嵌入式C开发人员的最好的0x10道笔试题
2006-11-22 15:53

约定:
1) 下面的测试题中，认为所有必须的头文件都已经正确的包含了
2)数据类型
char 一个字节 1 byte
int 两个字节 2 byte (16位系统，认为整型是2个字节)
long int 四个字节 4 byte
float 四个字节4 byet
double 八个字节 8 byte
long double 十个字节 10 byte
pointer 两个字节 2 byte(注意，16位系统,地址总线只有16位)

第1题: 考查对volatile关键字的认识

```
#include<setjmp.h>
static jmp_buf  buf;
main()
{
  volatile  int b;
  b =3;
  if(setjmp(buf)!=0)
  {
    printf("%d ", b);
    exit(0);
  }
  b=5;
  longjmp(buf , 1);
}
```

请问，这段程序的输出是
(a) 3
(b) 5
(c) 0
(d) 以上均不是

第2题:考查类型转换

```
main()
{
   struct node
   {
     int a;
     int b;
     int c;
   };
   struct node  s= { 3, 5,6 };
   struct node *pt = &s;
   printf("%d" ,  *(int*)pt);
}
```

这段程序的输出是:
(a) 3
(b) 5
(c) 6
(d) 7

第3题:考查递归调用

```
 int  foo ( int x , int  n)
{
  int val;
  val =1;

  if (n>0)
  {
    if (n%2 == 1)  val = val *x;

    val = val * foo(x*x , n/2);
  }
  return val;
}
```

这段代码对x和n完成什么样的功能(操作)?
(a) x^n (x的n次幂)
(b) x*n(x与n的乘积)
(c) n^x(n的x次幂)
(d) 以上均不是

第**4**题**:**考查指针**,**这道题只适合于那些特别细心且对指针和数组有深入理解的人

```
main()
{
  int  a[5] = {1,2,3,4,5};
  int *ptr =  (int*)(&a+1);
  printf("%d %d" , *(a+1), *(ptr-1) );
}
```

这段程序的输出是**:**

(a) 2 2
(b) 2 1
(c) 2 5
(d) 以上均不是

第**5**题**:**考查多维数组与指针

```
void foo(int [][3] );
main()
{
  int a [3][3]= { { 1,2,3} , { 4,5,6},{7,8,9}};
  foo(a);
  printf("%d" , a[2][1]);
}
void foo( int b[][3])
{
  ++ b;
  b[1][1] =9;
}
```

这段程序的输出是**:**

(a) 8
(b) 9
(c) 7
(d)以上均不对

第**6**题目**:**考查逗号表达式

```
main()
{
  int a, b,c, d;
  a=3;
  b=5;
  c=a,b;
  d=(a,b);
  printf("c=%d" ,c);
  printf("d=%d" ,d);
}
```

这段程序的输出是**:**

(a) c=3 d=3
(b) c=5 d=3
(c) c=3 d=5
(d) c=5 d=5

第**7**题**:**考查指针数组

```
main()
{
  int a[][3] = { 1,2,3 ,4,5,6};
  int (*ptr)[3] =a;
  printf("%d %d "  ,(*ptr)[1], (*ptr)[2] );
  ++ptr;
  printf("%d %d"  ,(*ptr)[1], (*ptr)[2] );
}
```

这段程序的输出是**:**

(a) 2 3 5 6
(b) 2 3 4 5
(c) 4 5 0 0
(d) 以上均不对

第8题:考查函数指针

```
int *f1(void)
{
  int x =10;
  return(&x);
}
int *f2(void)
{
  int*ptr;
  *ptr =10;
  return ptr;
}
int *f3(void)
{
  int *ptr;
  ptr=(int*) malloc(sizeof(int));
  return ptr;
}
```

上面这3个函数哪一个最可能引起指针方面的问题

(a) 只有 **f3**
(b) 只有**f1 and f3**
(c) 只有**f1 and f2**
(d) f1 , f2 ,f3

第9题:考查自加操作(++)

```
main()
{
  int i=3;
  int j;
  j = sizeof(++i+ ++i);
  printf("i=%d j=%d", i ,j);
}
```

