第一部分：基本概念及其它问答题

1、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。

2、“引用”与指针的区别是什么？

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。

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。

流操作符<<和>>、赋值操作符=的返回值、拷贝构造函数的参数、赋值操作符=的参数、其它情况都推荐使用引用。

3、.h 头文件中的 ifndef/define/endif 的作用？

答：防止该头文件被重复引用。

4、#include<file.h> 与 #include "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。

5、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。

6、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。

7、什么是平衡二叉树？

答 ：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。

8、堆栈溢出一般是由什么原因导致的？

答 ： 1.没有回收垃圾资源

2.层次太深的递归调用

9、冒泡排序算法的时间复杂度是什么？

答 ： O(n^2)

10、什么函数不能声明为虚函数？

答：constructor

11、队列和栈有什么区别？

答：队列先进先出，栈后进先出

12、不能做 switch()的参数类型

答 ： switch 的参数不能为实型。

13、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

14、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量，假定你将那个变量写错了，那么在编译期间会报错，如果你用 extern 方式引用时，假定你犯了同样的错误，那么在编译期间不会报错，而在连接期间报错。

15、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。

16、语句 for(； 1 ；)有什么问题？它是什么意思？

答 、和 while(1)相同，无限循环。

17、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。

18、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

19、程序的内存分配

答：一个由 c/C++编译的程序占用的内存分为以下几个部分

1、栈区（stack）—由编译器自动分配释放，存放函数的参数值，局部变量的值等。其操作方式类似于数据结构中的栈。

2、堆区（heap）—一般由程序员分配释放，若程序员不释放，程序结束时可能由 OS 回收。注意它与数据结构中的堆是两回事，分配方式倒是类似于链表，呵呵。

3、全局区（静态区）（static）—全局变量和静态变量的存储是放在一块的，初始化的全局变量和静态变量在一块区域，未初始化的全局变量和未初始化的静态变量在相邻的另一块区域。程序结束后由系统释放。

4、文字常量区—常量字符串就是放在这里的。程序结束后由系统释放。

5、程序代码区—存放函数体的二进制代码

例子程序

这是一个前辈写的，非常详细

```
//main.cpp
   int a=0;       //全局初始化区
   char *p1;      //全局未初始化区
   main()
   {
```

intb;栈

char s[]="abc"; //栈

char *p2; //栈

char *p3="123456"; //123456\0 在常量区，p3 在栈上。

static int c=0; //全局（静态）初始化区

p1 = (char*)malloc(10);

p2 = (char*)malloc(20); //分配得来得 10 和 20 字节的区域就在堆区。

strcpy(p1,"123456"); //123456\0 放在常量区，编译器可能会将它与 p3 所向"123456"优化成一个地方。

}

20、解释堆和栈的区别

答：堆（heap）和栈(stack)的区别

（1）申请方式

stack:由系统自动分配。例如，声明在函数中一个局部变量 int b;系统自动在栈中为 b 开辟空间

heap:需要程序员自己申请，并指明大小，在 c 中 malloc 函数

如 p1=(char*)malloc(10);

在 C++中用 new 运算符

如 p2=(char*)malloc(10);

但是注意 p1、p2 本身是在栈中的。

（2）申请后系统的响应

栈：只要栈的剩余空间大于所申请空间，系统将为程序提供内存，否则将报异常提示栈溢出。

堆：首先应该知道操作系统有一个记录空闲内存地址的链表，当系统收到程序的申请时，

会遍历该链表，寻找第一个空间大于所申请空间的堆结点，然后将该结点从空闲结点链表中删除，并将该结点的空间分配给程序，另外，对于大多数系统，会在这块内存空间中的首地址处记录本次分配的大小，这样，代码中的 delete 语句才能正确的释放本内存空间。另外，由于找到的堆结点的大小不一定正好等于申请的大小，系统会自动的将多余的那部分重新放入空闲链表中。

