预处理器（Preprocessor）

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，直接写出你是如何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，因为直到嵌入(inline)操作符变为标准C的一部分，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？

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

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 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

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 (0×1<<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）

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。

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，你就当是这个娱乐吧…

动态内存分配（Dynamic memory allocation）

14. 尽管不像非嵌入式计算机那么常见，嵌入式系统还是有从堆（heap）中动态分配内存的过程的。那么嵌入式系统中，动态分配内存可能发生的问题是什么？

这里，我期望应试者能提到内存碎片，碎片收集的问题，变量的持行时间等等。这个主题已经在ESP杂志中被广泛地讨论过了（主要是 P.J. Plauger, 他的解释远远超过我这里能提到的任何解释），所有回过头看一下这些杂志吧！让应试者进入一种虚假的安全感觉后，我拿出这么一个小节目：下面的代码片段的输出是什么，为什么？

char \*ptr;

if ((ptr = (char \*)malloc(0)) == NULL)

puts(“Got a null pointer”);

else

puts(“Got a valid pointer”);

这是一个有趣的问题。最近在我的一个同事不经意把0值传给了函数malloc，得到了一个合法的指针之后，我才想到这个问题。这就是上面的代码，该代码的输出是”Got a valid pointer”。我用这个来开始讨论这样的一问题，看看被面试者是否想到库例程这样做是正确。得到正确的答案固然重要，但解决问题的方法和你做决定的基本原理更重要些。

Typedef

15. Typedef 在C语言中频繁用以声明一个已经存在的数据类型的同义字。也可以用预处理器做类似的事。例如，思考一下下面的例子：

#define dPS struct s \*

typedef struct s \* tPS;

以上两种情况的意图都是要定义dPS 和 tPS 作为一个指向结构s指针。哪种方法更好呢？（如果有的话）为什么？

这是一个非常微妙的问题，任何人答对这个问题（正当的原因）是应当被恭喜的。答案是：typedef更好。思考下面的例子：

dPS p1,p2;

tPS p3,p4;

第一个扩展为

struct s \* p1, p2;

上面的代码定义p1为一个指向结构的指，p2为一个实际的结构，这也许不是你想要的。第二个例子正确地定义了p3 和p4 两个指针。

16. 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。

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。

3. 以下代码中的两个sizeof用法有问题吗？

void UpperCase( char str[] ) // 将 str 中的小写字母转换成大写字母

{

for( size\_t i=0; iif( ‘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。

一个32位的机器,该机器的指针是多少位

指针是多少位只要看地址总线的位数就行了。80386以后的机子都是32的数据总线。所以指针的位数就是4个字节了。

main()

{

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

int \*ptr=(int \*)(&a+1);

printf(“%d,%d”,\*(a+1),\*(ptr-1));

}

输出：2,5

\*(a+1）就是a[1]，\*(ptr-1)就是a[4],执行结果是2，5

&a+1不是首地址+1，系统会认为加一个a数组的偏移，是偏移了一个数组的大小（本例是5个int）

int \*ptr=(int \*)(&a+1);

则ptr实际是&(a[5]),也就是a+5

原因如下：

&a是数组指针，其类型为 int (\*)[5];

而指针加1要根据指针类型加上一定的值，

不同类型的指针+1之后增加的大小不同

a是长度为5的int数组指针，所以要加 5\*sizeof(int)

所以ptr实际是a[5]

但是prt与(&a+1)类型是不一样的(这点很重要)

所以prt-1只会减去sizeof(int\*)

a,&a的地址是一样的，但意思不一样，a是数组首地址，也就是a[0]的地址，&a是对象（数组）首地址，a+1是数组下一元素的地址，即a[1],&a+1是下一个对象的地址，即a[5].