这段程序的输出是:

(a) i=4 j=2
(b) i=3 j=2
(c) i=3 j=4
(d) i=3 j=6

第10题:考查形式参数，实际参数，指针和数组

```
void f1(int *, int);
void f2(int *, int);
void(*p[2]) ( int *, int);
main()
{
  int a;
  int b;
  p[0] = f1;
  p[1] = f2;
  a=3;
  b=5;
  p[0](&a , b);
  printf("%d\t %d\t" , a ,b);
  p[1](&a , b);
  printf("%d\t %d\t" , a ,b);
}
void f1( int* p , int q)
{
  int tmp;
  tmp =*p;
  *p = q;
  q= tmp;
}
void f2( int* p , int q)
{
  int tmp;
  tmp =*p;
  *p = q;
  q= tmp;
}
```

这段程序的输出是:

(a) 5 5 5 5
(b) 3 5 3 5
(c) 5 3 5 3

(d) 3 3 3 3

第**11**题:考查自减操作**(--)**

```
void e(int );
main()
{
  int a;
  a=3;
  e(a);
}
void e(int n)
{
  if(n>0)
  {
    e(--n);
    printf("%d" , n);
    e(--n);
  }
}
```

这段程序的输出是:

(a) 0 1 2 0
(b) 0 1 2 1
(c) 1 2 0 1
(d) 0 2 1 1

第**12**题:考查**typedef**类型定义,函数指针

```
typedef int (*test) ( float * , float*)
test tmp;
```

tmp 的类型是

(a) 函数的指针，该函数以 两个指向浮点数**(float)**的指针**(pointer)**作为参数**(arguments)**
Pointer to function of having two arguments that is pointer to float
(b) 整型
(c) 函数的指针，该函数以 两个指向浮点数**(float)**的指针**(pointer)**作为参数**(arguments),**并且函数的返回值类型是整型
Pointer to function having two argument that is pointer to float and return int
(d) 以上都不是

第**13**题:数组与指针的区别与联系

```
main()
{
  char p;
  char buf[10] ={ 1,2,3,4,5,6,9,8};
  p = (buf+1)[5];
  printf("%d" , p);
}
```

这段程序的输出是:

(a) 5
(b) 6
(c) 9
(d) 以上都不对

第**14**题: 考查指针数组的指针

```
Void f(char**);
main()
{
  char * argv[] = { "ab" ,"cd" , "ef" ,"gh", "ij" ,"kl" };
  f( argv );
}
void f( char **p )
{
  char* t;
  t= (p+= sizeof(int))[-1];
  printf( "%s" , t);
}
```

这段程序的输出是:

(a) ab
(b) cd
(c) ef

(d) gh

第**15**题**:**此题考查的是**C**的变长参数**,**就像标准函数库里**printf()**那样**,**这个话题一般国内大学课堂是不会讲到的**,**不会也情有可原呵呵**,**

```
#include<stdarg.h>
int ripple ( int , ...);
main()
{
  int num;
  num = ripple ( 3, 5,7);
  printf( " %d" , num);
}
int ripple (int n, ...)
{
  int i , j;
  int k;
  va_list p;
  k= 0;
  j = 1;
  va_start( p , n);
  for (; j<n;  ++j)
  {
    i =  va_arg( p , int);
    for (; i;    i &=i-1  )
      ++k;
  }
  return k;
}
```

这段程序的输出是**:**

(a) 7
(b) 6
(c) 5
(d) 3

第**16**题**:**考查静态变量的知识

```
int counter (int i)
{
  static int count =0;
  count = count +i;
  return (count );
}
main()
{
  int i , j;
  for (i=0; i <=5; i++)
    j = counter(i);
}
```

本程序执行到最后**,j**的值是**:**

(a) 10
(b) 15
(c) 6
(d) 7

详细参考答案

第**1**题**: (b)**
volatile字面意思是易于挥发的。这个关键字来描述一个变量时，意味着 给该变量赋值**(**写入**)**之后，马上再读取，写入的值与读取的值可能不一样**,**所以说它**"**容易挥发**"**的。
这是因为这个变量可能一个寄存器，直接与外部设备相连，你写入之后，该寄存器也有可能被外部设备的写操作所改变**;**或者，该变量被一个中断程序，或另一个进程
改变了**.**
volatile 不会被编译器优化影响，在**longjump** 后**,**它的值 是后面假定的变量值**,b**最后的值是**5,**所以**5**被打印出来**.**

setjmp : 设置非局部跳转 **/* setjmp.h*/**

Stores context information such as register values so that the lomgjmp function can return control to the statement following the one calling setjmp.Returns 0 when it is initially called.