（3）申请大小的限制

栈：在 Windows 下,栈是向低地址扩展的数据结构，是一块连续的内存的区域。这句话的意思是栈顶的地址和栈的最大容量是系统预先规定好的，在 WINDOWS 下，栈的大小是 2M（也有的说是 1M，总之是一个编译时就确定的常数），如果申请的空间超过栈的剩余空间时，将提示 overflow。因此，能从栈获得的空间较小。

堆：堆是向高地址扩展的数据结构，是不连续的内存区域。这是由于系统是用链表来存储的空闲内存地址的，自然是不连续的，而链表的遍历方向是由低地址向高地址。堆的大小受限于计算机系统中有效的虚拟内存。由此可见，堆获得的空间比较灵活，也比较大。

（4）申请效率的比较：

栈:由系统自动分配，速度较快。但程序员是无法控制的。

堆:是由 new 分配的内存，一般速度比较慢，而且容易产生内存碎片，不过用起来最方便.

另外，在 WINDOWS 下，最好的方式是用 Virtual Alloc 分配内存，他不是在堆，也不是在栈,而是直接在进程的地址空间中保留一块内存，虽然用起来最不方便。但是速度快，也最灵活。

（5）堆和栈中的存储内容

栈：在函数调用时，第一个进栈的是主函数中后的下一条指令（函数调用语句的下一条可执行语句）的地址，然后是函数的各个参数，在大多数的 C 编译器中，参数是由右往左入栈的，然后是函数中的局部变量。注意静态变量是不入栈的。

当本次函数调用结束后，局部变量先出栈，然后是参数，最后栈顶指针指向最开始存的地址，也就是主函数中的下一条指令，程序由该点继续运行。

堆：一般是在堆的头部用一个字节存放堆的大小。堆中的具体内容由程序员安排。

（6）存取效率的比较

```
char s1[]="aaaaaaaaaaaaaa";
```

```
char *s2="bbbbbbbbbbbbbbbb";
```

aaaaaaaaaa 是在运行时刻赋值的；

而 bbbbbbbbbb 是在编译时就确定的；

但是，在以后的存取中，在栈上的数组比指针所指向的字符串(例如

堆)快。

比如：

```
#include

voidmain()

{

char a=1;

char c[]="1234567890";

char *p="1234567890";

a = c[1];

a = p[1];

return;

}
```

对应的汇编代码

```
10:a=c[1];

004010678A4DF1movcl,byteptr[ebp-0Fh]

0040106A884DFCmovbyteptr[ebp-4],cl

11:a=p[1];

0040106D8B55ECmovedx,dwordptr[ebp-14h]

004010708A4201moval,byteptr[edx+1]

004010738845FCmovbyteptr[ebp-4],al
```

第一种在读取时直接就把字符串中的元素读到寄存器 **cl** 中，而第二种则要先把指针值读到 **edx** 中，在根据 **edx** 读取字符，显然慢了。

21、什么是预编译,何时需要预编译?

答：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。

22、关键字 const 是什么含意？

答：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

23、关键字 volatile 有什么含意 并给出三个不同的例子。

答：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;
}
```

24、三种基本的数据模型

答：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。

25、结构与联合有和区别？

答：**(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

26、描述内存分配方式以及它们的区别?

答：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

27、请说出 const 与#define 相比，有何优点？

答：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。

28、简述数组与指针的区别？

答：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
}
```

29、分别写出 BOOL,int,float,指针类型的变量 a 与“零”的比较语句。

答：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)

30、如何判断一段程序是由 C 编译程序还是由 C++编译程序编译的？

答：#ifdef __cplusplus

cout<<"c++";

#else

cout<<"c";

#endif

31、论述含参数的宏与函数的优缺点

答：	带参宏	函数
处理时间	编译时	程序运行时
参数类型	没有参数类型问题	定义实参、形参类型
处理过程	不分配内存	分配内存
程序长度	变长	不变
运行速度	不占运行时间	调用和返回占用时间

