线上的寻址物理内存的地址信号，是地址变换的最终结果地址。  
 如果启用了分页机制，那么线性地址会使用页目录和页表中的项变换成物理地址。  
 如果没有启用分页机制，那么线性地址就直接成为物理地址了。  
  在x86下，外设的i/o地址是独立的，即有专门的指令访问外设i/o，i/o地址就是你所说的“总线地址”。  
 而“物理地址”就是ram地址。在arm中，i/o和ram统一编址，但linux为了统一各个平台，仍然保留这个概念，其实就是物理地址。    
101、编写内核程序中申请内存和编写应用程序时申请内存有什么区别  
     应用程序使用C函数库中的内存分配函数malloc()申请内存内核会为进程使用的代码和数据空间维护一个当前位置的值brk，  
 这个值保存在每个进程的数据结构中。它指出了进程代码和数据（包括动态分配的数据空间）在进程地址空间中的末端位置。  
 当malloc（）函数为程序分配内存时，它会通过系统调用brk（）把程序要求新增的空间长度通知内核，  
 内核代码从而可以根据malloc（）所提供的信息来更新brk的值，但此时并不为新申请的空间映射物理内存页面。  
 只有当程序寻址到某个不存在对应物理页面的地址时，内核才会进行相关物理内存页面的映射操作。  
     当用户使用内存释放函数free（）动态释放已申请的内存块时，c库中的内存管理函数就会把所释放的内存块标记为空闲，  
 以备程序再次申请内存时使用。在这个过程中内核为该进程所分配的这个物理页面并不会被释放掉。  
 只有当进程最终结束时内核才会全面收回已分配和映射到该进程地址空间范围内的所有物理内存页面。  
102、如何用C语言实现读写寄存器变量  
 #define rBANKCON0  (\*(volatile unsigned long \*)0x48000004)   
 rBankCON0 = 0x12;   
103、sscanf("123456 ", "%4s", buf);                     1234  
 sscanf("123456 asdfga","%[^ ]", buf);               123456  
    sscanf("123456aafsdfADDEFF", "%[1-9a-z]", buf);     123456aafsdf  
    sscanf("123afsdfADJKLJ", "%[^A-Z]", buf);          123afsdf  
104、char\* s="AAA";  
 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。   
105、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：  
  int do\_dup(int a[],int N)                 int do\_dup(int a[],int N)//a[0]为监视哨    
  {                                           {  
   int i;                                       int temp;   
   int s;                                       while (a[0]!=a[a[0]])  
   int num;                                     {  
   for(i=0;i<N;i++)                                 temp=a[0];  
    s+=a[i];                                     a[0]=a[temp];  
   num=s-N\*(N-1)/2;(num即为重复数)                  a[temp]=temp;           
  }                                                 }  
              return a[0];  
              }  
106、tcp/udp是属于哪一层？tcp/udp有何优缺点？  
 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  
107、 char a = 100;  
   char b =  150;//10010110//01101010  
   unsigned char c ;  
      c = (a < b)? a:b;   --》 c=150;  
108、应用程序ping发出的是ICMP请求报文  
109、在C语言中memcpy和memmove是一样的吗？  
 memmove()与memcpy()一样都是用来拷贝src所指向内存内容前n个字节到dest所指的地址上，不同是，当src和dest所指的内存区域重叠时，  
 memmove()仍然可以正确处理，不过执行效率上略慢些。  
110、C语言程序代码优化方法  
  \* 选择合适的算法和数据结构  
  \* 使用尽量小的数据类型  
  \* 使用自加、自减指令  
  \* 减少运算的强度  
   求余运算（a=a%8改为a=a&7）  
   平方运算（a=pow(a,2.0)改为a=a\*a）  
   用移位实现乘除法运算  
  \* 延时函数的自加改为自减  
  \* switch语句中根据发生频率来进行case排序  
   