1.请问以下代码有什么问题：

int main()

{

char a;

char \*str=&a;

strcpy(str,”hello”);

printf(str);

return 0;

}

没有为str分配内存空间，将会发生异常

问题出在将一个字符串复制进一个字符变量指针所指地址。虽然可以正确输出结果，但因为越界进行内在读写而导致程序崩溃。

char\* s=”AAA”;

printf(“%s”,s);

s[0]=’B';

printf(“%s”,s);

有什么错？

“AAA”是字符串常量。s是指针，指向这个字符串常量，所以声明s的时候就有问题。

cosnt char\* s=”AAA”;

然后又因为是常量，所以对是s[0]的赋值操作是不合法的。

1、写一个“标准”宏，这个宏输入两个参数并返回较小的一个。

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

2、嵌入式系统中经常要用到无限循环，你怎么用C编写死循环。

while(1){}或者for(;;)

3、关键字static的作用是什么？

定义静态变量

4、关键字const有什么含意？

表示常量不可以修改的变量。

5、关键字volatile有什么含意？并举出三个不同的例子？

提示编译器对象的值可能在编译器未监测到的情况下改变。

int (\*s[10])(int) 表示的是什么啊

int (\*s[10])(int) 函数指针数组，每个指针指向一个int func(int param)的函数。

1.有以下表达式：

int a=248; b=4;int const c=21;const int \*d=&a;

int \*const e=&b;int const \*f const =&a;

请问下列表达式哪些会被编译器禁止？为什么？

\*c=32;d=&b;\*d=43;e=34;e=&a;f=0x321f;

\*c 这是个什么东东，禁止

\*d 说了是const， 禁止

e = &a 说了是const 禁止

const \*f const =&a; 禁止

2.交换两个变量的值，不使用第三个变量。即a=3,b=5,交换之后a=5,b=3;

有两种解法, 一种用算术算法, 一种用^(异或)

a = a + b;

b = a – b;

a = a – b;

or

a = a^b;// 只能对int,char..

b = a^b;

a = a^b;

or

a ^= b ^= a;

3.c和c++中的struct有什么不同？

c和c++中struct的主要区别是c中的struct不可以含有成员函数，而c++中的struct可以。c++中struct和class的主要区别在于默认的存取权限不同，struct默认为public，而class默认为private

4.#include

#include

void getmemory(char \*p)

{

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

strcpy(p,”hello world”);

}

int main( )

{

char \*str=NULL;

getmemory(str);

printf(“%s/n”,str);

free(str);

return 0;

}

程序崩溃，getmemory中的malloc 不能返回动态内存， free（）对str操作很危险

5.char szstr[10];

strcpy(szstr,”0123456789″);

产生什么结果？为什么？

长度不一样，会造成非法的OS

6.列举几种进程的同步机制，并比较其优缺点。

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

共享存储系统

消息传递系统

管道：以文件系统为基础

8.类的静态成员和非静态成员有何区别？

类的静态成员每个类只有一个，非静态成员每个对象一个

9.纯虚函数如何定义？使用时应注意什么？

virtual void f()=0;

是接口，子类必须要实现

10.数组和链表的区别

数组：数据顺序存储，固定大小

连表：数据可以随机存储，大小可动态改变

11.进程死锁的原因

资源竞争及进程推进顺序非法

12.死锁的4个必要条件

互斥、请求保持、不可剥夺、环路

13.死锁的处理

鸵鸟策略、预防策略、避免策略、检测与解除死锁

14. 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

15.ISO的七层模型是什么？tcp/udp是属于哪一层？tcp/udp有何优缺点？

应用层

表示层

会话层

运输层

网络层

物理链路层

物理层

tcp /udp属于运输层

TCP 服务提供了数据流传输、可靠性、有效流控制、全双工操作和多路复用技术等。

与 TCP 不同， UDP 并不提供对 IP 协议的可靠机制、流控制以及错误恢复功能等。由于 UDP 比较简单， UDP 头包含很少的字节，比 TCP 负载消耗少。

tcp: 提供稳定的传输服务，有流量控制，缺点是包头大，冗余性不好

udp: 不提供稳定的服务，包头小，开销小

1) 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) xn  
(b) x\*n  
(c) nx  
(d) 以上均不是  
第4题:考查指针  
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:  
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:  
#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:  
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);}  
The value of j at the end of the execution of the this program is:  
(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 variable isn’t affected by the optimization. Its value after the longjump is the last value variable assumed.  
b last value is 5 hence 5 is printed.  
setjmp : Sets up for nonlocal goto /\* 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: longjmp Performs nonlocal goto /\* 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)