32、用两个栈实现一个队列的功能？要求给出算法和思路！

答 、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

33、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？

答：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。

34、位操作（Bit manipulation）

答： 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。

35、访问固定的内存位置（Accessing fixed memory locations）

答：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 = 0xaa66;
```

A more obscure approach is:

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。

36、中断（Interrupts）

答： 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("\nArea = %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。

37、动态内存分配（Dynamic memory allocation）

答：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。

38、Typedef

答：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。

39、用变量 **a** 给出下面的定义

答：**a)** 一个整型数（**An integer**）

int a

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

int *p

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**）

int **p

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

int a[10]

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**）

int *a[10]

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

int *a[10]

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**）

int fuc(int x)

int *p=fuc

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

40、解释局部变量、全局变量和静态变量的含义。

答：

41、写一个“标准”宏

答：交换两个参数值的宏定义为：**. #define SWAP(a,b)**

(a)=(a)+(b);

(b)=(a)-(b);

(a)=(a)-(b);

输入两个参数，输出较小的一个：**#define MIN(A,B) ((A) < (B))? (A) : (B))**

表明 1 年中有多少秒（忽略闰年问题）：**#define SECONDS_PER_YEAR (60 * 60 * 24 * 365)UL**

#define DOUBLE(x) x+x 与 **#define DOUBLE(x)**（（x）+（x））

i = 5*DOUBLE(5)； i 为 30 **i = 5*DOUBLE(5)； i 为 50**

已知一个数组 table，用一个宏定义，求出数据的元素个数

```
#define NTBL
#define NTBL (sizeof(table)/sizeof(table[0]))
```

42、A.c 和 B.c 两个 c 文件中使用了两个相同名字的 static 变量,编译的时候会不会有问题?这两个 static 变量会保存到哪里（栈还是堆或者其他的）?

答：static 的全局变量，表明这个变量仅在本模块中有意义，不会影响其他模块。

他们都放在数据区，但是编译器对他们的命名是不同的。

如果要使变量在其他模块也有意义的话，需要使用 extern 关键字。

43、一个单向链表，不知道头节点,一个指针指向其中的一个节点，问如何删除这个指针指向的节点？

答：将这个指针指向的 next 节点值 copy 到本节点，将 next 指向 next->next,并随后删除原 next 指向的节点。

第二部分：程序代码评价或者找错

1、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。

2、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。

3、 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。

4、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

5、请写出下列代码的输出内容

```
#include<stdio.h>
main()
{
int a,b,c,d;
a=10;
b=a++;
c=++a;
d=10*a++;
printf("b，c，d：%d，%d，%d"，b，c，d）;
return 0;
}
```

答：10，12，120

6、写出下列代码的输出内容

```
#include<stdio.h>
int inc(int a)
{
return(++a);
```

```
}
int multi(int*a,int*b,int*c)
{
return(*c=*a**b);
}
typedef int(FUNC1)(int in);
typedef int(FUNC2) (int*,int*,int*);

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

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

答：110

7、请找出下面代码中的所以错误

说明：以下代码是把一个字符串倒序，如“abcd”倒序后变为“dcba”

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

答：

方法 1：

```
int main(){
```

```
char* src = "hello,world";
int len = strlen(src);
char* dest = (char*)malloc(len+1);//要为\0 分配一个空间
char* d = dest;
char* s = &src[len-1];//指向最后一个字符
while( len-- != 0 )
*d++=*s--;
*d = 0;//尾部要加\0
printf("%s\n",dest);
free(dest);// 使用完，应当释放空间，以免造成内存汇泄露
return 0;
}
```

方法 2：

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

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

8、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;
```

答案：把循环语句内外换一下

9、请问下面程序会出现什么情况?

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

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

答案：死循环

10、以下 **3** 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**

11、写出下面的结果

```
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。

12、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; 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。

13、写出输出结果

```
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].

14、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。

15、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

16、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;
```

17、下面的程序会出现什么结果

```
.#include <stdio.h>
   #include <stdlib.h>
   void getmemory(char *p)
   {
      p=(char *) malloc(100);
      strcpy(p,"hello world");
   }
   int main( )
   {
      char *str=NULL;
      getmemory(str);
      printf("%s/n",str);
      free(str);
```

```
    return 0;
}
```