Lonjjmp: 执行一个非局部跳转 **/* setjmp.h*/**

Transfers control to the statement where the call to setjmp (which initialized buf) was made. Execution continues at this point as if longjmp cannot return the value 0.A nonvolatile automatic variable might be changed by a call to longjmp.When you use setjmp and longjmp, the only automatic variables guaranteed to remain valid are those declared volatile.

Note: Test program without volatile qualifier (result may very)

更详细介绍，请参阅 C语言的**setjmp**和**longjmp**

第**2**题: **(a)**
结构题的成员在内存中的地址是按照他们定义的位置顺序依次增长的。如果一个结构体的指针被看成 它的第一个成员的指针,那么该指针的确指向第一个成员

第**3**题: **(a)**
此题目较难.

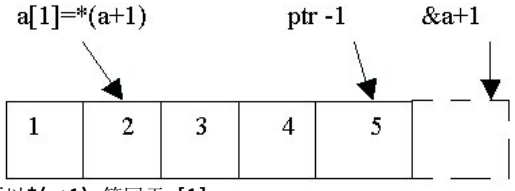
这个程序的非递归版本

```
int  what ( int x , int  n)
{
  int val;
  int product;
  product =1;
  val =x;
  while(n>0)
  {
    if (n%2 == 1)
      product = product*val;   /*如果是奇数次幂,  x(val)
                                   要先乘上一次,;
                                 偶数次幂,最后返回时才会到这里
                                   乘以1*/
    val = val* val;
    n = n/2;
  }
   return product;
}
```

/* 用二元复乘策略 ***/**
算法描述

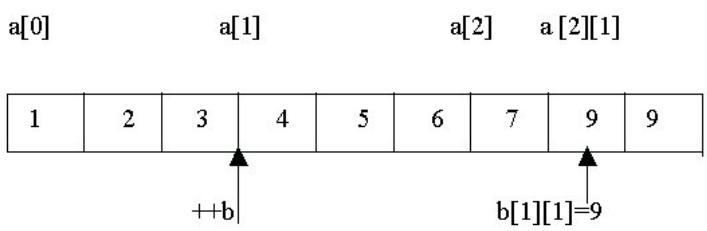
```
(while n>0)
{
  if  next most significant binary digit of  n( power)  is one
  then multiply accumulated product by current val  ,
  reduce n(power)  sequence by a factor of two using integer division .
  get next val by multiply current value of itself
}
```

第**4**题: **(c)**
a的类型是一个整型数组,它有**5**个成员
&a的类型是一个整型数组的指针
所以**&a + 1**指向的地方等同于 **a[6]**



所以***(a+1)** 等同于**a[1]**
ptr等同 **a[6], ptr-1**就等同与**a[5]**

第**5**题: **(b)**



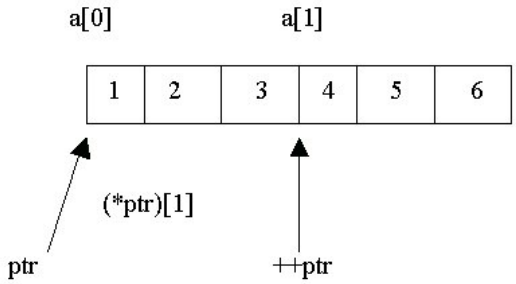
题目自身就给了足够的提示
b[0][0] = 4
b[1][0] = 7

第**6**题: **(c)**
考查逗号表达式**,**逗号表达式的优先级是很低的，比 赋值**(=)**的优先级 低**.** 逗号表达式的值就是最后一个元素的值
逗号表达式的还有一个作用就是分割函数的参数列表**..**

