半又公司面试宝典

C/C++版本

编辑：蓦然回首

监制：杭州电大

2010-04-10

试题 1：C 语言面试题一——华为篇

1.static 有什么用途？（请至少说明两种）
1）限制变量的作用域
2）设置变量的存储域（堆，主动分配内存也是堆）

2.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。
3) 不存在指向空值的引用，但是存在指向空值的指针。

3.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

4.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，局部变量在栈

5.什么是平衡二叉树？
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

6.堆栈溢出一般是由什么原因导致的？
没有回收垃圾资源

7.什么函数不能声明为虚函数？
constructor (构造函数)

8.冒泡排序算法的时间复杂度是什么？(其它排序算法的时间复杂度)
O(n^2)

9.写出 float x 与“零值”比较的 if 语句。
if(x>0.000001&&x<-0.000001)

10.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

11.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。

19.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

```
#include <stdio.h>
#define NULL 0
#define TYPE struct stu
#define LEN sizeof (struct stu)
struct stu
```

```
        {
         int data;
          struct stu *next;
        };

TYPE *line(int n)
     {
         int sum=1;
        struct stu *head,*pf,*pb;
        int i;
        for(i=0;i<n;i++)
        {
          pb=(TYPE*) malloc(LEN);
          pb->data=sum;
          if (i==0)
            pf=head=pb;
           else
            pf->next=pb;
          if (i==(n-1))
          pb->next=head;
          else pb->next=NULL;
          pf=pb;
           sum++;
        }
        return(head);
    }

main()
{
  int M,N,x,i;
  struct stu *p,*q;

   printf("please scanf M and N (M<N)");
   scanf("%d %d",&M,&N);
   p=line(N);
   x=N;
   while(x)
   {
     for(i=1;i<M-1;i++)
        {
            p=p->next;
         }
         q=p->next;
         printf("%d\n",q->data)  ;
```

```
            p->next = p->next->next;
            p=p->next;
            free(q) ;
            x--;

     }
   getch();

}
```

20.不能做 switch()的参数类型是：
switch 的参数不能为实型。（只能是 int char）

试题 2：
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 atol()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);
}
```

4.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

5.直接链接两个信令点的一组链路称作什么?
PPP 点到点连接

6.接入网用的是什么接口?TCP/IP

7.voip 都用了那些协议?RTCP

8.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

9.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

11.unsigned char *p1;
unsigned long *p2;
p1=(unsigned char *)0x801000;
p2=(unsigned long *)0x810000;
请问 p1+5=？　;
p2+5=？　;
答：p1+5=0x801005　;
p2+5=0x810020　;

二.选择题:
1.Ethternet 链接到 Internet 用到以下那个协议? D
A.HDLC;B.ARP;C.UDP;D.TCP;E.ID

2.属于网络层协议的是:BC
A.TCP;B.IP;C.ICMP;D.X.25

3.Windows 消息调度机制是:c
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>>4)%256
}
请问 hash(16),hash(256)的值分别是:A

A.1.16;B.8.32;C.4.16;D.1.32

三.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

2.

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

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

死循环(unsigned char 0 到 255)

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;
    }
    .......
  }
```

试题 3：华为全套完整试题

高级题

1、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)
}
```

2、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;
}
```

3、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;
}
```

试题 4:（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';
}
```

4、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。

试题 5：华为软件研发面试题 2

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 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，假定你将那个编写错了，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、请写出下列代码的输出内容

```
#include <stdio.h>
int main(void)
{
    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

5、static 全局变量与普通的全局变量有什么区别？static 局部变量和普通局部变量有什么区别？static 函数与普通函数有什么区别？

答: 1) 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。

2) 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。
3) 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
综上所述:
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

6、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。
7、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 data)+sizeof(max));的执行结果是：
考点:区别 struct 与 union.(一般假定在 32 位机器上)
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

8、队列和栈有什么区别?
   队列先进先出，栈后进先出

9、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)
  {
        FUNC1 p=&inc;
        int temp =p(arg1);
        fun(&temp,&arg1, arg2);
        printf("%dn",*arg2);
    }
main()
{
       int a;            //局部变量 a 为 0;
        show(multi,10,&a);
        return 0;
    }
```

答：110

10、请找出下面代码中的所有错误 (题目不错,值得一看)

说明：以下代码是把一个字符串倒序，如“abcd”倒序后变为“dcba”

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

答：

方法 1：一共有 4 个错误;

```
int main()
{
  char* src = "hello,world";
  int len = strlen(src);
  char* dest = (char*)malloc(len+1);//要为分配一个空间    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("%sn",dest);
    free(dest);        // 使用完，应当释放空间，以免造成内存汇泄露
    dest = NULL;     //防止产生野指针
    return 0;
  }
```

方法 2： (方法一需要额外的存储空间,效率不高.) 不错的想法

```
#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]; //小心一点
      str[len-i-1]=t;
  }
  printf("%s",str);
  return 0;
}
```

11.对于一个频繁使用的短小函数,在 C 语言中应用什么实现,在 C++中应用什么实现?

c 用宏定义，c++用 inline

12.直接链接两个信令点的一组链路称作什么?

PPP 点到点连接

13.接入网用的是什么接口?

V5 接口

14.voip 都用了那些协议?

H.323 协议簇、SIP 协议、Skype 协议、H.248 和 MGCP 协议

15.软件测试都有那些种类?

黑盒：针对系统功能的测试白盒：测试函数功能，各函数接口

16.确定模块的功能和模块的接口是在软件设计的那个队段完成的?

概要设计阶段

17.

```
unsigned char *p1;
```

```
unsigned long *p2;
p1=(unsigned char *)0x801000;
p2=(unsigned long *)0x810000;
```
请问 p1+5= ;
p2+5= ;
答案:0x801005(相当于加上 5 位) 0x810014(相当于加上 20 位);

选择题:
21.Ethternet 链接到 Internet 用到以下那个协议? D
A.HDLC;B.ARP;C.UDP;D.TCP;E.ID

22.属于网络层协议的是:(B C)
A.TCP;B.IP;C.ICMP;D.X.25

23.Windows 消息调度机制是:(C)
A.指令队列;B.指令堆栈;C.消息队列;D.消息堆栈;

找错题:
25.请问下面程序有什么错误?
```
int a[60][250][1000],i,j,k;
for(k=0;kMax_GT_Length)
{         return GT_Length_ERROR;
  }       ..... }
```
答: 死循环//

问答题:
29.IP Phone 的原理是什么?
IP 电话（又称 IP PHONE 或 VoIP）是建立在 IP 技术上的分组化、数字化传输技术,其基本原理是:通过语音压缩算法对语音数据进行压缩编码处理,然后把这些语音数据按 IP 等相关协议进行打包,经过 IP 网络把数据包传输到接收地,再把这些语音数据包串起来,经过解码解压处理后,恢复成原来的语音信号,从而达到由 IP 网络传送语音的目的。

30.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

31.1 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?
1 号信令接续慢，但是稳定，可靠。
7 号信令的特点是:信令速度快，具有提供大量信令的潜力，具有改变和增加信令的灵活性，便于开放新业务，在通话时可以随意处理信令，成本低。目前得到广泛应用。

32.列举 5 种以上的电话新业务
如“闹钟服务”、“免干扰服务”、“热线服务”、“转移呼叫”、“遇忙回叫”、“缺席用户服务”、“追查恶意呼叫”、“三方通话”、“会议电话”、“呼出限制”、“来电显示”、“虚拟网电话”等。

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;
```
答: 把循环语句内外换一下

2.
```
#define Max_CB 500
void LmiQueryCSmd(Struct MSgCB * pmsg)
{
    unsigned char ucCmdNum;
    ......
    for(ucCmdNum=0;ucCmdNum<Max_CB;ucCmdNum++)
    {
    ......;
    }
}
```
答: 死循环,unsigned int 的取值范围是 0~255

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++);
```
答:结果与编译器相关,得到的可能不是平方值.

试题 6:微软亚洲技术中心的面试题!!!

1. 进程和线程的差别。

答: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

答: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

3. Heap 与 stack 的差别。

答: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

4．Windows 下的内存是如何管理的？
分页管理

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’？
答:键盘钩子 SetWindowsHookEx

14．存储过程是什么？有什么用？有什么优点？
答: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？
答：Template 可以独立于任何特定的类型编写代码,是泛型编程的基础.
当我们编写的类和函数能够多态的用于跨越编译时不相关的类型时,用 Template.
模板主要用于 STL 中的容器,算法,迭代器等以及模板元编程.
(C++的 template 是实现在库设计和嵌入式设计中的关键。
template 能实现抽象和效率的结合；同时 template 还能有效地防止代码膨胀)

16．谈谈 Windows DNA 结构的特点和优点。
答:Windows Distributed interNet Application Architecture（Windows 分布式应用结构，简称 Windows DNA）是微软创建新一代高适应性商业解决方案的框架，它使公司能够充分地挖掘数字神经系统的优点。Windows DNA 是第一个将 Internet、客户/服务器、和用于计算的 PC 模型结合并集成在一起的为新一类分布式计算方案而设计的应用软件体系结构

17．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？
答: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。

试题 7：思科

1．用宏定义写出 swap（x，y）
答#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：
int do_dup(int a[],int N)
答:int do_dup(int a[],int N)　　//未经调试
```
{
      int sun = 0;
      int sum2;
      for(int i=0;i<N;++i)
      {
        Sum+=a[i];
      }
      Sum2 = (1+N-1)*N/2;
      Return (sum-sum2);
}
```

3 一语句实现 x 是否为 2 的若干次幂的判断
答:方法 1)int i = 512;
cout << boolalpha << ((i & (i - 1)) ? false : true) << endl; //位与为 0,则表示是 2 的若干次幂
　　2)return (x>>N==1);

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
答:unsigned int intvert(unsigned int x,int p,int n) //假定 p=4,n=3
```
{
unsigned int _t = 0;
unsigned int _a = 1;
for(int i = 0; i < n; ++i)//循环的目的主要是-t
```

```
{
    _t |= _a;         //位或
    _a = _a << 1;
}
_t = _t << p;    //转换后_t 变为 1110000
x ^= _t;         /异或,将原来的位取反
return x;
}
```

试题 8：慧通：

1．什么是预编译，何时需要预编译：
答：就是指程序执行前的一些预处理工作,主要指#表示的.
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。

2．下述三个有什么区别？
```
char * const p;
char const * p
const char *p
```
解答：
```
char * const p; //常量指针，p 的值不可以修改
char const * p；//指向常量的指针，指向的常量值不可以改
const char *p；  //和 char const *p
```

3．解释下列输出结果
```
char str1[] = "abc";
char str2[] = "abc";
const char str3[] = "abc";
const char str4[] = "abc";
const char *str5 = "abc";
const char *str6 = "abc";
char *str7 = "abc";
char *str8 = "abc";
cout << ( str1 == str2 ) << endl;
cout << ( str3 == str4 ) << endl;
cout << ( str5 == str6 ) << endl;
cout << ( str7 == str8 ) << endl;
```
结果是：0 0 1 1
解答：str1,str2,str3,str4 是数组变量，它们有各自的内存空间；
而 str5,str6,str7,str8 是指针，它们指向相同的常量区域。

4．以下代码中的两个 sizeof 用法有问题吗？[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 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。

注意:数组名作为函数参数时,退化为指针.

数组名作为 sizeof()参数时,数组名不退化,因为 sizeof 不是函数.

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。

5．指出下面代码的输出，并解释为什么。(不错,对地址掌握的深入挖潜)

```
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].

6.请问以下代码有什么问题：

1).

```
int main()
{
    char a;
    char *str=&a;
    strcpy(str,"hello");
    printf(str);
    return 0;
}
```

答;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。

Strcpy 的在库函数 string.h 中.程序的主要错误在于越界进行内存读写导致程序崩溃//

2).

```
char* s="AAA";
printf("%s",s);
s[0]='B';
printf("%s",s);
```

有什么错？

答: "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。

试题 9：

1、写一个“标准”宏，这个宏输入两个参数并返回较小的一个。

答:#define Min(X, Y) ((X)>(Y)?(Y):(X)) //结尾没有;

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。

答:while(1){}或者 for(;;) //前面那个较好

3、关键字 static 的作用是什么？

答:1)定义静态局部变量,作用域从函数开始到结束.

2) 在模块内的 static 函数只可被这一模块内的其它函数调用，这个函数的使用范围被限制在声明它的模块内；

3) 在类中的 static 成员变量属于整个类所拥有，对类的所有对象只有一份拷贝

4、关键字 const 有什么含意？

答 :1)表示常量不可以修改的变量。

2)可以修饰参数,作为输入参数.

3)修饰函数,防止以外的改动.

4)修饰类的成员函数,不改变类中的数据成员.

5、关键字 volatile 有什么含意？并举出三个不同的例子？
答: 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。
例子: 硬件时钟;多线程中被多个任务共享的变量等

6. 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。

试题 10：
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？
答:*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;
答: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?
答: 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);
}
int main( )
{
```

```
    char *str=NULL;
    getmemory(str);
    strcpy(p,"hello world");
    printf("%s/n",str);
    free(str);
    return 0;
}
```
答: 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？
答;正常输出,长度不一样，会造成非法的 OS,覆盖别的内容.