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

18、下面的语句会出现什么结果？

```
char szstr[10];
    strcpy(szstr,"0123456789");
```

答案：长度不一样，会造成非法的 OS，应该改为 char szstr[11]；

19、(void *)ptr 和 (*(void**))ptr 的结果是否相同？

答：其中 ptr 为同一个指针

.(void *)ptr 和 (*(void**))ptr 值是相同的

20、问函数既然不会被其它函数调用，为什么要返回 1？

```
int main()
    {
      int x=3;
      printf("%d",x);
      return 1;
      }
```

答：mian 中，c 标准认为 0 表示成功，非 0 表示错误。具体的值是某

中具体出错信息

21、对绝对地址 **0x100000** 赋值且想让程序跳转到绝对地址是 **0x100000** 去执行

(unsigned int*)0x100000 = 1234;

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)();**

22、输出多少？并分析过程

unsigned short A = 10;

printf("~A = %u\n", ~A);

char c=128;

printf("c=%d\n",c);

第一题，～**A** ＝**0xfffffff5,int** 值 为－**11**，但输出的是 **uint**。所以输出 **4294967285**

第二题，**c**＝**0x10,**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。

23、分析下面的程序：

```
void GetMemory(char **p,int num)
{
    *p=(char *)malloc(num);
}
int main()
{
    char *str=NULL;
    GetMemory(&str,100);
    strcpy(str,"hello");
    free(str);
    if(str!=NULL)
    {
        strcpy(str,"world");
    }
    printf("\n str is %s",str);
    getchar();
}
```

问输出结果是什么？希望大家能说说原因，先谢谢了

输出 str is world。

free 只是释放的 str 指向的内存空间,它本身的值还是存在的.

所以 free 之后，有一个好的习惯就是将 str=NULL.

此时 str 指向空间的内存已被回收,如果输出语句之前还存在分配空间的操作的话,这段存储空间是可能被重新分配给其他变量的,

尽管这段程序确实是存在大大的问题(上面各位已经说得很清楚了)，但是通常会打印出 world 来。

这是因为，进程中的内存管理一般不是由操作系统完成的，而是由库函数自己完成的。

当你 malloc 一块内存的时候，管理库向操作系统申请一块空间（可能会比你申请的大一些)，然后在这块空间中记录一些管理信息（一般是在你申请的内存前面一点)，并将可用内存的地址返回。但是释放内存的时候，管理库通常都不会将内存还给操作系统，因此你是可以继续访问这块地址的，只不过。。。。。。。楼上都说过了，最好别这么干。

24、char a[10],strlen(a)为什么等于 15？运行的结果

```
#include "stdio.h"
#include "string.h"

void main()
{
char aa[10];
```

```
printf("%d",strlen(aa));
}
```

sizeof()和初不初始化，没有关系；

strlen()和初始化有关。

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

25、long a=0x801010;a+5=?

答：0x801010 用二进制表示为：“1000 0000 0001 0000 0001 0000”，十进制的值为 8392720，再加上 5 就是 8392725

26、给定结构 struct A

```
{
        char t:：4;
        char k:4;
        unsigned short i:8;
        unsigned long m;
};问 sizeof(A) = ?
```

给定结构 struct A

```
{
```

```
        char t:4; 4 位
        char k:4; 4 位
        unsigned short i:8; 8 位
        unsigned long m; //  偏移 2 字节保证 4 字节对齐
}; //  共 8 字节
```

27、下面的函数实现在一个数上加一个数，有什么错误？请改正。

```
int add_n ( int n )
{
    static int i = 100;
    i += n;
    return i;
}
```

当你第二次调用时得不到正确的结果，难道你写个函数就是为了调用一次？问题就出在 static 上

28、给出下面程序的答案

```
#include<iostream.h>
#include <string.h>
#include <malloc.h>
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
```

```
typedef struct    AA
{
        int b1:5;
        int b2:2;
}AA;
void main()
{
        AA aa;
        char cc[100];
          strcpy(cc,"0123456789abcdefghijklmnopqrstuvwxyz");
       memcpy(&aa,cc,sizeof(AA));
        cout << aa.b1 <<endl;
        cout << aa.b2 <<endl;
}
```

答案是 -16 和 1

首先 sizeof(AA)的大小为 4,b1 和 b2 分别占 5bit 和 2bit.