E1, E2, ..., En
上面这个表示式的左右是**,E1, E2,... En**的值被分别计算出来，**En**计算出来的结构赋给整个逗号表达式

```
c=a,b;        / *yields c=a* /
d=(a,b);     /* d =b  */
```

第**7**题: **(a)**



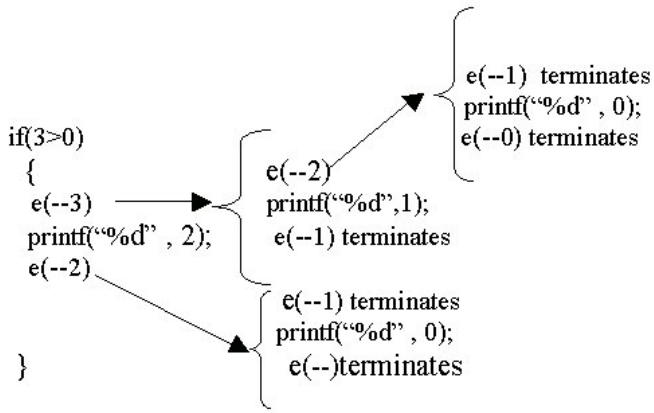
ptr是一个数组的指针，该数组有**3**个**int**成员

第**8**题: **(c)**
f1显然有问题，它返回一个局部变量的指针，局部变量是保存在**stack**中的**,**退出函数后，局部变量就销毁了，保留其指针没有意义，因为其指向的**stack**空间可能被其他变量覆盖了
f2也有问题**, ptr**是局部变量，未初始化，它的值是未知的，***ptr**不知道指向哪里了，直接给***ptr**赋值可能会覆盖重要的系统变量，这就是通常说的野指针的一种

第**9**题: **(b)**
sizeof 操作符给出其操作数需要占用的空间大小，它是在编译时就可确定的，所以其操作数即使是一个表达式，也不需要在运行时进行计算**.(++i + ++ i)**是不会执行的，所以
i 的值还是**3**

第**10**题: **(a)**
很显然选**a.**
f1交换***p** 和 **q**的值，**f1**执行完后**, *p** 和 **q**的值的确交换了**,** 但 **q**的改变不会影响到 **b**的改变**, *p** 实际上就是 **a**
所以执行**f1**后**, a=b=5**
这道题考查的知识范围很广**,**包括**typedef**自定义类型**,**函数指针**,**指针数组
void(*p[2]) (int *, int);
定义了一个函数指针的数组**p,p**有两个指针元素**.** 元素是函数的指针，函数指针指向的函数是一个带**2**个参数**,**返回**void**的函数，所带的两个参数是 指向整型的指针，和整型
p[0] = f1; p[1] = f2 contain address of function .function name without parenthesis represent address of function Value and address of variable is passed to function only argument that is effected is a (address is passed). Because of call by value f1, f2 can not effect b

第**11**题: **(a)**



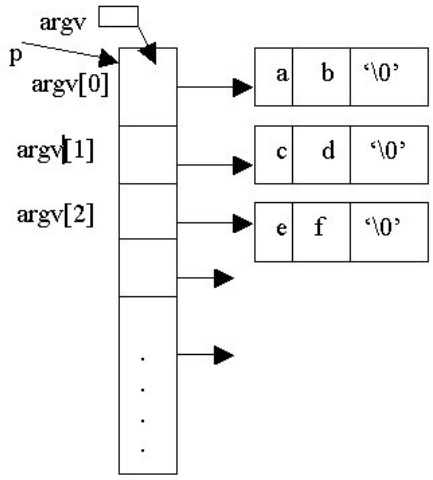
考查--操作和递归调用,仔细分析一下就可以了

第12题: (c)
建议不会的看看C专家编程
从左往有，遇到括号停下来，将第一个括号里的东西看成一个整体

第13题: (c)

考查什么时候数组就是指针.对某些类型T而言，如果一个表达式是 T[] (T的数组), 这个表达式的值实际上就是指向该数组的第一个元素的指针.所以(buf+1)[5]实际上就是*(buf +6)或者buf[6]