111、找出一个字符串中一个最长的连续的数字，并标注出位置和个数。  
 void main()    
  {  
  char input[100];    
  char output[100] = {0};  
  int  count = 0, maxlen = 0, i = 0;  
  char \*in=input, \*out=output,\*temp=NULL,\*final=NULL;  
    
  printf("Please input string(length under 100):\n");  
  scanf("%s", input);  
  printf("Input string is %s\n", input);

  while(\*in!='\0')  
  {  
    if(\*in>='0'&&\*in<='9')  
    {  
      count=0;  
      temp=in;  
      for(;(\*in>='0')&&(\*in<='9');in++)  
     count++;  
      if (maxlen<count)  
      {  
     maxlen=count;  
     =temp;     
      }   
    }   
    in++;  
  }

  for(i=0; i<maxlen; i++)  
  \*(out++) = \*(final++);  
  \*out='\0';

  printf("Maxlen is %d\n", maxlen);  
  printf("Output is %s\n", output);  
}  
112、写出螺旋矩阵  
 void Matrix(int m,int n)  //顺时针  
 {  
  int i,j,a=1;  
  int s[100][100];  
  int small = (m<n)?m:n;  
  int k=small/2;  
  for(i=0;i<k;i++)  
  {  
   for(j=i;j<n-i-1;j++)  
    s[i][j]=a++;  
   for(j=i;j<m-i-1;j++)  
    s[j][n-i-1]=a++;  
   for(j=n-i-1;j>i;j--)  
    s[m-i-1][j]=a++;  
   for(j=m-i-1;j>i;j--)  
    s[j][i]=a++;  
  }  
  if(small & 1)  
  {  
   if(m<n)  
    for(i=k;i<n-k;++i)  
     s[k][i]=a++;  
   else  
    for(i=k;i<m-k;++i)  
     s[i][k]=a++;  
  }  
  for(i=0;i<m;i++)  
  {  
   for(j=0;j<n;j++)  
    printf("=",s[i][j]);  
   printf("\n");  
  }  
 }  
113、int \*a = (int \*)2;  
  printf("%d",a+3); 答案是2+3\*4=14；int类型地址加1 相当于加4个字节  
114、main()  
 {  
    