经过 strcpy 和 memcpy 后,aa 的 4 个字节所存放的值是:

0,1,2,3 的 ASC 码，即 00110000,00110001,00110010,00110011

所以，最后一步：显示的是这 4 个字节的前 5 位，和之后的 2 位

分别为：10000,和 01

因为 int 是有正负之分　　所以：答案是-16 和 1

29、求函数返回值，输入 x=9999;

```
int func （ x ）
{
    int countx = 0;
    while ( x )
    {
        countx ++;
        x = x&(x-1);
    }
    return countx;
}
```

结果呢？

知道了这是统计 9999 的二进制数值中有多少个 1 的函数，且有

9999＝9×1024＋512＋256＋15

9×1024 中含有 **1** 的个数为 **2**；

512 中含有 **1** 的个数为 **1**；

256 中含有 **1** 的个数为 **1**；

15 中含有 **1** 的个数为 **4**；

故共有 **1** 的个数为 **8**，结果为 **8**。

1000 - 1 = 0111，正好是原数取反。这就是原理。

用这种方法来求 **1** 的个数是很效率很高的。

不必去一个一个地移位。循环次数最少。

30、分析：

```
struct bit
{      int a:3;
        int    b:2;
        int c:3;
};
int main()
{
   bit s;
   char *c=(char*)&s;
     cout<<sizeof(bit)<<endl;
   *c=0x99;
     cout << s.a <<endl <<s.b<<endl<<s.c<<endl;
          int a=-1;
     printf("%x",a);
   return 0;
}
```

输出为什么是

4

1

-1

-4

ffffffff

因为 0x99 在内存中表示为 100 11 001 , a = 001, b = 11, c = 100

当 c 为有符合数时, c = 100, 最高 1 为表示 c 为负数，负数在计算机用补码表示，所以 c = -4;同理

b = -1;

当 c 为有符合数时, c = 100,即 c = 4,同理 b = 3

31、下面这个程序执行后会有什么错误或者效果:

```
#define MAX 255
int main()
{
    unsigned char A[MAX],i;//i 被定义为 unsigned char
    for (i=0;i<=MAX;i++)
        A[i]=i;
}
```

解答：死循环加数组越界访问（C/C++不进行数组越界检查）

MAX=255

数组 A 的下标范围为:0..MAX-1,这是其一..

其二.当 i 循环到 255 时,循环内执行:

```
A[255]=255;
```

这句本身没有问题..但是返回 for (i=0;i<=MAX;i++)语句时,

由于 unsigned char 的取值范围在(0..255),i++以后 i 又为 0 了..无限循环下去.

32、写出 sizeof(struct name1)=,sizeof(struct name2)=的结果

```
struct name1{
    char    str;
    short x;
    int     num;
}

struct name2{
    char str;
    int num;
    short x;
}
```

sizeof(struct name1)=8,sizeof(struct name2)=12

在第二个结构中，为保证 num 按四个字节对齐，char 后必须留出 3 字节的空间;同时为保证整个结构的自然对齐(这里是 4 字节对齐)，在 x 后还要补齐 2 个字节，这样就是 12 字节。

33、struct s1

{

int i: 8;

int j: 4;

int a: 3;

double b;

};

struct s2

{

int i: 8;

int j: 4;

double b;

int a:3;