第2题:    (a)  
The members of structures have address in increasing order of their declaration. If a pointer to a structure is cast to the type of a pointer to its first member, the result refers to the first member.  
第3题:   (a)  
Non recursive version of the program  
int   what ( int x ,int n){   int val;  int product;  product =1;val =x;   while(n>0)

{   if (n%2 == 1)  product = product\*val;   n = n/2;   val = val\* val;   }}  
/\* Code raise a number (x) to a large power (n) using binary doubling strategy \*/  
Algorithm description  
(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)  
type of a is array of int  
type of &a is pointer to array of int  
Taking a pointer to the element one beyond the end of an array is sure to work.  
第5题:   (b)

第6题:   (c)  
The comma separates the elements of a function argument list. The comma is also used as an operator in comma expressions. Mixing the two uses of comma is legal, but you must use parentheses to distinguish them. the left operand E1 is evaluated as a void expression, then E2 is evaluated to give the result and type of the comma expression. By recursion, the expression  
E1, E2, …, En  
results in the left-to-right evaluation of each Ei, with the value and type of En giving the result of the whole expression.  
c=a,b;   / \*yields c=a\* /d=(a,b); /\* d =b   \*/

第7题:   (a)  
/\* ptr is pointer to array of 3 int \*/

第8题:   (c)  
f1 and f2 return address of local variable ,when function exit local variable disappeared  
第9题:   (b)  
sizeof operator gives the number of bytes required to store an object of the type of its operand . The operands is either an expression, which is not evaluated ( (++i + ++ i ) is not evaluated so i remain 3 and j is sizeof int that is 2) or a parenthesized type name.  
第10题:   (a)  
void(\*p[2]) ( int \*, int);  
define array of pointer to function accept two argument that is pointer to int and return int. 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)  
第12题:   (c)  
C provide a facility called typedef for creating new data type names, for example declaration  
typedef char string  
Makes the name string a synonym for int .The type string can be used in declaration, cast, etc, exactly the same way that the type int can be. Notice that the type being declared in a typedef appears in the position of a variable name not after the word typedef.  
第13题:   (c)  
If the type of an expression is “array of T” for some type T, then the value of the expression is a pointer to the first object in the array, and the type of the expression is altered to “pointer to T”  
So (buf+1)[5] is equvalent to \*(buf +6) or buf[6]  
第14题:   (b)  
p+=sizeof(int) point to argv[2]  
(p+=sizeof(int))[-1] points to argv[1]  
第15题:   (c)  
When we call ripple value of the first argument passed to ripple is collected in the n that is 3. va\_start initialize p to point to first unnamed argument that is 5 (first argument).Each call of va\_arg return an argument and step p to the next argument. va\_arg uses a type name to determine what type to return and how big a step to take Consider inner loop  
(; i; i&=i-1) k++ /\* count number of   1 bit in i \*  
in five number of 1 bits is (101) 2  
in seven number of 1 bits is (111) 3  
hence k return 5  
example  
let  i=9=1001      i-1= 1000             (i-1) +1 = i                1000  +1    1 001  
The right most 1 bit of i has corresponding 0 bit in i-1 this way i & i-1, in a two complement number system will delete the right most 1 bit I(repeat until I become 0 gives number of 1 bits)  
第16题:   (b)  
The answer is (b)  
Static variable count remain in existence rather than coming and going each time function is called  
so first call counter(0) count =0  
second call counter(1) count = 0+1;  
third call counter(2) count = 1+2; /\* count = count +i \*/  
fourth call counter(3) count = 3+3;  
fifth call counter(4) count = 6+4;  
sixth call counter(5) count = 10+5;