6.列举几种进程的同步机制，并比较其优缺点。
答:原子操作
信号量机制
自旋锁
管程，会合，分布式系统

7.进程之间通信的途径
答 共享存储系统
消息传递系统
管道：以文件系统为基础

试题 11：面试经典试题
1 编程基础
1.1 基本概念
1. const 的理解：const char*, char const*, char*const 的区别问题几乎是 C++面试中每次 都会有的题目。 事实上这个概念谁都有只是三种声明方式非常相似很容易记混。Bjarne 在他的 The C++ Programming Language 里面给出过一个助记的方法： 把一个声明从右向左读。
char * const cp; (* 读成 pointer to)
cp is a const pointer to char
const char * p;
p is a pointer to const char;
char const * p;
同上因为 C++里面没有 const*的运算符，所以 const 只能属于前面的类型。

2. c 指针
int *p[n];-----指针数组，每个元素均为指向整型数据的指针。
int (*p)[n];------p 为指向一维数组的指针，这个一维数组有 n 个整型数据。
*p+1; 指向数组的第二个元素
(*p+0)+1; 指向数组的第二个元素

p + 1　　指针指向下一个数组的首元素
int *p();----------函数带回指针，指针指向返回的值。
int (*p)();------p 为指向函数的指针。

3. 数组越界问题 (这个题目还是有点小险的)
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;
for (i=0;i<=MAX;i++)
A[i]=i;
}
```
解答：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.
注：char 类型为一个字节，取值范围是[-128，127]，unsigned char [0 ,255]

4. C++:memset ,memcpy 和 strcpy 的根本区别?
答:#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''；例:char a[100];memset(a, '', 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);
```
    {
       ASSERT((dest!=NULL)&&(src!=NULL));
       Char *address = dest;
       While((*dest++=*src++)!=' \0' )
          Continue;
       Return dest;
    }
```
用法：#include <string.h>
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);
```
    {
```

```
    ASSERT((dest!=NULL)&&(src!=NULL));
    ASSERT((dest>src+count)||(src>dest+count));// 防止内存重叠，也可以用
restrict 修饰指针
    Byte* bdest = (Byte*)dest;
    Byte* bsrc = (Byte*) src;
    While(count-->0)
      *bdest++ = **bsrc++;
    Return dest;
   }
```

用法：#include <memory.h>
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。

5. ASSERT()是干什么用的
答: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。

6. system（"pause");作用?
答:系统的暂停程序，按任意键继续，屏幕会打印，"按任意键继续。。。。。" 省去了使用 getchar（）;

7. 请问 C++的类和 C 里面的 struct 有什么区别?
答:c++中的类具有成员保护功能，并且具有继承，多态这类 oo 特点，而 c 里的 struct 没有 c 里面的 struct 没有成员函数,不能继承,派生等等.

8. 请讲一讲析构函数和虚函数的用法和作用?
答:析构函数也是特殊的类成员函数，它没有返回类型，没有参数，不能随意调用，也没有重载。只是在类对象生命期结束的时候，由系统自动调用释放在构造函数中分配的资源。这种在运行时，能依据其类型确认调用那个函数的能力称为多态性，或称迟后联编。另： 析

构函数一般在对象撤消前做收尾工作，比如回收内存等工作，
虚拟函数的功能是使子类可以用同名的函数对父类函数进行覆盖，并且在调用时自动调用子类覆盖函数，如果是纯虚函数，则纯粹是为了在子类覆盖时有个统一的命名而已。
注意:子类重新定义父类的虚函数的做法叫覆盖,override,而不是 overload(重载),重载的概念不属于面向对象编程,重载指的是存在多个同名函数,这些函数的参数表不同..重载是在编译期间就决定了的,是静态的,因此,重载与多态无关.与面向对象编程无关.
含有纯虚函数的类称为抽象类,不能实例化对象,主要用作接口类//

9．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;全局变量的生命周期是整个程序运行的时间，而局部变量的生命周期则是局部函数或过程调用的时间段。其实现是由编译器在编译时采用不同内存分配方法。
全局变量在 main 函数调用后，就开始分配，
静态变量则是在 main 函数前就已经初始化了。
局部变量则是在用户栈中动态分配的（还是建议看编译原理中的活动记录这一块）

10．8086 是多少位的系统？在数据总线上是怎么实现的？
答:8086 系统是 16 位系统，其数据总线是 20 位。

12 程序设计
1．编写用 C 语言实现的求 n 阶阶乘问题的递归算法：
答:long int fact(int n)
{
If(n==0||n==1)
   Return 1;
Else
 Return n*fact(n-1);
}

2．二分查找算法：
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;
}
```

3. 递归计算如下递归函数的值（斐波拉契）：

```
f(1)=1
f(2)=1
f(n)=f(n-1)+f(n-2) n>2
```

解：非递归算法:

```
int f(int n)
{
    int i,s,s1,s2;
    s1=1;/*s1 用于保存 f(n-1)的值*/
    s2=1;/*s2 用于保存 f(n-2)的值*/
    s=1;
    for(i=3;i<=n;i++)
    {
        s=s1+s2;
        s2=s1;
        s1=s;
    }
    return(s);
}
```

递归算法:

```
Int f(int n)
{
 If(n==1||n==2)
   Rerurn 1;
 Else
   Rerutn f(n-1)+f(n-2);
}
```

4. 交换两个数，不用第三块儿内存：

答:

```
int a = ……;
int b = ……;
a = a + b;
b = a - b;
```

```
a = a - b;
```

5. 冒泡排序：

答:
```
void BubbleSort(elemtype x[],int n)    //时间复杂度为 O(n*n);
{
int i,j;
elemtype temp;
for(i=1;i<n;i++)
for(j=0;j<n-i;j++)
{
    if(x[j].key>x[j+1].key)
    {
        temp=x[j];
        x[j]=x[j+1];
        x[j+1]=temp;
    }
}
}
```

//补充一个改进的冒泡算法:

```
 void BubbleSort(elemtype x[],int n)
{
 Int i,j;
 BOOL exchange;  //记录交换标志
for(i=1;i<n;++i)     //最多做 n-1 趟排序
 {
Exchange = false;
For(j=n-1;j>=i;--j)
{
 If(x[j]>x[j+1])
 {
    x[0] = x[j];
    X[j] = x[j+1];
    X[j+1] = x[0];
    Exchange = true;    //发生了交换,设置标志为真.
 }
}
if (!Exchange )       //为发生替换,提前终止算法
return;

 }
}
```

7. 类的知识 (非常不错的一道题目)..

C＋＋

```
#include <iostream.h>
class human
{
    public:
    human(){ human_num++;
}; //默认构造函数
static int human_num;     //静态成员
~human()
{
    human_num--;
    print();
}
void print()             //
{
    cout<<"human num is: "<<human_num<<endl;
}
protected:
private:
};
int human::human_num = 0;   //类中静态数据成员在外部定义,仅定义一次
human f1(human x)
{
    x.print();
    return x;
}
int main(int argc, char* argv[])
{
    human h1; //调用默认构造函数,human_num 变为 1
    h1.print(); // 打印 Human_man:1
    human h2 = f1(h1); //先调用函数 f1(),输出 human_num:1,而后输出 human_num 为 0,
    h2.print();//打印输出:human_num:0
    return 0;
} //依次调用两个析构函数:输出:human_num:-1,human_num:-2;
```

输出：

1

1

0

0

－1

－2

－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－

分析：

```
human h1; //调用构造函数,---hum_num = 1;
h1.print(); //输出:"human is 1"
```

human h2 = f1(h1); //再调用 f1(h1)的过程中,由于函数参数是按值传递对象,调用默认的复制构造函数,h2 并没有调用定义的构造函数.

试题 11：C/C++ 程序设计员应聘常见面试试题深入剖析

1.找错题

试题 1：

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

试题 2：

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

试题 3：

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

解答：

试题 1 字符串 str1 需要 11 个字节才能存放下（包括末尾的''），而 string 只有 10 个字节的空间，strcpy 会导致数组越界；

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

对试题 3，if(strlen(str1) <= 10)应改为 if(strlen(str1) < 10)，因为 strlen 的结果未统计'' 所占用的 1 个字节。

剖析：

考查对基本功的掌握：

(1)字符串以’\0’结尾；

(2)对数组越界把握的敏感度；

(3)库函数 strcpy 的工作方式，如果编写一个标准 strcpy 函数的总分值为 10，下面给出几个不同得分的答案：

2 分

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

4 分

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

7 分

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

10 分

```
//为了实现链式操作，将目的地址返回，加 3 分!

char * strcpy( char *strDest, const char *strSrc )
{
  assert( (strDest != NULL) && (strSrc != NULL) );
  char *address = strDest;
  while( (*strDest++ = * strSrc++) != '' );
    return address;
}
```

从 2 分到 10 分的几个答案我们可以清楚的看到，小小的 strcpy 竟然暗藏着这么多玄机，真不是盖的！需要多么扎实的基本功才能写一个完美的 strcpy 啊！

(4)对 strlen 的掌握，它没有包括字符串末尾的''。

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

```
{
```

```
  assert( strt != NULL ); //断言字符串地址非0
  int len;
  while( (*str++) != '' )
  {
    len++;
  }
  return len;
}
```

试题4：

```
void GetMemory( char *p )
{
  p = (char *) malloc( 100 );
}

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

试题5：

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

void Test( void )
{
  char *str = NULL;
  str = GetMemory();
  printf( str );
}
```

试题6：

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

void Test( void )
```

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

试题 7：

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

解答：

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

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

后的 str 仍然为 NULL；

试题 5 中

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

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

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

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

后未判断内存是否申请成功，应加上：

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

试题 7 存在与试题 6 同样的问题，在执行

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

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

```
str = NULL;
```

试题 6 的 Test 函数中也未对 malloc 的内存进行释放。

剖析：

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

对内存操作的考查主要集中在：

（1）指针的理解；

（2）变量的生存期及作用范围；

（3）良好的动态内存申请和释放习惯。

再看看下面的一段程序有什么错误：

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

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

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

3.内功题

试题 1：分别给出 BOOL，int，float，指针变量 与“零值”比较的 if 语句（假设变量名为 var）

解答：

BOOL 型变量：if(!var)

int 型变量： if(var==0)

float 型变量：

```
const float EPSINON = 0.00001;
if ((x >= - EPSINON) && (x <= EPSINON)
```

指针变量： if(var==NULL)

剖析：

考查对 0 值判断的“内功”，BOOL 型变量的 0 判断完全可以写成 if(var==0)，而 int 型变量也可以写成 if(!var)，指针变量的判断也可以写成 if(!var)，上述写法虽然程序都能正确运行，但是未能清晰地表达程序的意思。

一般的，如果想让 if 判断一个变量的“真”、“假”，应直接使用 if(var)、if(!var)，

表明其为“逻辑”判断；如果用 if 判断一个数值型变量(short、int、long 等)，应该用 if(var==0)，表明是与 0 进行“数值”上的比较；而判断指针则适宜用 if(var==NULL)，这是一种很好的编程习惯。

浮点型变量并不精确，所以不可将 float 变量用“==”或“！=”与数字比较，应该设法转化成“>=”或“<=”形式。如果写成 if (x == 0.0)，则判为错，得 0 分。

试题 2：以下为 Windows NT 下的 32 位 C++程序，请计算 sizeof 的值

```
void Func ( char str[100] )
{
  sizeof( str ) = ?
}

void *p = malloc( 100 );
sizeof ( p ) = ?
```

解答：

```
sizeof( str ) = 4
sizeof ( p ) = 4
```

剖析：

Func (char str[100])函数中数组名作为函数形参时，在函数体内，数组名失去了本身的内涵，仅仅只是一个指针；在失去其内涵的同时，它还失去了其常量特性，可以作自增、自减等操作，可以被修改。

数组名的本质如下：

（1）数组名指代一种数据结构，这种数据结构就是数组；

例如：

```
char str[10];
cout << sizeof(str) << endl;
```

输出结果为 10，str 指代数据结构 char[10]。

（2）数组名可以转换为指向其指代实体的指针，而且是一个指针常量，不能作自增、自减等操作，不能被修改；

```
char str[10];
str++; //编译出错，提示 str 不是左值
```

（3）数组名作为函数形参时，沦为普通指针。

Windows NT 32 位平台下，指针的长度（占用内存的大小）为 4 字节，故 sizeof(str) 、sizeof (p) 都为 4。

试题 3：写一个“标准”宏 MIN，这个宏输入两个参数并返回较小的一个。另外，当你写下面的代码时会发生什么事？

```
least = MIN(*p++, b);
```

解答：

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

MIN(*p++, b)会产生宏的副作用

剖析：

这个面试题主要考查面试者对宏定义的使用，宏定义可以实现类似于函数的功能，但是它终归不是函数，而宏定义中括弧中的“参数”也不是真的参数，在宏展开的时候对“参数”进行的是一对一的替换。

程序员对宏定义的使用要非常小心，特别要注意两个问题：

（1）谨慎地将宏定义中的“参数”和整个宏用用括弧括起来。所以，严格地讲，下述解答：

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

都应判0分；

（2）防止宏的副作用。

宏定义#define MIN(A,B) ((A) <= (B) ? (A) : (B))对MIN(*p++, b)的作用结果是：

```
((*p++) <= (b) ? (*p++) : (*p++))
```

这个表达式会产生副作用，指针p会作三次++自增操作。

除此之外，另一个应该判0分的解答是：

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

这个解答在宏定义的后面加“;”，显示编写者对宏的概念模糊不清，只能被无情地判0分并被面试官淘汰。

试题4：为什么标准头文件都有类似以下的结构？

```
#ifndef __INCvxWorksh
#define __INCvxWorksh
#ifdef __cplusplus

extern "C" {
#endif
/*...*/
#ifdef __cplusplus
}

#endif
#endif /* __INCvxWorksh */
```

解答：

头文件中的编译宏

```
#ifndef  __INCvxWorksh
```

```
#define __INCvxWorksh
#endif
```

的作用是防止被重复引用。

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 symbol 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 symbol 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。_foo_int_int 这样的名字包含了函数名和函数参数数量及类型信息，C++就是考这种机制来实现函数重载的。

为了实现 C 和 C++的混合编程，C++提供了 C 连接交换指定符号 extern "C"来解决名字匹配问题，函数声明前加上 extern "C"后，则编译器就会按照 C 语言的方式将该函数编译为_foo，这样 C 语言中就可以调用 C++的函数了。

试题 5：编写一个函数，作用是把一个 char 组成的字符串循环右移 n 个。比如原来是“abcdefghi”如果 n=2，移位后应该是“hiabcdefgh”

函数头是这样的：

```
//pStr 是指向以''结尾的字符串的指针
//steps 是要求移动的 n

void LoopMove ( char * pStr, int steps )
{
  //请填充...
}
```

解答：

正确解答 1：

```
void LoopMove ( char *pStr, int steps )
{
  int n = strlen( pStr ) - steps;
  char tmp[MAX_LEN];
  strcpy ( tmp, pStr + n );
  strcpy ( tmp + steps, pStr);
  *( tmp + strlen ( pStr ) ) = '';
  strcpy( pStr, tmp );
}
```

正确解答 2：