第14题: (b)



sizeof(int)的值是2,所以p+=sizeof(int) 指向 argv[2],这点估计大家都没有什么疑问

(p+=sizeof(int))[-1] 指向 argv[1],能理解吗，因为(p+=sizeof(int))[-1] 就相当于 (p+=2)[-1] ,也就是(p+2-1)

第15题: (c)

在C编译器通常提供了一系列处理可变参数的宏，以屏蔽不同的硬件平台造成的差异，增加程序的可移植性。这些宏包括va_start、 va_arg和va_end等。
采用ANSI标准形式时，参数个数可变的函数的原型声明是：
type funcname(type para1, type para2, ...)
这种形式至少需要一个普通的形式参数，后面的省略号不表示省略，而是函数原型的一部分。type是函数返回值和形式参数的类型。

不同的编译器，对这个可变长参数的实现不一样 ， gcc4.x中是内置函数.

关于可变长参数,可参阅

http://www.upsdn.net/html/2004-11/26.html

http://www.upsdn.net/html/2004-11/24.html

程序分析

```
va_list p;  /*定义一个变量 ,保存  函数参数列表 的指针*/
va_start( p , n);     /*用va_start宏 初始化 变量p,
                        va_start宏的第2个参数n  ,
                        是一个固定的参数,
                        必须是我们自己定义的变长函数的最后一个入栈的参数
                        也就是调用的时候参数列表里的第1个参数*/
for (; j<n;  ++j)     /* j从1开始,   遍历所有可变参数  */
{
    i =  va_arg( p , int);      /*va_arg取出当前的参数,
                                  并认为取出的参数是一个整数(int)  */
    for (; i;    i &=i-1  )      /*判断取出的i是否为0*/
      ++k;                          /* 如果i不为0,   k自加,
                                  i与i-1进行与逻辑运算, 直到i 为0
                                  这是一个技巧, 下面会谈到它的功能*/
}
```

当我们调用**ripple**函数时，传递给**ripple**函数的 参数列表的第一个参数**n**的值是**3 .**

va_start 初始化 **p**士气指向第一个未命名的参数**(n**是有名字的参数**) ,**也就是 **is 5 (**第一个**).**

每次对 **va_arg**的调用，都将返回一个参数，并且把 **p** 指向下一个参数**.**

va_arg 用一个类型名来决定返回的参数是何种类型**,**以及在 **var_arg**的内部实现中决定移动多大的距离才到达下一个 参数

```
(; i; i&=i-1) k++         /* 计算i有多少bit被置1 */
```

5用二进制表示是 **(101) 2**
7用二进制表示 **(111) 3**
所以 **k** 返回 **5(2+3),**也即本题应该选**c**

举个例子，就很好理解了

```
令  i= 9 = 1001
     i-1  = 1000
    (i-1) +1 = i
                1000
                  +1
               1 001
```

因为**i**与**i-1**的最右边的那位**(**最低位**)** 肯定是不同，如果**i1,i-1**肯定是**0**，反之亦然**.** **i & i-1** 这个运算，在二相补的数字系统中，将会消除最右边的**1**位

第**16**题**: (b)**
答案是 **(b)**
相传高斯小学一年级的时候就会做这类等比数列的题目了**.**这道题考查的是静态变量的知识**,**当每次调用完函数之后**,**静态变量的值不会丢失**,**这与栈中的临时局部变量明显不同的地方**.**
所以**,**第一次调用**counter(0)**之后**,count =0**
第二次调用 **counter(1)**后 **count = 0+1;**
第三次调用 **counter(2) count = 1+2;** **/* count = count +i */**
第四次调用 **counter(3) count = 3+3;**
第五次调用 **counter(4) count = 6+4;**
第六次调用 **counter(5) count = 10+5;**

命题人信息 **Ashok K. Pathak a member (Research Staff) at Bharat Electronics Limited (CRL) , Ghaziabad. He has been developing embedded application for the past five years .Ashok holds a M.E in computer science and engineering . Ashok recently completed a book about' "Advanced Test in C and Embedded System Programming" , Published by BPB , ND .**