  char a,b,c,d;  
  scanf("%c%c",&a,&b);  
  c=getchar();d=getchar();  
  printf("%c%c%c%c\n",a,b,c,d);  
 } 输出：12  
    3  
115、int a[2] = {1, 2};  
 //指向常量的指针,指针可以变，指针指向的内容不可以变  
 const int \*p = a;//与int const \*p = a;等价  
 p++;//ok  
 \*p = 10;//error  
 //常指针,指针不可以变，指针指向的内容可以变  
 int\* const p2 = a;  
 p2++;//error  
 \*p2 = 10;//ok  
 //指向常量的常指针,都不可以改变  
 p3++;//error  
 \*p3 = 10;//error  
116、#error 预处理指令的作用是，编译程序时，只要遇到#error 就会生成一个编译错误提  
  示消息，并停止编译  
117、中断活动的全过程大致为：  
 1、中断请求：中断事件一旦发生或者中断条件一旦构成，中断源提交“申请报告”，  
   与请求CPU暂时放下目前的工作而转为中断源作为专项服务  
 2、中断屏蔽：虽然中断源提交了“申请报告”，但是，是否得到CPU的响应，  
   还要取决于“申请报告”是否能够通过2道或者3道“关卡”（中断屏蔽）送达CPU  
  （相应的中断屏蔽位等于1，为关卡放行；反之相应的中断屏蔽位等于0，为关卡禁止通行）；  
 3、中断响应：如果一路放行，则CPU响应中断后，将被打断的工作断点记录下来  
  （把断点地址保护到堆栈），挂起“不再受理其他申请报告牌”  
  （清除全局中断标志位GIE=0），跳转到中断服务子程序  
 4、保护现场：在处理新任务时可能破坏原有的工作现场，所以需要对工作现场和工作环境进行适当保护；  
 5、调查中断源：检查“申请报告”是由哪个中断源提交的，以便作出有针对性的服务；  
 6、中断处理：开始对查明的中断源进行有针对性的中断服务；  
 7、清除标志：在处理完毕相应的任务之后，需要进行撤消登记（清除中断标志），以避免造成重复响应；  
 8、恢复现场：恢复前面曾经被保护起来的工作现场，以便继续执行被中断的工作；  
 9、中断返回：将被打断的工作断点找回来（从堆栈中恢复断点地址），  
 并摘下“不再受理其他申请报告牌”（GIE＝1），继续执行原先被打断的工作。     
118、linux进程间通讯的几种方式的特点和优缺点，和适用场合。分类：  
 #管道( pipe )：管道是一种半双工的通信方式，数据只能单向流动，  
    而且只能在具有亲缘关系的进程间使用。进程的亲缘关系通常是指父子进程关系。    
 #有名管道(named pipe) ： 有名管道也是半双工的通信方式，但是它允许无亲缘关系进程间的通信。   
 #信号量( semophore ) ： 信号量是一个计数器，可以用来控制多个进程对共享资源的访问。  
   它常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源。因此，主要作  
   为进程间以及同一进程内不同线程之间的同步手段。    
 #消息队列( message queue ) ： 消息队列是由消息的链表，存放在内核中并由消息队列标识  
   符标识。消息队列克服了信号传递信息少、管道只能承载无格式字节流以及缓冲区大小受限等缺点。  
 #信号 ( sinal ) ： 信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生。   
 #共享内存( shared memory)：共享内存就是映射一段能被其他进程所访问的内存，  
   这段共享内存由一个进程创建，但多个进程都可以访问。共享内存是最快的IPC方式，  
   它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，  
   如信号量，配合使用，来实现进程间的同步和通信。   
 #套接字( socket ) ： 套解口也是一种进程间通信机制，与其他通信机制不同的是，  
   它可用于不同及其间的进程通信。  
119、关键字volatile有什么含意?并给出三个不同的例子。  
 一个定义为volatile的变量是说这变量可能会被意想不到地改变，这样，编译器就不会去假设  
 这个变量的值了。精确地说就是，优化器在用到这个变量时必须每次都小心地重新读取这个变量的值，  
 而不是使用保存在寄存器里的备份。  
 下面是volatile变量的几个例子：  
 1). 并行设备的硬件寄存器（如：状态寄存器）  
 2). 一个中断服务子程序中会访问到的非自动变量(Non-automatic variables)  
 3). 多线程应用中被几个任务共享的变量 回答不出这个问题的人是不会被雇佣的。我认为这是区分C程序员和嵌入式系统程序员的最基本的问题。嵌入式系统程序员经常同硬件、中断、RTOS等等打交道，所用这些都要求volatile变量。不懂得volatile内容将会带来灾难。  
 假设被面试者正确地回答了这是问题（嗯，怀疑这否会是这样），我将稍微深究一下，  
 看一下这家伙是不是直正懂得volatile完全的重要性。  
 1). 一个参数既可以是const还可以是volatile吗？解释为什么。  
 2). 一个指针可以是volatile 吗？解释为什么。  
 3). 下面的函数有什么错误：  
 int square(volatile int \*ptr) {  
    return \*ptr \* \*ptr;  
  }  
 下面是答案：  
 1). 是的。一个例子是只读的状态寄存器。它是volatile因为它可能被意想不到地改变。  
     它是const因为程序不应该试图去修改它。  
 2). 是的。尽管这并不很常见。一个例子是当一个中服务子程序修该一个指向一个buffer的指针时。  
 3). 这段代码的有个恶作剧。这段代码的目的是用来返指针\*ptr指向值的平方，但是，  
    由于\*ptr指向一个volatile型参数，编译器将产生类似下面的代码：  
 int square(volatile int \*ptr) {  
   int a,b;  
   a = \*ptr;  
   b = \*ptr;  
   return a \* b;  
  }  
  由于\*ptr的值可能被意想不到地该变，因此a和b可能是不同的。  
  结果，这段代码可能返不是你所期望的平方值！正确的代码如下：  
 long square(volatile int \*ptr) {  
   int a;  
   a = \*ptr;  
   return a \* a;  
  }  
  Volatile 关键字告诉编译器不要持有变量的临时性拷贝。一般用在多线程程序中，  
  以避免在其中一个线程操作该变量时，将其拷贝入寄存器。  
 请看以下情形: A线程将变量复制入寄存器，然后进入循环，反复检测寄存器的值是否满足  
 一定条件(它期待B线程改变变量的值。在此种情况下，当B线程改变了变量的值时，  
 已改变的值对其在寄存器的值没有影响。所以A线程进入死循环。  
  volatile 就是在此种情况下使用。    
120、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   
121、main()   
 {   
  char \*p1=“name”;   
  char \*p2;   
  p2=(char\*)malloc(20);   
  memset (p2, 0, 20);   
  while(\*p2++ = \*p1++);   
  printf(“%s\n”,p2);   
 }   
 答案：Answer:empty string. 因p2++已经指到了'\0'了;  
122、操作系统的内存分配一般有哪几种方式，各有什么优缺点？  
 定长和变长。  
 变长：内存时比较灵活，但是易产生内存碎片。  
 定长：灵活性差，但分配效率较高，不会产生内存碎片   
123、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   
124、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   
125、#define N 500  
 unsigned char count;  
 for(count=0;count < N;count++)  
 {  
  printf("---%d---\n",count);  
 }死循环，因为unsigned char 最大为255  
126、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   
127、ICMP（ICMP协议对于网络安全具有极其重要的意义）功能主要有：  
 · 侦测远端主机是否存在。  
 · 建立及维护路由资料。  
 · 重导资料传送路径。  
 · 资料流量控制。  
   