```
void LoopMove ( char *pStr, int steps )
{
  int n = strlen( pStr ) - steps;
  char tmp[MAX_LEN];
  memcpy( tmp, pStr + n, steps );
  memcpy(pStr + steps, pStr, n );
  memcpy(pStr, tmp, steps );
```

}

剖析：

这个试题主要考查面试者对标准库函数的熟练程度，在需要的时候引用库函数可以很大程度上简化程序编写的工作量。

最频繁被使用的库函数包括：

（1） strcpy

（2） memcpy

（3） memset

试题 6：请说出 static 和 const 关键字尽可能多的作用

解答：

static 关键字至少有下列 n 个作用：

（1）函数体内 static 变量的作用范围为该函数体，不同于 auto 变量，该变量的内存只被分配一次，因此其值在下次调用时仍维持上次的值；

（2）在模块内的 static 全局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；

（3）在模块内的 static 函数只可被这一模块内的其它函数调用，这个函数的使用范围被限制在声明它的模块内；

（4）在类中的 static 成员变量属于整个类所拥有，对类的所有对象只有一份拷贝；

（5）在类中的 static 成员函数属于整个类所拥有，这个函数不接收 this 指针，因而只能访问类的 static 成员变量。

const 关键字至少有下列 n 个作用：

（1）欲阻止一个变量被改变，可以使用 const 关键字。在定义该 const 变量时，通常需要对它进行初始化，因为以后就没有机会再去改变它了；

（2）对指针来说，可以指定指针本身为 const，也可以指定指针所指的数据为 const，或二者同时指定为 const；

（3）在一个函数声明中，const 可以修饰形参，表明它是一个输入参数，在函数内部不能改变其值；

（4）对于类的成员函数，若指定其为 const 类型，则表明其是一个常函数，不能修改类的成员变量；

（5）对于类的成员函数，有时候必须指定其返回值为 const 类型，以使得其返回值不为“左值”。例如：

```
const classA operator*(const classA& a1,const classA& a2);
```

operator*的返回结果必须是一个 const 对象。如果不是，这样的变态代码也不会编译出错：

```
classA a, b, c;
(a * b) = c; // 对 a*b 的结果赋值
```

操作(a * b) = c 显然不符合编程者的初衷，也没有任何意义。

剖析：

惊讶吗？小小的 static 和 const 居然有这么多功能，我们能回答几个？如果只能回答 1~2 个，那还真得闭关再好好修炼修炼。

这个题可以考查面试者对程序设计知识的掌握程度是初级、中级还是比较深入，没有一

定的知识广度和深度，不可能对这个问题给出全面的解答。大多数人只能回答出 static 和 const 关键字的部分功能。

试题 7：写一个函数返回 1+2+3+…+n 的值（假定结果不会超过长整型变量的范围）

解答：

```
int Sum( int n )
{
  return ( (long)1 + n) * n / 2;    //或 return (1l + n) * n / 2;
}
```

剖析：

对于这个题，只能说，也许最简单的答案就是最好的答案。下面的解答，或者基于下面的解答思路去优化，不管怎么“折腾”，其效率也不可能与直接 return (1 l + n) * n / 2 相比！

```
int Sum( int n )
{
  long sum = 0;
  for( int i=1; i<=n; i++ )
  {
    sum += i;
  }
  return sum;
}
```

所以程序员们需要敏感地将数学等知识用在程序设计中。

终于明白了：按值传递意味着当将一个参数传递给一个函数时，函数接收的是原始值的一个副本。因此，如果函数修改了该参数，仅改变副本，而原始值保持不变。按引用传递意味着当将一个参数传递给一个函数时，函数接收的是原始值的内存地址，而不是值的副本。因此，如果函数修改了该参数，调用代码中的原始值也随之改变。

不管是在 c/c++中还是在 java 函数调用都是传值调用，.

当参数是对象的时候，传递的是对象的引用，这个和 c/c++传递指针是一个道理，在函数中改变引用本身，不会改变引用所指向的对象.

华为面试题

试题 12：金山公司几道面试题

1、 In C++, there're four type of Casting Operators, please enumerate and explain them especially the difference.

解析： C++类型转换问题

答案： reinterpret_cast,static_cast,const_cast,dynamic_cast

static_cast 数制转换

dynamic_cast 用于执行向下转换和在继承之间的转换

const_cast 去掉 const

reinterpret_cast 用于执行并不安全的 orimplmentation_dependent 类型转换

2、 以下代码有什么问题，如何修改？

```
#include <iostream>
#include <vector>
using namespace std;
void print(vector<int>);
int main()
{
    vector<int> array;
    array.push_back(1);
    array.push_back(6);
    array.push_back(6);
    array.push_back(3);
    //删除 array 数组中所有的 6
    vector<int>::iterator itor;
    vector<int>::iterator itor2;
    itor=array.begin();

    for(itor=array.begin(); itor!=array.end(); )
    {
        if(6==*itor)
        {
            itor2=itor;
            array.erase(itor2);
        }
        itor++;
    }
    print(array);
    return 0;
}
void print(vector<int> v)
{
    cout << "n vector size is: " << v.size() << endl;
    vector<int>::iterator p = v.begin();
}
```

我的答案是，迭代器问题，只能删除第一个 6，以后迭代器就失效了，不能删除之后的元素。但我不知道怎么改

```
void print(const vector<int>&);
int main()
{
    vector<int> array;
    array.push_back(1);
    array.push_back(6);
    array.push_back(6);
    array.push_back(3);
```

```
    //删除 array 数组中所有的 6
    array.erase( remove( array.begin(), array.end(), 6 ) , array.end() );

    print(array);
    return 0;
}
void print(const vector<int>& v)
{
    cout << "n vector size is: " << v.size() << endl;
    copy(v.begin(), v.end(), ostream_iterator<int>(cout, " ") );
}
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> array;
    array.push_back(1);
    array.push_back(6);
    array.push_back(6);
    array.push_back(6);
    array.push_back(6);
    array.push_back(6);
    array.push_back(3);
    array.push_back(9);
    array.push_back(8);
    array.push_back(5);
    //&Eacute;&frac34;&sup3;&yacute;array&Ecirc;&yacute;×é&Ouml;&ETH;&Euml;ù&O
    acute;&ETH;&micro;&Auml;6
    vector<int>::iterator itor;
    itor=array.begin();
    for(itor=array.begin(); itor!=array.end();++itor )
    {
        if(6==*itor)
        {
          itor=array.erase(itor);
          --itor;
        }
    }
    cout << "vector size is: " << array.size() << endl;
    for(itor=array.begin(); itor!=array.end();++itor )
    {
        cout<<*itor<<" ";
    }
```

```
    system("pause");
    return 0;
}
```

答案： 执行 itor=array.erase(itor);这句话后，itor 不会移动，而只是把删除的数后面的数都往前移一位，所以删除了第一个 6 后，指针指向第 2 个 6，然后在来个 itor++，指针就指向 array.end()了，给你画个草图：

1 6 6 3 array.end() //最开始指针 itor 指向第一个 6；

1 6 3 array.end() //删除第一个 6 后，指向第二个 6

1 6 3 array.end() //itor++后，就指向 3 了，所以不能删除

3. What are three ways in which a thread can enter the waiting state?

答:

CPU 调度给优先级更高的 thread，原先 thread 进入 waiting

阻塞的 thread 获得资源或者信号，进入 waiting

一、单项选择题：（共 12 题，每题 2 分，共 24 分）

1. 下面哪一个不是 C++的标准数据类型？ （ D）

A. int B. char

C. bool D. real

2. break 关键字在哪一种语法结构中不能使用？ （C）

A. for 语句 B. switch 语句

C. if 语句 D. while 语句

3. 类的继承方式有几种？ （B ）

A. 两种 B. 三种

C. 四种 D. 六种

4. extern 关键字的作用是什么？ （D）

A. 声明外部链接 B. 声明外部头文件引用

C. 声明使用扩展 C++语句 D. 声明外部成员函数、成员数据。

5. C 库函数 strstr 的功能是？ （A ）

A. 查找子串 B. 计算字符串长度

C. 字符串比较 D. 连结字符串

6. stl::deque 是一种什么数据类型？ （A ）

A. 动态数组 B. 链表

C. 堆栈 D. 树

7. STL 库里含有下面的哪一种泛型算法？ （D ）

A. KMP 查找 B. 折半查找

C. 冒泡排序 D. 快速排序

8. 现在最快且最通用的排序算法是什么？ （A ）

A. 快速排序 B. 冒泡排序
C. 选择排序 D. 外部排序

9. Win32 下的线程的哪一种优先级最高？ （ C）
A. THREAD_PRIORITY_HIGHEST 高优先级
B. THREAD_PRIORITY_IDLE 最低优先级,仅在系统空闲时执行
C. THREAD_PRIORITY_TIME_CRITICAL 最高优先级
D. THREAD_PRIORITY_ABOVE_NORMAL 高于普通优先级

10. 下面四个选项中，哪一个不是 WinMain 函数的参数？ （D ）
A. HINSTANCE B. INT
C. LPSTR D. WPARAM

11. VC++的编译器中，运算符 new 底层的实现是什么？ （B ）
A. VirtualAlloc() B. HeapAlloc()
C. GlobalAlloc() D. AllocateUserPhysicalPages()

12. 下面哪一本 C++参考书最厚？ （ C）
A. 《Think in C++》 B. 《深入浅出 MFC》
C. 《C++ Primer》 D. 《Effective C++》

13. 当调用 Windows API 函数 InvalidateRect,将会产生什么消息（A）
A. WM_PAINT B. WM_CREATE
C. WM_NCHITTEST D. WM_SETFOCUS

14. 关于 virtual void Draw()=0，下面说法正确的有几个（C）
(1)它是纯虚函数(对)
(2)它在定义它的类中不能实现(对)
(3)定义它的类不可实例化(对)
(4)如果一个类要继承一个 ADT 类,必须要实现其中的所有纯虚函数(错)//可以不实现,派生之后的类仍旧作为一个抽象类.
A. 1 B. 2
C. 3 D. 4

二、不定项选择题：（共 6 题，每题 3 分，共 18 分，多选、错选、漏选均不给分）
1. vector::iterator 重载了下面哪些运算符？ （ACD）
A. ++ B. >>
C. *（前置） D. ==

2. CreateFile( )的功能有哪几个？ （AB ）
A. 打开文件 B. 创建新文件
C. 文件改名 D. 删除文件

3. 下面哪些是句柄（HANDLE）？ （ABCD ）

A. HINSTANCE 实例句柄 B. HWND 窗口句柄
C. HDC 设备描述符号句柄 D. HFONT 字体句柄

4. 下面哪些不是 OpenGL 标准几何元素的绘制模式？ （A ）
A. GL_FOG B. GL_LINE_STRIP
C. GL_POINTS D. GL_TRIANGLE_FAN

5. 下面哪些运算符不能被重载？ （ABD ）
A. 做用域运算符“::” B. 对象成员运算符“.”
C. 指针成员运算符“->” D. 三目运算符“? :”

6. 下面哪些人曾参与了世界上第一个 C++编译器的开发？ （ ）
A. Bill Gates B. Stanley Lippman
C. Anderson Hejlsberg D. Bjarne Stroustrup

7. 以下说法正确的是？ （ ABC）
A. 头文件中的 ifndef/define/endif 是为了防止该头文件被重复引用。
B. 对于#include <filename.h> ，编译器从标准库路径开始搜索 filename.h
对于#include “filename.h” ，编译器从用户的工作路径开始搜索 filename.h
C. C++语言支持函数重载，C 语言不支持函数重载。函数被 C++编译后在库中的名字与 C 语言的不同。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。C++提供了 C 连接交换指定符号 extern“C”来解决名字匹配问题。
D. fopen 函数只是把文件目录信息调入内存。//错,fopen 是把整个文件读入内存

三、填空题：（共 8 题，每题 3 分，共 24 分）
1. 一个大小为 320 X 192，颜色为灰度索引色的设备相关位图有_____字节。如果此位图颜色为 24 位真彩色，则它的大小有_____字节。

2. Windows API 的中文意义是____ windows 应用程序接口___。

3. 计算反正弦的库函数是__asin()____；计算浮点数绝对值的库函数是__fabs()____；计算浮点数 n 次方的库函数是__pow()____;将浮点数转化为字符串的库函数是___fcvt()___。

4. 如果 i 等于 5，那么(++i) - -的返回值是__6__。

5. API LoadBitmap()的功能是从__指定的模块和或应用程序实例__中读取位图数据到内存。

6. new 和__delete___对应，malloc 和_free___对应，他们之间_ 不能__交叉混用。calloc 的功能是__为数组动态分配内存___，realloc 的功能是_改变原有内存区域的大小_。

7. SendMessage 和 PostMessage 都会向窗体发送一个消息，但 SendMessage__将一条消息发送到指定窗口,立即处理__而 PostMessage__将一条消息投递到指定窗口的消息队列,不需要立即处理___。

8．输出指定圆心、半径、边数的圆上的点：

```
const int nCount = 12;
const double dOrgX = 5.0,
dOrgY = 3.0;
const double dRadius = 2.0;
for( int i = 0; i < nCount; i++ )
{
    double dAngle = M_PI * 2.0 / (double)nCount * i;
    cout << "第" << i << "点：X = " << ________; cout << ", Y = " << __________ <<
    endl;
}
```

三、判断题：（共 12 题，每题 2 分，共 24 分）

1．一个类必须要有一个不带参数的构造函数。 错

2．你不能写一个虚的构造函数。 对

3．类里面所有的函数都是纯虚函数时才是纯虚类。 错

4．const 成员函数对于任何本类的数据成员都不能进行写操作。 对

5．函数中带默认值的参数必须位于不带默认值的参数之后。 对

6．char *p = "Test"; p[0] = 'R'; 错

7．cout << "Test"; 对

8．stl::list 不支持随机访问叠代器。 对

9．stl::vector 的效率比 stl::list 高。 错

10．VC 和 VC++是一回事，而 VC++是一种比 C++更难一些的语言。 错

11．理论上，new 和 malloc 造成的内存泄露都会由操作系统回收。 错

12．在 C++中 struct 和 class 的差别很大，所以从语法上不能混用。对

四、简述题(共 3 题，每题 5 分，共 15 分)

1．请简述 PeekMessage 和 GetMessage 的区别。

答：Peekmessage 和 Getmessage 都是向系统的消息队列中取得消息，两个函数的不同在于取不到消息的时候,若 Getmessage（）向消息队列中取不到消息，则程序的主线程会被 OS（操作系统）挂起,等到有合适的消息时才返回;若是用 Peekmessage（）在消息队列中取不到消息,则程序会取得 OS 控制权，运行一段时间。

另外,在处理消息的时候,GetMessag()会将消息从队列中删除,而 PeekMessage()可以设置最后一个参数 wRemoveMsg 来决定是否将消息保留在队列中。

2．请列出你所知道的在 Windows SDK 平台上，实现计时功能的方法。

答:可以使用 SetTimer 函数创建一个计时器,SetTimer 的函数原型如下：