};

printf("sizeof(s1)= %d\n", sizeof(s1));

printf("sizeof(s2)= %d\n", sizeof(s2));

result: 16, 24

第一个 **struct s1**

{

int i: 8;

```
    int j: 4;
    int a: 3;
    double b;
};
```

理论上是这样的，首先是 i 在相对 0 的位置，占 8 位一个字节，然后，j 就在相对一个字节的位置，由于一个位置的字节数是 4 位的倍数，因此不用对齐，就放在那里了，然后是 a，要在 3 位的倍数关系的位置上，因此要移一位，在 15 位的位置上放下，目前总共是 18 位，折算过来是 2 字节 2 位的样子，由于 double 是 8 字节的，因此要在相对 0 要是 8 个字节的位置上放下，因此从 18 位开始到 8 个字节之间的位置被忽略，直接放在 8 字节的位置了，因此，总共是 16 字节。第二个最后会对照是不是结构体内最大数据的倍数，不是的话，会补成是最大数据的倍数

34、在对齐为 4 的情况下

```
struct BBB
{
    long num；
    char *name;
    short int data;
    char ha;
    short ba[5];
```

}*p;

p=0x1000000;

p+0x200=____;

(Ulong)p+0x200=____;

(char*)p+0x200=____;

希望各位达人给出答案和原因，谢谢拉

解答：假设在 **32** 位 **CPU** 上，

sizeof(long) = 4 bytes

sizeof(char *) = 4 bytes

sizeof(short int) = sizeof(short) = 2 bytes

sizeof(char) = 1 bytes

由于是 **4** 字节对齐，

sizeof(struct BBB) = sizeof(*p)

= 4 + 4 + 2 + 1 + 1/*补齐*/ + 2*5 + 2/*补齐*/ = 24 bytes (经 **Dev-C++** 验证)

p=0x1000000;

p+0x200=____;

= 0x1000000 + 0x200*24

(Ulong)p+0x200=____;

= 0x1000000 + 0x200

(char*)p+0x200=____;

= 0x1000000 + 0x200*4

35、找错

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

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

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

36、写出输出结果

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

```
printf("%d\n",line[i]);
return 0;
}

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

输出：

```
1
2
3
4
5
6
7
8
9
10
```

37、写出程序运行结果

```
int sum(int a)
{
auto int c=0;
static int b=3;
c+=1;
b+=2;
return(a+b+c);
}

void main()
{
int I;
int a=2;
for(I=0;I<5;I++)
{
printf("%d,", sum(a));
}
}
```

// static 会保存上次结果，记住这一点，剩下的自己写

输出：8,10,12,14,16,

38、评价代码

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

则 func(1)=?

// b 定义后就没有赋值

```
int a[3];
a[0]=0; a[1]=1; a[2]=2;
int *p, *q;
p=a;
q=&a[2];
```

则 a[q-p]=a[2]

解释：指针一次移动一个 int 但计数为 1

39、请问一下程序将输出什么结果？

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

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

40、写出输出结果

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

```
}test;

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

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

t.a 为 01,输出就是 1

t.b 为 11，输出就是－1

t.c 为 1，输出也是-1

3 个都是有符号数 int 嘛。

这是位扩展问题

01

11

1

编译器进行符号扩展

41、对下面程序进行分析

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

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

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

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

正确应如下

```
void test2()
{
```

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

42、分析：

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

输出：8 8

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

43、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

44、写出输出结果

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

int main()
{
    int b = 3;
```

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

输出：m=7,n=4,b=7(VC6.0)

这种方式和编译器中得函数调用关系相关即先后入栈顺序。不过不同编译器得处理不同。也是因为 C 标准中对这种方式说明为未定义，所以

各个编译器厂商都有自己得理解，所以最后产生得结果完全不同。

因为这样，所以遇见这种函数，我们首先要考虑我们得编译器会如何处理

这样得函数，其次看函数得调用方式，不同得调用方式，可能产生不同得

结果。最后是看编译器优化。

45、找出错误

```
#include    string.h
main(void)
{    char    *src="hello,world";
     char    *dest=NULL;
     dest=(char    *)malloc(strlen(src));
     int    len=strlen(str);
```

```
        char    *d=dest;
        char    *s=src[len];
        while(len--!=0)
           d++=s--;
        printf("%s",dest);
}
```

找出错误!!

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