   
 单向、双向链表操作、写一个快速排序算法（原理：找一个基准值，分别将大于和小于基准值的数据放到基准值左右两边，即一次划分。由于处在两边的数据也是无序的，所以再用同样的划分方法对左右两边的序列进行再次划分，直到划分元素只剩1个时结束，）  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
1、单向链表逆序  
LONDE \*link\_reverse(LNODE \*head)  
{  
 LNODE \*pb,\*pt;  
 if(head == NULL)  
  return head;  
 pb = head->next;  
 head->next=NULL;  
 while(pb != NULL)  
 {  
  pt = pb->next;  
  pb->next = head;  
  head = pb;  
  pb = pt;  
 }  
 return head;  
}  
2、快速排序  
void quick\_sort(int num,int start\_num,int end\_num)  
{  
 if(start\_num < end\_num)  
 {  
  int i = start\_num;  
  int j = end\_num;  
  int temp = num[start\_num];  
  while(i < j)  
  {  
   while(i < j && num[j] < temp)  
    j--;  
   if(i < j)  
    num[i++] = num[j];//把小于基准值放在左边  
   while(i < j && num[i] >= temp)  
    i++;  
   if(i < j)  
    num[j--] = num[i];//把大于基准值放在右边  
  }  
  num[i] = temp;  
  quick\_sort(num,start\_num,i-1);  
  quick\_sort(num,i+1,end\_num);  
 }  
}  
3、//二分擦找  
 int binary\_search(int array[],int value,int size)  
{  
 int low=0,high=size-1,mid;  
   
 while(low<=high)    //只要高低不碰头就继续二分查找  
 {  
  mid=(low+high)/2;  
  if(value==array[mid])  //比较是不是与中间元素相等  
   return mid;  
  else if(value > array[mid]) //每查找一次，就判断一次所要查找变量所在范围，并继续二分  
   low=mid;     //如果大小中间值，下限移到中间的后一个位，上限不变，往高方向二分  
  else  
   high=mid;        //上限移到中间的前一个位，往低方向二分  
 }  
 return -1;  
}  
/\*双向循环链表插入函数\*/  
TYPE \*insert\_link(TYPE \*head,TYPE \*p\_in)  
{  
 TYPE \*p\_mov = head,p\_front = head;  
   
 if(head == NULL)  
 {  
  head = p\_in;  
  p\_in->next = head;  
  p\_perior = head;  
 }  
 else  
 {  
  while((p\_in->[] > p\_mov->[]) && (p\_mov->next != head))  
  {  
   p\_front = p\_mov;  
   p\_mov = p\_mov->next;  
  }  
    
  if(p\_in->[] <= p\_mov->[])  
  {  
   if(head == p\_mov)  
   {  
    p\_in->prior = head->prior;  
    head->prior->next = p\_in;  
    p\_in->next = p\_mov;  
    p\_mov->prior = p\_in;  
    head = p\_in;  
   }  
   else  
   {  
    pf->next = p\_in;  
    p\_in->prior = p\_front;  
    p\_in->next = p\_mov;  
    p\_mov->prior = p\_in;  
   }  
  }  
  else  
  {  
   p\_mov->next = p\_in;  
   p\_in->prior = p\_mov;  
   p\_in->next = head;  
   head->prior = p\_in;  
  }  
 }  
 return head;  
}

/\*双向链表删除函数\*/  
TYPE \*delete\_link(TYPE \*head,int num)  
{  
 TYPE \*p\_mov = head,p\_front = head;  
   
 if(head == NULL)  
  printf("Not link\n");  
 while((p\_mov->num != num) && (p\_mov->next != head))  
 {  
  p\_front = p\_mov;  
  p\_mov = p\_mov->next;  
 }  
 if(p\_mov->num == num)  
 {  
  if(p\_mov == head)  
  {  
   if(p\_mov->next == head && p\_mov->prior == head)  
   {  
    free(pb);  
    head =NULL;  
    return head;  
   }  
   head->next->prior = head->prior;  
   head->prior->next = head->next;  
   head = head->next;  
  }  
  else  
  {  
   p\_front->next = p\_mov->next;  
   p\_mov->next->prior = p\_front;  
  }  
  free(p\_mov);  
  printf("The node is delete\n");  
 }  
 else  
 {  
  printf("The node not been found\n");  
 }  
 return head;  
}