```
UINT_PTR  SetTimer( HWND hWnd, UINT_PTR nIDEvent, UINT uElapse, TIMERPROC lpTimerFunc
```

3．请简述你所知道的 const 的各种用法。
答：const 常量
const 修饰类的数据成员
const 修饰指针
const 应用在函数声明中
const 应用在类成员函数

五、编程题：（共 3 题，第 1 小题 7 分，第 2 小题 14 分，第 3 小题 24 分）
1．深度遍历二叉树。

```
struct Node
{
Node *Parent;

Node *Left, *Right;

};
void Through(Node *Root)
{
}
```

2．二分法查找。

```
int DicFind( int *Array, int Count, int Value )
{

}
```

3．写出字符串类 String 的默认构造函数、析构函数和重载赋值运算符。
已知类 String 的原型为：

```
class String
{
    public:
    String( const char *pStr = NULL ); // 默认构造函数
    ~String( void ); // 析构函数
    String &operate = ( const String &Source ); // 重载赋值运算符
    private:
    char *m_pData; // 指向字符串的指针
};
```

试题 13：
1．用预处理指令#define 声明一个常数，用以表明 1 年中有多少秒(忽略闰年问题)

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

我在这想看到几件事情：

1) #define 语法的基本知识(例如：不能以分号结束，括号的使用，等等)

2)懂得预处理器将为你计算常数表达式的值，因此，直接写出你是怎么样计算一年中有多少秒而不是计算出实际的值，是更清晰而没有代价的。

3) 意识到这个表达式将使一个16位机的整型数溢出-因此要用到长整型符号L,告诉编译器这个常数是的长整型数。

4) 如果你在你的表达式中用到UL(表示无符号长整型)，那么你有了一个好的起点。记住，第一印象很重要。

2．写一个"标准"宏MIN，这个宏输入两个参数并返回较小的一个。

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

这个测试是为下面的目的而设的：

1) 标识#define在宏中应用的基本知识。这是很重要的。因为在 嵌入(inline)操作符变为标准C的一部分之前，宏是方便产生嵌入代码的唯一方法，对于嵌入式系统来说，为了能达到要求的性能，嵌入代码经常是必须的方法。

2)三重条件操作符的知识。这个操作符存在 C 语言中的原因是它使得编译器能产生比if-then-else更优化的代码，了解这个用法是很重要的。

3) 懂得在宏中小心地把参数用括号括起来

4) 我也用这个问题开始讨论宏的副作用，例如：当你写下面的代码时会发生什么事？

3．预处理器标识#error的目的是什么？

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

死循环(Infinite loops)

4．嵌入式系统中经常要用到无限循环，你怎么样用C编写死循环呢？

这个问题用几个解决方案。我首选的方案是：

```
while(1)
{
}
```

一些程序员更喜欢如下方案：

```
for(;;)
{
}
```

这个实现方式让我为难，因为这个语法没有确切表达到底怎么回事。如果一个应试者给出这个作为方案，我将用这个作为一个机会去探究他们这样做的基本原理。如果他们的基本答案是："我被教着这样做，但从没有想到过为什么。"这会给我留下一个坏印象。第三个方案是用 goto

```
Loop:
...
goto Loop;
```

应试者如给出上面的方案,这说明或者他是一个汇编语言程序员(这也许是好事)或者他是一个想进入新领域的BASIC/FORTRAN程序员。

数据声明(Data declarations)

5. 用变量 a 给出下面的定义

a) 一个整型数(An integer)。

b)一个指向整型数的指针(A pointer to an integer)。

c)一个指向指针的的指针，它指向的指针是指向一个整型数(A pointer to a pointer to an intege)r 。

d)一个有 10 个整型数的数组(An array of 10 integers)。

e) 一个有 10 个指针的数组，该指针是指向一个整型数的(An array of 10 pointers to integers)。

f) 一个指向有 10 个整型数数组的指针(A pointer to an array of 10 integers)。

g) 一个指向函数的指针，该函数有一个整型参数并返回一个整型数(A pointer to a function that takes an integer as an argument and returns an integer)。

h) 一个有 10 个指针的数组，该指针指向一个函数，该函数有一个整型参数并返回一个整型数(An array of ten pointers to functions that take an integer argument and return an integer)。

答案是：

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

6. 关键字 static 的作用是什么？

这个简单的问题很少有人能回答完全。在 C 语言中，关键字 static 有三个明显的作用：

1). 在函数体，一个被声明为静态的变量在这一函数被调用过程中维持其值不变。

2). 在模块内（但在函数体外），一个被声明为静态的变量可以被模块内所用函数访问，但不能被模块外其它函数访问。

它是一个本地的全局变量。

3). 在模块内，一个被声明为静态的函数只可被这一模块内的其它函数调用。那就是，

这个函数被限制在声明它的模块的本地范围内使用。

大多数应试者能正确回答第一部分，一部分能正确回答第二部分，同是很少的人能懂得第三部分。

这是一个应试者的严重的缺点，因为他显然不懂得本地化数据和代码范围的好处和重要性。

7. 关键字 const 是什么含意？

我只要一听到被面试者说：“const 意味着常数”，我就知道我正在和一个业余者打交道。

去年 Dan Saks 已经在他的文章里完全概括了 const 的所有用法，因此 ESP(译者：Embedded Systems Programming)的每一位读者应该非常熟悉 const 能做什么和不能做什么.

如果你从没有读到那篇文章，只要能说出 const 意味着“只读”就可以了。尽管这个答案不是完全的答案，但我接受它作为一个正确的答案。（如果你想知道更详细的答案，仔细读一下 Saks 的文章吧。）
如果应试者能正确回答这个问题，我将问他一个附加的问题：下面的声明都是什么意思？

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

前两个的作用是一样，a 是一个常整型数。第三个意味着 a 是一个指向常整型数的指针（也就是，整型数是不可修改的，但指针可以）。第四个意思 a 是一个指向整型数的常指针（
也就是说，指针指向的整型数是可以修改的，但指针是不可修改的）。
最后一个意味着 a 是一个指向常整型数的常指针（也就是说，指针指向的整型数是不可修改的，同时指针也是不可修改的）。
如果应试者能正确回答这些问题，那么他就给我留下了一个好印象。
顺带提一句，也许你可能会问，即使不用关键字 const，也还是能很容易写出功能正确的程序，
那么我为什么还要如此看重关键字 const 呢？我也如下的几下理由：
1). 关键字 const 的作用是为给读你代码的人传达非常有用的信息，实际上，
声明一个参数为常量是为了告诉了用户这个参数的应用目的。如果你曾花很多时间清理其它人留下的垃圾，
你就会很快学会感谢这点多余的信息。（当然，懂得用 const 的程序员很少会留下的垃圾让别人来清理的。）
2). 通过给优化器一些附加的信息，使用关键字 const 也许能产生更紧凑的代码。
3). 合理地使用关键字 const 可以使编译器很自然地保护那些不希望被改变的参数，防止其被无意的代码修改。
简而言之，这样可以减少 bug 的出现。

8. 关键字 volatile 有什么含意 并给出三个不同的例子。
一个定义为 volatile 的变量是说这变量可能会被意想不到地改变，这样，编译器就不会去假设这个变量的值了。
精确地说就是，优化器在用到这个变量时必须每次都小心地重新读取这个变量的值，而不是使用保存在寄存器里的备份。
下面是 volatile 变量的几个例子：
1). 并行设备的硬件寄存器（如：状态寄存器）
2). 一个中断服务子程序中会访问到的非自动变量(Non-automatic variables)
3). 多线程应用中被几个任务共享的变量
回答不出这个问题的人是不会被雇佣的。我认为这是区分 C 程序员和嵌入式系统程序员的最基本的问题。
嵌入式系统程序员经常同硬件、中断、RTOS 等等打交道，所用这些都要求 volatile 变量。不懂得 volatile 内容将会带来灾难。

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

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

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

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

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

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

位操作（Bit manipulation）
9. 嵌入式系统总是要用户对变量或寄存器进行位操作。给定一个整型变量 a，写两段代码，
第一个设置 a 的 bit 3，
第二个清除 a 的 bit 3。在以上两个操作中，要保持其它位不变。

对这个问题有三种基本的反应
1). 不知道如何下手。该被面者从没做过任何嵌入式系统的工作。
2). 用 bit fields。Bit fields 是被扔到 C 语言死角的东西，它保证你的代码在不同编译
器之间是不可移植的，
同时也保证了的你的代码是不可重用的。我最近不幸看到 Infineon 为其较复杂的通信芯片
写的驱动程序，它用到了 bit fields 因此完全对我无用，因为我的编译器用其它的方式来

实现 bit fields 的。从道德讲：永远不要让一个非嵌入式的家伙粘实际硬件的边。
3). 用 #defines 和 bit masks 操作。这是一个有极高可移植性的方法，是应该被用到的方法。最佳的解决方案如下：

```
#define BIT3 (0x1<<3)
static int a;
void set_bit3(void)
{
    a |= BIT3;
}
void clear_bit3(void)
{
    a &= ~BIT3;
}
```

一些人喜欢为设置和清除值而定义一个掩码同时定义一些说明常数，这也是可以接受的。我希望看到几个要点：说明常数、|=和&=~操作。

访问固定的内存位置（Accessing fixed memory locations） CC++ Development
10. 嵌入式系统经常具有要求程序员去访问某特定的内存位置的特点。在某工程中，要求设置一绝对地址为 0x67a9 的整型变量的值为 0xaa66。编译器是一个纯粹的 ANSI 编译器。写代码去完成这一任务。

这一问题测试你是否知道为了访问一绝对地址把一个整型数强制转换（typecast）为一指针是合法的。
这一问题的实现方式随着个人风格不同而不同。典型的类似代码如下：

```
int *ptr;
ptr = (int *)0x67a9;
*ptr = 0xaa55;
```

一个较晦涩的方法是：

```
*(int * const)(0x67a9) = 0xaa55;
```

即使你的品味更接近第二种方案，但我建议你在面试时使用第一种方案。

中断（Interrupts）
11. 中断是嵌入式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。
具代表事实是，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 double compute_area (double radius)
{
    double area = PI * radius * radius;
    printf(" Area = %f", area);
    return area;
```

```
}
```

这个函数有太多的错误了，以至让人不知从何说起了：
1). ISR 不能返回一个值。如果你不懂这个，那么你不会被雇用的。
2). ISR 不能传递参数。如果你没有看到这一点，你被雇用的机会等同第一项。
3). 在许多的处理器/编译器中，浮点一般都是不可重入的。有些处理器/编译器需要让额处的寄存器入栈，
有些处理器/编译器就是不允许在 ISR 中做浮点运算。此外，ISR 应该是短而有效率的，在 ISR 中做浮点运算是不明智的。
4). 与第三点一脉相承，printf()经常有重入和性能上的问题。如果你丢掉了第三和第四点，我不会太为难你的。
不用说，如果你能得到后两点，那么你的被雇用前景越来越光明了。

代码例子（Code examples）
12 . 下面的代码输出是什么，为什么？

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

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

```
<p align="left"></p>
```

13. 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;
```

这一问题真正能揭露出应试者是否懂得处理器字长的重要性。在我的经验里，好的嵌入式程序员非常准确地明白硬件的细节和它的局限，
然而 PC 机程序往往把硬件作为一个无法避免的烦恼。
到了这个阶段，应试者或者完全垂头丧气了或者信心满满志在必得。如果显然应试者不是很好，那么这个测试就在这里结束了。
但如果显然应试者做得不错，那么我就扔出下面的追加问题，这些问题是比较难的，我想仅仅非常优秀的应试者能做得不错。

提出这些问题，我希望更多看到应试者应付问题的方法，而不是答案。不管如何，你就当这个是娱乐吧…

试题 14：C++笔试题（1）

一道 C 语言笔试题

设一数据库人员档案表为如下结构：name(姓名)、address(地址)、
Teleno(电话号码)、image (相片数据)；存储长度：8Bytes、20Bytes、12Bytes、
变长的二进制数据；若将数据库中的档案记录信息存入文件，其中相片数据信息数据可能或有或无，
且即使有，数据信息亦不定长。问题：每一记录存储于文件时，其相片数据要求按它的实际长度来存放，
试设计一数据结构，并解释各成员变量含义。

一些 C 语言笔试题

一、请填写 BOOL , float, 指针变量 与“零值”比较的 if 语句。(10 分)

1. 请写出 BOOL flag 与“零值”比较的 if 语句。(3 分)

标准答案：

```
if ( flag )
if ( !flag )
```

如下写法均属不良风格，不得分。

```
if (flag == TRUE)
if (flag == 1 )
if (flag == FALSE)
    if (flag == 0)
```

2. 请写出 float x 与“零值”比较的 if 语句。(4 分)

标准答案示例：

```
const float EPSINON = 0.000001;
if ((x <= - EPSINON) && (x >= EPSINON)
```

不可将浮点变量用“==”或“！=”与数字比较，应该设法转化成“>=”或“<=”此类形式。

如下是错误的写法，不得分。

```
if (x == 0.0)
if (x != 0.0)
```

3. 请写出 char *p 与“零值”比较的 if 语句。(3 分)

标准答案：

```
if (p == NULL)
if (p != NULL)
```

如下写法均属不良风格，不得分。

```
if (p == 0)
if (p != 0)
if (p)
```

```
    if (!)
```

二、以下为 Windows NT 下的 32 位 C++程序，请计算 sizeof 的值（10 分）

```
    char  str[] = “Hello” ;
    char   *p = str ;
    int     n = 10;
```

1. 请计算

```
sizeof (str ) =  6    （2 分）  //加上一个/0 结束标志符共 6 位，且 CHAR 为一个字节
  sizeof ( p ) =   4    （2 分）  //指针变量的是四个字节
  sizeof ( n ) =   4    （2 分）  //int 类型是四个字节
void Func ( char str[100])
{
请计算
sizeof( str ) =   4      （2 分）// void 类型为四字节
}
void *p = malloc( 100 );
请计算
sizeof ( p ) =  4       （2 分）  //指针类型为四字节
```

三、简答题（25 分）

1、头文件中的 ifndef/define/endif 干什么用？（5 分）

答：防止该头文件被重复引用。

2、#include　　和 #include “filename.h” 有什么区别？（5 分）

答：对于#include　，编译器从标准库路径开始搜索 filename.h

　　对于#include “filename.h” ，编译器从用户的工作路径开始搜索 filename.h

3、const 有什么用途？（请至少说明两种）（5 分）

答：

（1）可以定义 const 常量

（2）const 可以修饰函数的参数、返回值，甚至函数的定义体。被 const 修饰的东西都受到强制保护，可以预防意外的变动，能提高程序的健壮性。

4、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”？ （5 分）

答：C++语言支持函数重载，C 语言不支持函数重载。函数被 C++编译后在库中的名字与 C 语言的不同。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。

C++提供了 C 连接交换指定符号 extern“C”来解决名字匹配问题。

5、请简述以下两个 for 循环的优缺点（5 分）

```
for (i=0; ii++)
{
if (condition)
```

```
    DoSomething();
else
    DoOtherthing();
}
```

优点：程序简洁
缺点：多执行了 N-1 次逻辑判断，并且打断了循环“流水线”作业，使得编译器不能对循环进行优化处理，降低了效率。

```
if (condition)
{
for (i=0; ii++)
    DoSomething();
}
else
{
    for (i=0; ii++)
    DoOtherthing();
}
```

优点：循环的效率高　缺点：程序不简洁

四、有关内存的思考题（每小题 5 分，共 20 分）

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

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

后的 str 仍然为 NULL；

```
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。 char p[] = "hello world";
return p;
的 p[]数组为函数内的局部自动变量，在函数返回后，内存已经被释放。这是许多程序员常犯的错误，其根源在于不理解变量的生存期。
```
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 ）Test 函数中也未对 malloc 的内存进行释放。
（3）GetMemory 避免了试题 1 的问题，传入 GetMemory 的参数为字符串指针的指针，但是在 GetMemory 中执行申请内存及赋值语句
```
*p = (char *) malloc( num );
```
后未判断内存是否申请成功，应加上：
```
if ( *p == NULL )
{
   ...//进行申请内存失败处理
  }

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：执行
```
char *str = (char *) malloc(100);
```
后未进行内存是否申请成功的判断；另外，在 free(str)后未置 str 为空，导致可能变成一个“野”指针，应加上：
```
str = NULL;
```

五、编写 strcpy 函数（10 分）
已知 strcpy 函数的原型是
```
       char *strcpy(char *strDest, const char *strSrc);
```
其中 strDest 是目的字符串，strSrc 是源字符串。
（1）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);
{
    assert((strDest!=NULL) && (strSrc !=NULL));  // 2 分
    char *address = strDest;                    // 2 分
    while( (*strDest++ = * strSrc++) != ‘0’ )    // 2 分
       NULL ;
    return address ;                           // 2 分
}
```

（2）strcpy 能把 strSrc 的内容复制到 strDest，为什么还要 char * 类型的返回值？
答：为了实现链式表达式。   // 2 分
例如   int length = strlen(strcpy(strDest, “hello world”));

六、编写类 String 的构造函数、析构函数和赋值函数（25 分）
已知类 String 的原型为：
```
    class String
    {
      public:
        String(const char *str = NULL); // 普通构造函数
        String(const String &other);        // 拷贝构造函数
        ~ String(void);                     // 析构函数
        String & operate =(const String &other);    // 赋值函数
      private:
        char    *m_data;                // 用于保存字符串
    };
```
请编写 String 的上述 4 个函数。

标准答案：

```
// String 的析构函数
       String::~String(void)              // 3 分
{
    delete [] m_data;
// 由于 m_data 是内部数据类型，也可以写成 delete m_data;
       }
       // String 的普通构造函数
       String::String(const char *str )      // 6 分
{
    if(str==NULL)
    {
       m_data = new char[1];    // 若能加 NULL 判断则更好
       *m_data = '0' ;
    }
    else
    {
       int length = strlen(str);
       m_data = new char[length+1];  // 若能加 NULL 判断则更好
       strcpy(m_data, str);
    }
}
// 拷贝构造函数
    String::String(const String &other)   // 3 分
    {
    int length = strlen(other.m_data);
    m_data = new char[length+1];      // 若能加 NULL 判断则更好
    strcpy(m_data, other.m_data);
}
// 赋值函数
    String & String::operate =(const String &other)    // 13 分
    {
       // (1) 检查自赋值                    // 4 分
       if(this == &other)
           return *this;

// (2) 释放原有的内存资源            // 3 分
       delete [] m_data;

       // （3）分配新的内存资源，并复制内容 // 3 分
    int length = strlen(other.m_data);
    m_data = new char[length+1];         // 若能加 NULL 判断则更好
       strcpy(m_data, other.m_data);
```

```
        // （4）返回本对象的引用                // 3 分
        return *this;
}
```

试题 15： C/C++试题

一、请填写 BOOL ，float，指针变量 与“零值”比较的 if 语句。（10 分）

请写出 BOOL flag 与“零值”比较的 if 语句。（3 分）	

标准答案：
if (flag)
if (!flag)
	如下写法均属不良风格，不得分。
	if (flag == TRUE)
	if (flag == 1)
	if (flag == FALSE)
	if (flag == 0)

请写出 float x 与“零值”比较的 if 语句。（4 分）	

标准答案示例：
const float EPSINON = 0.00001;
if ((x >= - EPSINON) && (x <= EPSINON)
不可将浮点变量用“==”或“！=”与数字比较，应该设法转化成“>=”或“<=”此类形式。
	如下是错误的写法，不得分。
	if (x == 0.0)
	if (x != 0.0)

请写出 char *p 与“零值”比较的 if 语句。（3 分）	

标准答案：
if (p == NULL)
if (p != NULL)
	如下写法均属不良风格，不得分。
	if (p == 0)
	if (p != 0)
	if (p)
	if (!)

二、以下为 Windows NT 下的 32 位 C++程序，请计算 sizeof 的值（10 分）

char str[] = “Hello” ;
char *p = str ;
int n = 10;
请计算
sizeof (str) = 6 （2 分）
 	
sizeof (p) = 4 （2 分）
 	
sizeof (n) = 4 （2 分）
	void Func (char str[100])
	{
	请计算
	sizeof(str) = 4 （2 分）
	}
	
	void *p = malloc(100);
	请计算
	sizeof (p) = 4 （2 分）

三、简答题（25 分）

1、头文件中的 ifndef/define/endif 干什么用？（5 分）

答：防止该头文件被重复引用。

2、#include <filename.h> 和 #include “filename.h” 有什么区别？（5 分）

答：对于#include <filename.h> ，编译器从标准库路径开始搜索 filename.h

对于#include “filename.h”，编译器从用户的工作路径开始搜索 filename.h

3、const 有什么用途？（请至少说明两种）（5 分）
答：（1）可以定义 const 常量
（2）const 可以修饰函数的参数、返回值，甚至函数的定义体。被 const 修饰的东西都受到强制保护，可以预防意外的变动，能提高程序的健壮性。

4、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”？（5 分）
答：C++语言支持函数重载，C 语言不支持函数重载。函数被 C++编译后在库中的名字与 C 语言的不同。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。
C++提供了 C 连接交换指定符号 extern“C”来解决名字匹配问题。

5、请简述以下两个 for 循环的优缺点（5 分）

for (i=0; i<N; i++)
{
if (condition)
DoSomething();
else
DoOtherthing();
}
	if (condition)
	{
	for (i=0; i<N; i++)
	DoSomething();
	}
	else
	{
	for (i=0; i<N; i++)
	DoOtherthing();
	}

优点：程序简洁 缺点：多执行了 N-1 次逻辑判断，并且打断了循环“流水线”作业，使得编译器不能对循环进行优化处理，降低了效率。	优点：循环的效率高 缺点：程序不简洁

四、有关内存的思考题（每小题 5 分，共 20 分）

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。
	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 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。	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。

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
	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。

五、编写 strcpy 函数（10 分）

已知 strcpy 函数的原型是

char *strcpy(char *strDest, const char *strSrc);

其中 strDest 是目的字符串，strSrc 是源字符串。

（1）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);
{
    assert((strDest!=NULL) && (strSrc !=NULL)); // 2 分
    char *address = strDest;                    // 2 分
    while( (*strDest++ = * strSrc++) != ‘\0’ )    // 2 分
       NULL ;
    return address ;                            // 2 分
}
```

（2）strcpy 能把 strSrc 的内容复制到 strDest，为什么还要 char * 类型的返回值？

答：为了实现链式表达式。 // 2 分

例如 int length = strlen(strcpy(strDest, “hello world”));

六、编写类 String 的构造函数、析构函数和赋值函数（25 分）

已知类 String 的原型为：

```
class String
{
  public:
```

```
        String(const char *str = NULL); // 普通构造函数
        String(const String &other);        // 拷贝构造函数
        ~ String(void);                     // 析构函数
        String & operate =(const String &other);    // 赋值函数
      private:
        char    *m_data;                // 用于保存字符串
    };
```

请编写 String 的上述 4 个函数。

标准答案：

```
// String 的析构函数
       String::~String(void)              // 3 分
{
    delete [] m_data;
// 由于 m_data 是内部数据类型，也可以写成 delete m_data;
       }
       // String 的普通构造函数
       String::String(const char *str)      // 6 分
{
    if(str==NULL)
    {
       m_data = new char[1];    // 若能加 NULL 判断则更好
       *m_data = '\0';
    }
    else
    {
       int length = strlen(str);
       m_data = new char[length+1];  // 若能加 NULL 判断则更好
       strcpy(m_data, str);
    }
}
// 拷贝构造函数
    String::String(const String &other)   // 3 分
    {
    int length = strlen(other.m_data);
    m_data = new char[length+1];      // 若能加 NULL 判断则更好
    strcpy(m_data, other.m_data);
}
// 赋值函数
    String & String::operate =(const String &other)    // 13 分
    {
       // (1) 检查自赋值                    // 4 分
       if(this == &other)
           return *this;
```

```
// (2) 释放原有的内存资源                // 3 分
        delete [] m_data;

        // （3）分配新的内存资源，并复制内容 // 3 分
    int length = strlen(other.m_data);
    m_data = new char[length+1];          // 若能加 NULL 判断则更好
        strcpy(m_data, other.m_data);

        // （4）返回本对象的引用              // 3 分
        return *this;
}
```

试题 16：C 面试大全

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。这两者的区别在于非静态全局变量的作用域是整个源程序， 当一个源程序由多个源文件组成时，非静态的全局变量在各个源文件中都是有效的。 而静态全局变量则限制了其作用域，即只在定义该变量的源文件内有效， 在同一源程序的其它源文件中不能使用它。由于静态全局变量的作用域局限于一个源文件内，只能为该源文件内的函数公用， 因此可以避免在其它源文件中引起错误。从以上分析可以看出， 把局部变量改变为静态变量后是改变了它的存储方式即改变了它的生存期。把全局变量改变为静态变量后是改变了它的作用域， 限制了它的使用范围。 static 函数与普通函数作用域不同。仅在本文件。只在当前源文件中使用的函数应该说明为内部函数(static)，内部函数应该在当前源文件中说明和定义。对于可在当前源文件以外使用的函数，应该在一个头文件中说明，要使用这些函数的源文件要包含这个头文件 static 全局变量与普通的全局变量有什么区别：static 全局变量只初使化一次，防止在其他文件单元中被引用；static 局部变量和普通局部变量有什么区别：static 局部变量只被初始化一次，下一次依据上一次结果值；static 函数与普通函数有什么区别：static 函数在内存中只有一份，普通函数在每个被调用中维持一份拷贝

五.问答题:

1.IP Phone 的原理是什么？IPV6(128 位)

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？

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

3. 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******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：
int do_dup(int a[],int N)

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;

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 = _t << p; 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。

2. 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 && str<='z') str -= ('a'-'A'); } char str[] = "aBcDe"; cout

<< "str 字符长度为: " << sizeof(str)/sizeof(str[0]) << endl; UpperCase(str); cout << str << endl;

答：函数内的 sizeof 有问题。根据语法，sizeof 如用于数组，只能测出静态数组的大小，无法检测动态分配的或外部数组大小。函数外的 str 是一个静态定义的数组，因此其大小为 6，函数内的 str 实际只是一个指向字符串的指针，没有任何额外的与数组相关的信息，因此 sizeof 作用于上只将其当指针看，一个指针为 4 个字节，因此返回 4。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].

3、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。

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;

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(;;)

4、关键字 static 的作用是什么？

定义静态变量

5、关键字 const 有什么含意？表示常量不可以修改的变量。

6、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。 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。

7.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

8.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;

9.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

10.#i nclude <stdio.h>
#i nclude <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

11.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

12.列举几种进程的同步机制，并比较其优缺点。 原子操作信号量机制 自旋锁 管程，会合，分布式系统

13.进程之间通信的途径 共享存储系统 消息传递 系统管道：以文件系统为基础

14.进程死锁的原因资源竞争及进程推进顺序非法

15.死锁的 4 个必要条件互斥、请求保持、不可剥夺、环路

16.死锁的处理鸵鸟策略、预防策略、避免策略、检测与解除死锁

17. 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

18.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

19.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

20.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

22.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

23：(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

24：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?
mainc 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)();

25、已知一个数组 table，用一个宏定义，求出数据的元素个数
#define NTBL (sizeof(table)/sizeof(table[0]))

26、 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。

27、分析下面的程序
```
unsigned short A = 10;
printf("~A = %u\n", ~A);
uchar 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。

28、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？
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。

试题 17：

1、请说出 static 和 const 关键字尽可能多的作用
解答：
static 关键字至少有下列 n 个作用：
（1）函数体内 static 变量的作用范围为该函数体，不同于 auto 变量，该变量的内存只被分配一次，因此其值在下次调用时仍维持上次的值；
（2）在模块内的 static 全局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；
（3）在模块内的 static 函数只可被这一模块内的其它函数调用，这个函数的使用范围被限制在声明它的模块内；
（4）在类中的 static 成员变量属于整个类所拥有，对类的所有对象只有一份拷贝；
（5）在类中的 static 成员函数属于整个类所拥有，这个函数不接收 this 指针，因而只能访问类的 static 成员变量。

const 关键字至少有下列 n 个作用：

（1）欲阻止一个变量被改变，可以使用 const 关键字。在定义该 const 变量时，通常需要对它进行初始化，因为以后就没有机会再去改变它了；

（2）对指针来说，可以指定指针本身为 const，也可以指定指针所指的数据为 const，或二者同时指定为 const；

（3）在一个函数声明中，const 可以修饰形参，表明它是一个输入参数，在函数内部不能改变其值；

（4）对于类的成员函数，若指定其为 const 类型，则表明其是一个常函数，不能修改类的成员变量；

（5）对于类的成员函数，有时候必须指定其返回值为 const 类型，以使得其返回值不为“左值”。例如：

```
const classA operator*(const classA& a1,const classA& a2);
```

operator*的返回结果必须是一个 const 对象。如果不是，这样的变态代码也不会编译出错：

```
classA a, b, c;
(a * b) = c; // 对 a*b 的结果赋值
```

操作(a * b) = c 显然不符合编程者的初衷，也没有任何意义。

2.技巧题

试题 1：请写一个 C 函数，若处理器是 Big_endian 的，则返回 0；若是 Little_endian 的，则返回 1

解答：

```
int checkCPU()
{
  {
    union w
    {
      int a;
      char b;
    } c;
    c.a = 1;
    return (c.b == 1);
  }
}
```

剖析：

嵌入式系统开发者应该对 Little-endian 和 Big-endian 模式非常了解。采用 Little-endian 模式的 CPU 对操作数的存放方 式是从低字节到高字节，而 Big-endian 模式对操作数的存放方式是从高字节到低字节。例如，16bit 宽的数 0x1234 在 Little- endian 模式 CPU 内存中的存放方式（假设从地址 0x4000 开始存放）为：

内存地址 存放内容

0x4000 0x34

0x4001 0x12

而在 Big-endian 模式 CPU 内存中的存放方式则为：

内存地址 存放内容
0x4000 0x12
0x4001 0x34

32bit 宽的数 0x12345678 在 Little-endian 模式 CPU 内存中的存放方式（假设从地址 0x4000 开始存放）为：

内存地址 存放内容
0x4000 0x78
0x4001 0x56
0x4002 0x34
0x4003 0x12

而在 Big-endian 模式 CPU 内存中的存放方式则为：

内存地址 存放内容
0x4000 0x12
0x4001 0x34
0x4002 0x56
0x4003 0x78

联合体 union 的存放顺序是所有成员都从低地址开始存放，面试者的解答利用该特性，轻松地获得了 CPU 对内存采用 Little-endian 还是 Big-endian 模式读写。如果谁能当场给出这个解答，那简直就是一个天才的程序员。

试题 2：写一个函数返回 1+2+3+…+n 的值（假定结果不会超过长整型变量的范围）

解答：

```
int Sum( int n )
{
  return ( (long)1 + n) * n / 2;    //或 return (1l + n) * n / 2;
}
```

剖析：

对于这个题，只能说，也许最简单的答案就是最好的答案。下面的解答，或者基于下面的解答思路去优化，不管怎么“折腾”，其效率也不可能与直接 return (1 l + n) * n / 2 相比！

```
int Sum( int n )
{
  long sum = 0;
  for( int i=1; i<=n; i++ )
  {
    sum += i;
  }
  return sum;
}
```

试题 18：嵌入式研发工程师面试试题大全(ANSI CC++方面的知识)

一．ANSI C/C++方面的知识

1、简答题。

1、 如何在 C 中初始化一个字符数组。

逐个字符赋值：char s[] = { ‘A’ ,’ B’ ,’ C’ ,’ D’ };
字符串赋值：char s[] = {“ABCD”};
对于二维字符数组：char s[2][10] = {“cheng”,” jinzhou”};

2、 如何在 C 中为一个数组分配空间。
如果是栈的形式，Type s[N]定义后系统自动分配空间，分配的空间大小受操作系统限制；若是堆的形式，Type *s; s = (Type *)malloc(sizeof(Type) * N); 分配的空间大小不受操作系统限制。

3、 如何初始化一个指针数组。
这里有必要重新对比一下指针数组与数组指针的差异。
a. 指针数组：数组里存储的是指针。
如：int * s[5] 表示数组 s 里存储了 5 个指向整型的指针。
Char * s[3] = {“aaaaa”,” bbb”,” ccccc”} 表示数组 s 里存储 3 个指向字符型的指针，分别指向字符串 aaaaa、bbb、ccccc。
b. 数组指针：其实就是数组，里面存放的是数据。
如：int (* s)[5] 表示数组 s 里存储了 5 个整型数据。

4、 如何定义一个有 10 个元素的整数型指针数组。
Int * s [10];

5、 s[10]的另外一种表达方式是什么。
* (s + 10)
二维数组 S [5][8]的表示方法：*(*(s + 5) + 8)

7、 要使用 CHAR_BIT 需要包含哪个头文件。
Include limits.h
在该头文件里 #define CHAR_BIT 8

8、 对(-1.2345)取整是多少? —1

9、 如何让局部变量具有全局生命期。
使用 Static，局部变量就存储在全局区（静态区），便具有全局的生命期和局部的访问控制。

10、C 中的常量字符串应在何时定义?
没有理解到题目的意思，我只是想说明一点，定义常量字符串后它属于 const 型，不能去修改它，否则程序出错。

11、如何在两个.c 文件中引用对方的变量。
尚不清楚，望博友能告知，万分感谢!

12、使用 malloc 之前需要做什么准备工作。
定义一个指针后就可以 malloc 了。

13、realloc 函数在使用上要注意什么问题。
Realloc 后返回的指针与之前 malloc 返回的指针指向的地址不同。

14、strtok 函数在使用上要注意什么问题。
首次调用时，s 必须指向要分解的字符串，随后调用要把 s 设成 NULL

15、gets 函数在使用上要注意什么问题。
这里要将 Scanf( )、gets( )放在一起比较。Scanf( )是遇到空格就判断为输入结束，而 gets( )则遇到回车才判断为输入结束。

16、C 语言的词法分析在长度规则方面采用的是什么策略？
尚不清楚，望博友能告知，万分感谢！

17、a+++++b 所表示的是什么意思？有什么问题？
根据自增运算符的右结合性，它是(a++)+(++b)的意思，但有的编译器里省略括号就不能通过，同时也降低了程序可读性。

18、如何定义 Bool 变量的 TRUE 和 FALSE 的值。
```
#define TRUE 1
#define FALSE 0
```

19、C 语言的 const 的含义是什么。在定义常量时，为什么推荐使用 const，而不是#define。
Const 是只读的意思，它限定一个变量不允许被改变。
#define 缺乏类型检测机制，在预处理时候有可能引发错误。

Const 方面的其它知识扩展：
问题 1：const 变量 & const 限定的内容
下面的代码编译器会报一个错误，请问，哪一个语句是错误的呢？
```
    typedef char * pStr;
    char string[4] = "abc";
    const char *p1 = string;  // *p1 作为整体不能被修改，但 p1 可以修改，p1++合法
    const pStr p2 = string;  //p2 作为一个整体，不能被修改，但是下面的 p2++非法修改
    p1++;
    p2++;
```

问题 2：const 变量 & 字符串常量
请问下面的代码有什么问题？
```
char *p = "i''''m hungry!";  //定义的是字符串常量
    p[0]= ''''I''  //不能修改字符串常量
```

问题：const 变量 & 字符串常量 2
    char a[3] = "abc" 合法吗？使用它有什么隐患？
没有考虑到字符串结束符‘\0’，所以会产生意想不到的错误。

比如以下程序：

```
int main()
{
       int i;
    char p[6] = {''''a'''',''''b'''',''''c'''',''''d'''',''''e'''',''''f''''};
    printf("%s",p);
    while(1);
    return 0;
}
```

运行后显示： abcdef@

问题 3：const & 指针

类型声明中 const 用来修饰一个常量，有如下两种写法，那么，请问，下面分别用 const 限定不可变的内容是什么?

1)、const 在前面

a. const int nValue； //nValue 是 const

把类型 int 撇开，变量 nValue 作为一个整体，因此 nValue 是 const 型；

b. const char *pContent; //*pContent 是 const，pContent 可变

把类型 char 撇开，变量 *pContent 作为一个整体，因此 *pContent 是 const 型；

c. const (char *) pContent;//pContent 是 const,*pContent 可变

把类型 char * 撇开，注意这里（char * ）是一个整体，而变量 pContent 作为一个整体，因此 pContent 是 const 型；

d. char* const pContent; //pContent 是 const,*pContent 可变

const 与变量间没有类型，变量 pContent 作为一个整体，因此 pContent 是 const 型；

e. const char* const pContent; //pContent 和*pContent 都是 const

这里分为两层，外层：把类型 char 撇开，变量 * const pContent 作为一个整体，因此 *pContent 是 const 型；内层：没有类型，因此 pContent 是 const 型。

2)、const 在后面，与上面的声明对等 (这类型更容易判断)

a. int const nValue； // nValue 是 const

const 与变量之间没有类型，const 后面那部分整体是 const 型，因此 nValue 是 const 型

b. char const * pContent;// *pContent 是 const，pContent 可变

const 与变量之间没有类型，const 后面那部分整体是 const 型，因此 * pContent 是 const 型

c. (char *) const pContent;//pContent 是 const,*pContent 可变

const 与变量之间没有类型,const 后面那部分整体是 const 型,因此 pContent 是 const 型

d. char* const pContent;// pContent 是 const,*pContent 可变

const 与变量之间没有类型,const 后面那部分整体是 const 型,因此 pContent 是 const 型

e. char const* const pContent;// pContent 和*pContent 都是 const

分为两层，外层：撇开类型 char，const 后面那部分整体* const pContent 是 const 型，因此 * pContent 是 const 型;内层:const 与 pContent 之间无类型,因此 pContent 是 const 型。

C++中 CONST

C 中常用:#define 变量名 变量值定义一个值替代,然而却有个致命缺点:缺乏类型检测机制,这样预处理理在 C++中成为可能引发错误的隐患,于是引入 const.

const 使用:

1. 用于指针的两种情况:const 是一个左结合的类型修饰符.

```
int const *A; //A 可变,*A 不可变
int *const A; //A 不可变,*A 可变
```

2.限定函数的传递值参数:

```
void function(const int Var); //传递过来的参数在函数内不可以改变.
```

3.限定函数返回值型.

```
const int function(); //此时 const 无意义
const myclassname function(); //函数返回自定义类型 myclassname.
```

20、C 语言的 volatile 的含义是什么。使用时会对编译器有什么暗示。

volatile 的本意是“易变的”

由于访问寄存器的速度要快过 RAM,所以编译器一般都会作减少存取外部 RAM 的优化，但有可能会读脏数据。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。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

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。

Volatile 的完全扩展：

1). 一个参数既可以是 const 还可以是 volatile 吗？解释为什么。

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). 一个指针可以是 volatile 吗？解释为什么。

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). 下面的函数有什么错误：

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

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;
}
```

试题 19：华为面试题

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

可以用引用头文件的方式，也可以用 extern 关键字，如果用引用头文件方式来引用某个在头文件中声明的全局变理，假定你将那个变写错了，那么在编译期间会报错，如果你用 extern 方式引用时，假定你犯了同样的错误，那么在编译期间不会报错，而在连接期间报错。

3、全局变量可不可以定义在可被多个.C 文件包含的头文件中？为什么？

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
main()
{
  int a,b,c,d;
```

```
    a=10;
    b=a++;
    c=++a;
    d=10*a++;
    printf("b，c，d：%d，%d，%d"，b，c，d）;
    return 0;
}
```

答：10，12，120

7、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。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

8、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。

9、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

10、队列和栈有什么区别？

队列先进先出，栈后进先出

11、写出下列代码的输出内容

```
#include
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

12、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
#include
main()
{
    char str[]="hello,world";
    int len=strlen(str);
    char t;
    for(int i=0; i
    {
        t=str[i];
        str[i]=str[len-i-1]; str[len-i-1]=t;
    }
    printf("%s",str);
    return 0;
}
```

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

4.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

5.直接链接两个信令点的一组链路称作什么?
PPP 点到点连接

6.接入网用的是什么接口?
7.voip 都用了那些协议?
8.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

9.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

10.

```
enum string
 {
    x1,
    x2,
    x3=10,
    x4,
    x5,
  }x;
```

问 x= 0x801005，0x8010f4 ；

11.

```
unsigned char *p1;
unsigned long *p2;
p1=(unsigned char *)0x801000;
p2=(unsigned long *)0x810000;
```

请问 p1+5= ；
p2+5= ；

12.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

试题 20：c 语言笔试面试宝典

1.new、delete、malloc、free 关系

delete 会调用对象的析构函数,和 new 对应 free 只会释放内存,new 调用构造函数。malloc 与 free 是 C++/C 语言的标准库函数，new/delete 是 C++的运算符。它们都可用于申请动态内存和释放内存。对于非内部数据类型的对象而言，光用 maloc/free 无法满足动态对象的要求。对象在创建的同时要自动执行构造函数，对象在消亡之前要自动执行析构函数。由于 malloc/free 是库函数而不是运算符，不在编译器控制权限之内，不能够把执行构造函数和析构函数的任务强加于 malloc/free。因此 C++语言需要一个能完成动态内存分配和初始化工作

的运算符 new，以及一个能完成清理与释放内存工作的运算符 delete。注意 new/delete 不是库函数。

2.delete 与 delete []区别

delete 只会调用一次析构函数，而 delete[]会调用每一个成员的析构函数。在 More Effective C++中有更为详细的解释："当 delete 操作符用于数组时，它为每个数组元素调用析构函数，然后调用 operatordelete 来释放内存。"delete 与 New 配套，delete []与 new []配套

```
MemTest*mTest1=newMemTest[10];
MemTest*mTest2=newMemTest;
int*pInt1=newint[10];
int*pInt2=newint;
delete[]pInt1;  //-1-
delete[]pInt2;  //-2-
delete[]mTest1;//-3-
delete[]mTest2;//-4-
```

在-4-处报错。

这就说明：对于内建简单数据类型，delete 和 delete[]功能是相同的。对于自定义的复杂数据类型，delete 和 delete[]不能互用。delete[]删除一个数组，delete 删除一个指针简单来说，用 new 分配的内存用 delete 删除用 new[]分配的内存用 delete[]删除 delete[]会调用数组元素的析构函数。内部数据类型没有析构函数，所以问题不大。如果你在用 delete 时没用括号，delete 就会认为指向的是单个对象，否则，它就会认为指向的是一个数组。

3.继承优缺点。

类继承是在编译时刻静态定义的，且可直接使用，类继承可以较方便地改变父类的实现。但是类继承也有一些不足之处。首先，因为继承在编译时刻就定义了，所以无法在运行时刻改变从父类继承的实现。更糟的是，父类通常至少定义了子类的部分行为，父类的任何改变都可能影响子类的行为。如果继承下来的实现不适合解决新的问题，则父类必须重写或被其他更适合的类替换。这种依赖关系限制了灵活性并最终限制了复用性。

（待补充）

4.C++有哪些性质（面向对象特点）

封装，继承和多态。

在面向对象程序设计语言中，封装是利用可重用成分构造软件系统的特性，它不仅支持系统的可重用性，而且还有利于提高系统的可扩充性；消息传递可以实现发送一个通用的消息而调用不同的方法；封装是实现信息隐蔽的一种技术，其目的是使类的定义和实现分离。

5.子类析构时要调用父类的析构函数吗？

析构函数调用的次序是先派生类的析构后基类的析构，也就是说在基类的的析构调用的时候,派生类的信息已经全部销毁了定义一个对象时先调用基类的构造函数、然后调用派生类的构造函数；析构的时候恰好相反：先调用派生类的析构函数、然后调用基类的析构函数
JAVA 无析构函数深拷贝和浅拷贝

6.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。

7.什么是“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。

8.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。

9.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 。

10.“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;

11.“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。

12.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。

13.面关于“联合”的题目的输出？

a)

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

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

答案：266 (低位低地址，高位高地址，内存占用情况是 Ox010A）

b)

```
main()
 {
    union{                /*定义一个联合*/
        int i;
        struct{           /*在联合中定义一个结构*/
            char first;
            char second;
        }half;
    }number;
```

```
    number.i=0x4241;        /*联合成员赋值*/
    printf("%c%cn", number.half.first, mumber.half.second);
    number.half.first='a';  /*联合中结构成员赋值*/
    number.half.second='b';
    printf("%xn", number.i);
    getch();
}
```

答案： AB (0x41 对应'A',是低位；Ox42 对应'B',是高位）

6261 (number.i 和 number.half 共用一块地址空间）

14.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。

15.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。

16.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.

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

答案：i 为 30。

18. 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。

19. 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。

20.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。

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

答案：如果不是零，请解释一下编译器为什么没有让它为零。（Autodesk）肯定不是零。举个反例，如果是零的话，声明一个 class A[10]对象数组，而每一个对象占用的空间是零，这时就没办法区分 A[0],A[1]…了。

22. 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。

23.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)

24.请说出 const 与#define 相比，有何优点？

答案：

Const 作用：定义常量、修饰函数参数、修饰函数返回值三个作用。被 Const 修饰的东西都受到强制保护，可以预防意外的变动，能提高程序的健壮性。

1） const 常量有数据类型，而宏常量没有数据类型。编译器可以对前者进行类型安全检查。而对后者只进行字符替换，没有类型安全检查，并且在字符替换可能会产生意料不到的错误。

2） 有些集成化的调试工具可以对 const 常量进行调试，但是不能对宏常量进行调试。

25.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
}
```
26.求出两个数中的较大这

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

答案：((a + b) + abs(a - b)) / 2

27.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;
}
```
28.指针找错题

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

2.找错题 试题 1：

以下是引用片段：
```
void test1()   //数组越界
    {
    char string[10];
    char* str1 = "0123456789";
    strcpy( string, str1 );
    }
```
试题 2：

以下是引用片段：
```
  void test2()
    {
    char string[10], str1[10];
```

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

试题 3：

以下是引用片段：

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

解答：

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

(1)字符串以'\0'结尾;

(2)对数组越界把握的敏感度;

(3)库函数 strcpy 的工作方式，

29.如果编写一个标准 strcpy 函数

总分值为 10，下面给出几个不同得分的答案：2 分 以下是引用片段：

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

4 分 以下是引用片段：

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

7 分 以下是引用片段：

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

```
while( (*strDest++ = * strSrc++) != '\0' );
}
```

10 分 以下是引用片段：

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

从 2 分到 10 分的几个答案我们可以清楚的看到，小小的 strcpy 竟然暗藏着这么多玄机，真不是盖的!需要多么扎实的基本功才能写一个完美的 strcpy 啊!

(4)对 strlen 的掌握，它没有包括字符串末尾的'\0'。

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

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

试题 4：以下是引用片段：

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

试题 5：

以下是引用片段：

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

```
void Test( void )
{
char *str = NULL;
str = GetMemory();
printf( str );
}
```

试题 6：以下是引用片段：

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

试题 7：以下是引用片段：

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

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

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

后的 str 仍然为 NULL;试题 5 中

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

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

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

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

后未判断内存是否申请成功，应加上：

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

}
```

试题 7 存在与试题 6 同样的问题，在执行

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

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

str = NULL;

试题 6 的 Test 函数中也未对 malloc 的内存进行释放。

剖析：

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

软件开发网 www.mscto.com

对内存操作的考查主要集中在：

(1)指针的理解;

(2)变量的生存期及作用范围;

(3)良好的动态内存申请和释放习惯。

再看看下面的一段程序有什么错误：

以下是引用片段：

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

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

以下是引用片段：

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

30.h 头文件中的 ifndef/define/endif 的作用？

答：防止该头文件被重复引用。

31.＃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。

32.请问交换机和路由器各自的实现原理是什么？分别在哪个层次上面实现的？

交换机：数据链路层。路由器：网络层。

33.8086 是多少位的系统？在数据总线上是怎么实现的？

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

代码段寄存器 CS：存放当前执行的程序的段地址。

数据段寄存器 DS：存放当前执行的程序所用操作数的段地址。

堆栈段寄存器 SS：存放当前执行的程序所用堆栈的段地址。

附加段寄存器 ES：存放当前执行程序中一个辅助数据段的段地址。

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

8086／8088 微处理器的存储器管理

1.地址线（码）与寻址范围：N 条地址线　　寻址范围=2N

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

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

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

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

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

7.逻辑地址由某段的段地址和段内偏移地址(也叫偏移量)两部分所组成。写成：

段地址：偏移地址(例如，1234H：0088H)。

8.在硬件上起作用的是物理地址，物理地址＝段基地址×10H 十偏移地址

34. 大唐面试

考试时间一小时，第一部分是填空和选择：

1．数列 6，10，18，32，“？”，问“？”是几？

2．某人出 70 买进一个 x，80 卖出，90 买回，100 卖出，这桩买卖怎么样？

3．月球绕地球一圈，至少要多少时间？

4．7 个人用 7 小时挖了 7 米的沟，以同样的速度在 50 小时挖 50 米的沟要多少人？

5．鱼头长 9，鱼尾等于鱼头加半个鱼身，鱼身等于鱼头加鱼尾，问鱼全长多少？

6．一个小姐买了一块手表，回家发现手表比她家的表慢了两分钟，晚上看新闻的时候又发现她家的表比新闻里的时间慢了两分钟，则 。

A 手表和新闻里的时间一样

B 手表比新闻里的时间慢

C 手表比新闻里的时间快

7．王先生看到一则招聘启事，发现两个公司除了以下条件不同外，其他条件都相同

A 半年年薪 50 万，每半年涨 5 万

B 一年年薪 100 万，每一年涨 20 万

王先生想去一家待遇比较优厚的公司，他会去哪家？

10．问哪个袋子里有金子？

A 袋子上的标签是这样写的：B 袋子上的话是对的，金子在 A 袋子。

B 袋子上的标签是这样写的：A 袋子上的话是错的，金子在 A 袋子里。

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

3×（1̃01）+2=29，问这是怎么回事？

12．三篇写作，均为书信形式。

（1）一片中文的祝贺信，祝贺某男当了某公司 xx

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

35.static 有什么用途？（请至少说明两种）

答 、1.限制变量的作用域(文件级的)。

2.设置变量的存储域(全局数据区)。

36.引用与指针有什么区别？

答 、1) 引用必须被初始化，指针不必。

2) 引用初始化以后不能被改变，指针可以改变所指的对象。

3) 不存在指向空值的引用，但是存在指向空值的指针。

37.描述实时系统的基本特性

答 、在特定时间内完成特定的任务，实时性与可靠性。

38.全局变量和局部变量在内存中是否有区别？如果有，是什么区别？

答 、全局变量储存在静态数据区，局部变量在堆栈中。

39.Internet 采用哪种网络协议？该协议的主要层次结构？

答 、tcp/ip 应用层/传输层/网络层/数据链路层/物理层

40.Internet 物理地址和 IP 地址转换采用什么协议？

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

41.IP 地址的编码分为哪俩部分？

答 、IP 地址由两部分组成，网络号和主机号。不过是要和“子网掩码”按位与之后才能区分哪些是网络位哪些是主机位。

42.不能做 switch()的参数类型是：

答 、switch 的参数不能为实型。华为

43.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

44.如何引用一个已经定义过的全局变量？

答 、可以用引用头文件的方式，也可以用 extern 关键字，如果用引用头文件方式来引用某个在头文件中声明的全局变理，假定你将那个变写错了，那么在编译期间会报错，如果你用 extern 方式引用时，假定你犯了同样的错误，那么在编译期间不会报错，而在连接期间报错

45.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

46.语句 for(； 1 ；)有什么问题？它是什么意思？

答 、和 while(1)相同。

47.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

48.请写出下列代码的输出内容

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

答 、10，12，120

49.statac 全局变量、局部变量、函数与普通全局变量、局部变量、函数

static 全局变量与普通的全局变量有什么区别？static 局部变量和普通局部变量有什么区别？static 函数与普通函数有什么区别？

答 、全局变量(外部变量)的说明之前再冠以 static 就构成了静态的全局变量。全局变量本身就是静态存储方式，静态全局变量当然也是静态存储方式。这两者在存储方式上并无不同。这两者的区别虽在于非静态全局变量的作用域是整个源程序， 当一个源程序由多个源文件组成时，非静态的全局变量在各个源文件中都是有效的。 而静态全局变量则限制了其作用域， 即只在定义该变量的源文件内有效， 在同一源程序的其它源文件中不能使用它。由于静态全局变量的作用域局限于一个源文件内，只能为该源文件内的函数公用， 因此可以避免在其它源文件中引起错误。

从以上分析可以看出， 把局部变量改变为静态变量后是改变了它的存储方式即改变了它的生存期。把全局变量改变为静态变量后是改变了它的作用域， 限制了它的使用范围。

static 函数与普通函数作用域不同。仅在本文件。只在当前源文件中使用的函数应该说明为内部函数(static)，内部函数应该在当前源文件中说明和定义。对于可在当前源文件以外使用的函数，应该在一个头文件中说明，要使用这些函数的源文件要包含这个头文件

static 全局变量与普通的全局变量有什么区别：static 全局变量只初使化一次，防止在其他文件单元中被引用;

static 局部变量和普通局部变量有什么区别：static 局部变量只被初始化一次，下一次依据上一次结果值;

static 函数与普通函数有什么区别：static 函数在内存中只有一份，普通函数在每个被调用中维持一份拷贝

程序的局部变量存在于（堆栈）中，全局变量存在于（静态区 ）中，动态申请数据存在于（ 堆）中。

50.设有以下说明和定义：

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

则语句 printf("%d",sizeof(struct date)+sizeof(max));的执行结果是？

答 、结果是：___52____。DATE 是一个 union, 变量公用空间. 里面最大的变量类型是 int[5], 占用 20 个字节. 所以它的大小是 20
data 是一个 struct, 每个变量分开占用空间. 依次为 int4 + DATE20 + double8 = 32.
所以结果是 20 + 32 = 52.
当然...在某些 16 位编辑器下, int 可能是 2 字节,那么结果是 int2 + DATE10 + double8 = 20

51.预处理器标识#error 的目的是什么？

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

52.用变量 a 给出下面的定义

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

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

53.关键字 static 的作用是什么？

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

54.关键字 const 是什么含意？

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

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

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

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

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

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

56.下面的代码输出是什么，为什么？

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

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

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

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

这个问题将做为这个测验的一个愉快的结尾。不管你相不相信，上面的例子是完全合乎语法

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

58.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] );
}
```

59.什么是预编译,何时需要预编译?
预编译又称为预处理,是做些代码文本的替换工作。处理#开头的指令,比如拷贝#include 包含的文件代码，#define 宏定义的替换,条件编译等，就是为编译做的预备工作的阶段，主要处理#开始的预编译指令，预编译指令指示了在程序正式编译前就由编译器进行的操作，可以放在程序中的任何位置。
c 编译系统在对程序进行通常的编译之前，先进行预处理。c 提供的预处理功能主要有以下三种：1）宏定义　2）文件包含　3）条件编译
1、总是使用不经常改动的大型代码体。
2、程序由多个模块组成，所有模块都使用一组标准的包含文件和相同的编译选项。在这种情况下，可以将所有包含文件预编译为一个预编译头。

60.进程和线程的区别
什么是进程（Process）：普通的解释就是，进程是程序的一次执行，而什么是线程（Thread），线程可以理解为进程中的执行的一段程序片段。在一个多任务环境中下面的概念可以帮助我们理解两者间的差别：
进程间是独立的，这表现在内存空间，上下文环境；线程运行在进程空间内。 一般来讲（不使用特殊技术）进程是无法突破进程边界存取其他进程内的存储空间；而线程由于处于进程空间内，所以同一进程所产生的线程共享同一内存空间。 同一进程中的两段代码不能够同时执行，除非引入线程。线程是属于进程的，当进程退出时该进程所产生的线程都会被强制退出并清除。线程占用的资源要少于进程所占用的资源。 进程和线程都可以有优先级。在线程系统中进程也是一个线程。可以将进程理解为一个程序的第一个线程。
线程是指进程内的一个执行单元,也是进程内的可调度实体.与进程的区别:
(1)地址空间:进程内的一个执行单元;进程至少有一个线程;它们共享进程的地址空间;而进程有自己独立的地址空间;
(2)进程是资源分配和拥有的单位,同一个进程内的线程共享进程的资源
(3)线程是处理器调度的基本单位,但进程不是.

(4)二者均可并发执行.

61.插入排序和

插入排序基本思想：（假定从大到小排序）依次从后面拿一个数和前面已经排好序的数进行比较，比较的过程是从已经排好序的数中最后一个数开始比较，如果比这个数，继续往前面比较，直到找到比它大的数，然后就放在它的后面，如果一直没有找到，肯定这个数已经比较到了第一个数，那就放到第一个数的前面。那么一般情况下，对于采用插入排序法去排序的一组数，可以先选 取第一个数做为已经排好序的一组数。然后把第二个放到正确位置。选择排序(Selection Sort)是一种简单直观的排序算法。它的工作原理如下。首先在未排序序列中找到最小元素,存放到排序序列的起始位置,然后,再从剩余未排序元素中继续寻找最小元素,然后放到排序序列末尾。以此类推,直到所有元素均排序完毕。

62.运算符优先级问题

能正确表示 a 和 b 同时为正或同时为负的逻辑表达式是(D)。

sssA、(a>=0||b>=0)＆＆(a<0||b<0)

B、(a>=0＆＆b>=0)&&(a<0&＆b<0)

C、(a+b>0)＆＆(a+b<=0)

D、a*b>0

以下关于运算符优先顺序的描述中正确的是(C)。

A、关系运算符<算术运算符<赋值运算符<逻辑与运算符

B、逻辑与运算符<关系运算符<算术运算符<赋值运算符

C、赋值运算符<逻辑与运算符<关系运算符<算术运算符

D、算术运算符<关系运算符<赋值运算符<逻辑与运算符

63. C++中的 class 和 struct 的区别

从语法上，在 C++中（只讨论 C++中）。class 和 struct 做类型定义时只有两点区别：

（一）默认继承权限。如果不明确指定，来自 class 的继承按照 private 继承处理，来自 struct 的继承按照 public 继承处理；

（二）成员的默认访问权限。class 的成员默认是 private 权限，struct 默认是 public 权限。

除了这两点，class 和 struct 基本就是一个东西。语法上没有任何其它区别。

不能因为学过 C 就总觉得连 C++中 struct 和 class 都区别很大，下面列举的说明可能比较无聊，因为 struct 和 class 本来就是基本一样的东西，无需多说。但这些说明可能有助于澄清一些常见的关于 struct 和 class 的错误认识：

（1）都可以有成员函数；包括各类构造函数，析构函数，重载的运算符，友元类，友元结构，友元函数，虚函数，纯虚函数，静态函数；

（2）都可以有一大堆 public/private/protected 修饰符在里边；

（3）虽然这种风格不再被提倡，但语法上二者都可以使用大括号的方式初始化：

A a = {1, 2, 3};不管 A 是个 struct 还是个 class，前提是这个类/结构足够简单，比如所有的成员都是 public 的，所有的成员都是简单类型，没有显式声明的构造函数。

（4）都可以进行复杂的继承甚至多重继承，一个 struct 可以继承自一个 class，反之亦可；一个 struct 可以同时继承 5 个 class 和 5 个 struct，虽然这样做不太好。

（5）如果说 class 的设计需要注意 OO 的原则和风格，那么没任何理由说设计 struct 就不需要注意。

（6）再次说明，以上所有说法都是指在 C++语言中，至于在 C 里的情况，C 里是根本没有“class”，而 C 的 struct 从根本上也只是个包装数据的语法机制。

最后，作为语言的两个关键字，除去定义类型时有上述区别之外，另外还有一点点：“class”

这个关键字还用于定义模板参数，就像“typename”。但关键字“struct”不用于定义模板参数。
关于使用大括号初始化

class 和 struct 如果定义了构造函数的话，都不能用大括号进行初始化

如果没有定义构造函数，struct 可以用大括号初始化。

如果没有定义构造函数，且所有成员变量全是 public 的话，可以用大括号初始化。

关于默认访问权限

class 中默认的成员访问权限是 private 的，而 struct 中则是 public 的。

关于继承方式

class 继承默认是 private 继承，而 struct 继承默认是 public 继承。

关于模版

在模版中，类型参数前面可以使用 class 或 typename，如果使用 struct，则含义不同，struct 后面跟的是“non-type template parameter”，而 class 或 typename 后面跟的是类型参数。
class 中有个默认的 this 指针，struct 没有
不同点：构造函数，析构函数 this 指针

64.三种基本的数据模型

按照数据结构类型的不同，将数据模型划分为层次模型、网状模型和关系模型。

65. 分别划划 OSI 的七层网络结构图，和 TCP/IP 的五层结构图？

OSI 七层：应用层；表示层；会话层；传输层；网络层；数据链路层；物理层。 TCP/IP：应用层；传输层；网络层；数据链路层；物理层。 工作在 OSI 模型下的网络设备

物理层：中继器、集线器、还有我们通常说的双绞线也工作在物理层

数据链路层：网桥（现已很少使用）、以太网交换机（二层交换机）、网卡（其实网卡是一半工作在物理层、一半工作在数据链路层）

网络层：路由器、三层交换机

传输层：四层交换机、也有工作在四层的路由器

交换机最高还有七层的交换机，应用在电信骨干网络，提供网络高带宽和低时延

66. 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？

IP 协议(Internet Protocol)又称互联网协议，是支持网间互连的数据报协议，它与 TCP 协议（传输控制协议）一起构成了 TCP/IP 协议族的核心。它提供网间连接的完善功能， 包括 IP 数据报规定互连网络范围内的 IP 地址格式。

传输层主要为两台主机上的应用程序提供端到端的数据通信，它分为两个不同的协议：TCP（传输控制协议）和 UDP（用户数据报协议）。TCP 协议提供端到端的质量保证的数据传输，该层数据分组、质量控制和超时重发等，对于应用层来说，可以忽略这些工作。 UDP 协议则只是提供简单的把数据报从一端发送到另一端，至于数据是否到达或按时到达、数据是否损坏都必须由应用层来负责。这两种协议各有各自的用途，前者可用于面向连接的应用，而后者在及时性服务中重要的用途，如网络多媒体通信等。

67. 请问交换机和路由器分别的实现原理是什么？分别在哪个层次上面实现的？

交换机的功能都是在数据链路层实现。数据链路层的作用主要是控制数据流量，处理传输错误，提供物理地址（没有逻辑地址），以及管理对物理介质的访问。通过使用不同的链路层协议，网桥可以实现上述所有的功能。目前较为流行的链路层协议包括：以太网，令牌环以及 FDDI 等。 交换机的功能实现原理并不复杂，主要是通过分析流入的数据帧，根据帧中包含的信息做出转发决策，然后再把数据帧转发到目的地。网桥对数据帧的转发分为两种形式，如果使用的是源路径网桥技术，那么每一个数据帧中都已经包含了到达目的地的完整

路径；如果使用的是透明网桥技术，那么每一次数据帧都会被转发到下一个节点并最终到达目的地。

路由器是一种典型的网络层设备。它是两个局域网之间接帧传输数据，在 OSI / RM 之中被称之为中介系统，完成网络层中继或第三层中继的任务。路由器负责在两个局域网的网络层间接帧传输数据，转发帧时需要改变帧中的地址。

路由器（Router）是用于连接多个逻辑上分开的网络，所谓逻辑网络是代表一个单独的网络或者一个子网。当数据从一个子网传输到另一个子网时，可通过路由器来完成。因此，路由器具有判断网络地址和选择路径的功能，它能在多网络互联环境中，建立灵活的连接，可用完全不同的数据分组和介质访问方法连接各种子网，路由器只接受源站或其他路由器的信息，属网络层的一种互联设备。它不关心各子网使用的硬件设备，但要求运行与网络层协议相一致的软件。路由器分本地路由器和远程路由器，本地路由器是用来连接网络传输介质的，如光纤、同轴电缆、双绞线；远程路由器是用来连接远程传输介质，并要求相应的设备，如电话线要配调制解调器，无线要通过无线接收机、发射机。 一般说来，异种网络互联与多个子网互联都应采用路由器来完成。路由器的主要工作就是为经过路由器的每个数据帧寻找一条最佳传输路径，并将该数据有效地传送到目的站点。由此可见，选择最佳路径的策略即路由算法是路由器的关键所在。为了完成；这项工作，在路由器中保存着各种传输路径的相关数据——路径表（Routing Table），供路由选择；时使用。路径表中保存着子网的标志信息、网上路由器的个数和下一个路由器的名字等内容。路径表可以是由系统管理员固定设置好的，也可以由系统动态修改，可以由路由器自动调整，也可以由主机控制。

试题 21：华为全真试题

1.请你分别划划 **OSI** 的七层网络结构图，和 **TCP/**IP 的五层结构图？

OSI/ISO 根据整个计算机网络功能将网络分为：物理层、数据链路层、网络层、传输层、会话层、表示层、应用层七层。也称"七层模型"。

TCP/IP"五层模型"分为：物理层、网络接口层、网络层**(IP** 层**)**、传输层**(TCP/UDP** 层**)**、应用层。

2.请你详细的解释一下 **IP** 协议的定义，在哪个层上面，主要有什么作用？ **TCP** 与 **UDP** 呢？

IP 协议是网络层的协议，它实现了自动路由功能，也就是寻径的功能。**TCP** 协议是传输层的协议，它向下屏蔽了 **IP** 协议不可靠传输的特性，向上提供一个可靠的点到点的传输；**UDP** 也是传输层的协议，提供的是一种无连接的服务**,**主要考虑到很多应用不需要可靠的连接，但需要快速的传输，如局域网中的计算机传输文件一般使用 **UDP** 协议。

3.请问交换机和路由器分别的实现原理是什么？分别在哪个层次上面实现的？

交换机用在局域网中，交换机通过记录局域网内各节点机器的 **MAC** 地址就可以实现传递报文**,**无需看报文中的 **IP** 地址。路由器识别不同网络的方法是通过识别不同网络的网络 **ID** 号**(IP** 地址的高端部分**)**进行的，所以为了保证路由成功，每个网络都必须有一个唯一的网络编号。路由器通过察看报文中 **IP** 地址来决定路径，向那个子网**(**下一跳**)**路由。也就是说交换机工作在数据链路层看 **MAC** 地址，路由器工作在网际层看 **IP** 地址。但是由于现在网络设备的发展，很多设备既有交换机的功能有由路由器的功能**(**交换式路由器**)**使得两者界限越来越模糊。

4.请问 **C++**的类和 **C** 里面的 **struct** 有什么区别？

C++的 **class** 具有数据封装功能，其包含属性访问级别可以为 **private,public** 和

protect,还具有实现类接口功能和辅助功能的操作函数，而 **struct** 属性访问权限只有 **public**，没有数据封装功能，也就没有实现信息隐藏这一面向对象的思想的机制**,struct** 本身不含有操作函数，只有数据。

5.请讲一讲析构函数和虚函数的用法和作用？

析构函数是在类对象死亡时由系统自动调用，其作用是用来释放对象的指针数据成员所指的动态空间**,**如果在构造函数中，你申请了动态空间，那么为了避免引起程序错误，你必须在析构函数中释放这部分内存空间。如果基类的函数用 **virtual** 修饰，成为虚函数，则其派生类相应的重载函数仍能继承该虚函数的性质，虚函数进行动态联编，也即具有多态性**,** 也就是派生类可以改变基类同名函数的行为，在面向对象世界中，多态是最强大的机制，虚函数就是这一机制的 **c++**实现方式。

6.全局变量和局部变量有什么区别？实怎么实现的？操作系统和编译器是怎么知道的？

全局变量是整个程序都可访问的变量，谁都可以访问，生存期在整个程序从运行到结束**(**在程序结束时所占内存释放**)**；而局部变量存在于模块**(**子程序，函数**)**中，只有所在模块可以访问，其他模块不可直接访问，模块结束**(**函数调用完毕**)**，局部变量消失，所占据的内存释放。操作系统和编译器，可能是通过内存分配的位置来知道的，全局变量分配在全局数据段并且在程序开始运行的时候被加载**.** 局部变量则分配在堆栈里面。

7．**8086** 是多少位的系统？在数据总线上是怎么实现的？

答：**8086** 的机器字长是 **16** 位，**8086** 使用 **40** 个引脚的 **16** 个做地址**/**数据复用引腿来传输数据，一次读写过程由一个基本总线周期完成，它由 **4** 个时钟**(CLK)**周期组成，按时间顺序定义为 **T1**、**T2**、**T3**、**T4**。在 **T1** 期间 **8086** 发出访问目的地的地址信号和地址锁存选通信号 **ALE**；**T2** 期间发出读写命令信号 **RD**、**WR** 及其它相关信号；**T3** 期间完成数据的访问；**T4** 结束该总线周期。可见，地址与数据信号不会同时出现在一个时钟**(CLK)**周期，二者可以分时复用同一组引线。

一、判断题（对的写 **T**，错的写 **F** 并说明原因，每小题 **4** 分，共 **20** 分）

1、有数组定义 **int a[2][2]={{1},{2,3}};**则 **a[0][1]**的值为 **0**。（　正确　）

2、**int (*ptr) (),**则 **ptr** 是一维数组的名字。（ 错误　**int (*ptr) ();**定义一个指向函数的指针变量　）

3、指针在任何情况下都可进行**>,<,>=,<=,==**运算。（　错误　）

4、**switch(c)** 语句中 **c** 可以是 **int ,long,char ,float ,unsigned int** 类型。（ 错，不能用实形 ）

二、填空题（共 **30** 分）

1、在 **windows** 下，写出运行结果，每空 **2** 分，共 **10** 分。

```
char str[ ]=  " Hello " ;
char *p=str;
int n=10;
sizeof(str)=(      )
sizeof(p)=(       )
sizeof(n)=(       )
```

```
void func(char str[100]){    }
sizeof(str)=(     )
```

答案：**6**，**4**，**4**，**4,** 具体解释请参看我的空间里的"C/C++程序员应聘试题剖析"

2、

```
void getmemory(char **p, int num)
{ *p=(char *) malloc(num);}
void test(void)
{  char *str=NULL;
   getmemory(&str,100);
   strcpy(str, " hello " );
   printf(str);
}
```

运行 **test** 函数有什么结果？（　　）**10** 分

答案：输出 **hello**，但是发生内存泄漏。

3、设

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

(　　) 10 分

答案：**8**，**8**。这道题目的意义不大，因为在不同的编译器里 **printf** 的参数的方向是不一样的，在 **vc6.0** 下是从有到左，这里先***(++ptr)** 后***pt**，于是结果为 **8**，**8**

二、编程题（第一小题 **20**，第二小题 **30** 分）

1、 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**；

答案：一、

```
int strcmp(char  *source, char *dest)
{
 assert((source!=NULL)&&(dest!=NULL));
 int i,j;
 for(i=0; source[i]==dest[i]; i++)
 {
  if(source[i]=='\0' && dest[i]=='\0')
   return 0;
  else
   return -1;
 }
}
```

答案：二、

```
int strcmp(char *source, char *dest)
```

```
{
    while ( (*source != '\0') && (*source == *dest))
    {
source++;
dest++;
    }

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

}
```

2、 写一函数 **int fun(char *p)**判断一字符串是否为回文，是返回 **1**，不是返回 **0**，出错返回**-1**

```
int fun(char *p)
{
 if(p==NULL)
  return -1;
 else
 {
  int length = 0;
  int i = 0;
  int judge = 1;
  length = strlen(p);
  for(i=0; i<length/2; i++)
  {
   if(p[i]!=p[length-1-i])
    judge = 0;
   break;
  }
  if(judge == 0)
   return 0;
  else
   return 1;
 }
```