```
}
```

第三部分：编程题

1、读文件 file1.txt 的内容（例如）：

12

34

56

输出到 file2.txt：

56

34

12

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

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

FILE *fp1;
```

```
FILE *fp2;


fp1 = fopen("a.txt","r");
if(fp1 == NULL)
{printf("error1");
        exit(-1);
}


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


int i = 0;
        int j = 0;


while(fscanf(fp1,"%d",&a[i]) != EOF)
{
i++;
j++;
if(i >= MAX)
```

```
{
MAX = 2 * MAX;
b = (int*)realloc(a,MAX * sizeof(int));
if(b == NULL)
{
printf("error3");
exit(-1);
}
a = b;
}
}

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

fclose(fp1);
fclose(fp2);

return 0;



}
```

2、输出和为一个给定整数的所有组合

例如 **n=5**

5=1+4；5=2+3（相加的数不能重复）

则输出

1，4；2，3。

```
#include <stdio.h>


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


printf("please input the number\n");
scanf("%d",&i);
        if( i % 2 == 0)
                j = i / 2;
else
j = i / 2 + 1;


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

```
return 0;
}

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

3、递规反向输出字符串的例子,可谓是反序的经典例程.

```
void inverse(char *p)
{
      if( *p = = '\0' )
```

```
return;
        inverse( p+1 );
        printf( "%c", *p );
}

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

        return 0;
}
```

对 1 的另一种做法：

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

```
{
        FILE *fr = NULL;
        FILE *fw = NULL;
        fr = fopen("data", "rb");
        fw = fopen("dataout", "wb");
        test(fr, fw);
        fclose(fr);
        fclose(fw);
        return 0;
}
```

4、写一段程序，找出数组中第 k 大小的数，输出数所在的位置。例如{2，4，3，4，7}中，第一大的数是 7，位置在 4。第二大、第三大的数都是 4，位置在 1、3 随便输出哪一个均可。函数接口为：int find_orderk(const int* narry,const int n,const int k)

要求算法复杂度不能是 O(n^2）

谢谢！

可以先用快速排序进行排序，其中用另外一个进行地址查找

代码如下，在 VC++6.0 运行通过。给分吧^-^

```
//快速排序
```

```
#include<iostream>

usingnamespacestd;

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

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

returnlow;
}
```

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

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

intmain ()
{
intnarry[100],addr[100];
intsum = 1,t;

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

while (t != -1)
{
```

```
narry[sum] = t;
addr[sum - 1] = t;
sum++;

cin >> t;
}

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

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

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

intaa = 1;
intkk = 0;
for (;;)
{
```

```
if (aa == k)
break;
if (narry[kk] != narry[kk + 1])
{
aa += 1;
kk++;
}


}


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



return0;
}
```

5、两路归并排序

```
Linklist *unio(Linklist *p,Linklist *q){
linklist *R,*pa,*qa,*ra;
pa=p;
qa=q;
R=ra=p;
while(pa->next!=NULL&&qa->next!=NULL){
if(pa->data>qa->data){
ra->next=qa;
qa=qa->next;
}
else{
ra->next=pa;
pa=pa->next;
}
}
if(pa->next!=NULL)
ra->next=pa;
if(qa->next!=NULL)
ra->next==qa;
return R;
}
```

6、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] );
}
```

7、单连表的建立，把'a'--'z'26 个字母插入到连表中，并且倒叙，还要打印！

方法 1：

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

void main(void)
```

```
{      char c;


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


       p.next=NULL;
}
}
```

方法 2：

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

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

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

int longth = 'z' - 'b';
```

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

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

print(head);
```

8、请列举一个软件中时间换空间或者空间换时间的例子。

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

--->空优

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

9、outputstr 所指的值为 123456789

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

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

if( maxlen < count )
{
maxlen = count;
count = 0;
final = temp;
```

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

10、不用库函数,用 C 语言实现将一整型数字转化为字符串

方法 1：

```
int getlen(char *s){
        int n;
        for(n = 0; *s != '\0'; s++)
                n++;
        return n;
}
void reverse(char s[])
{
```

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

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

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

```
}
```

方法 2:

```
#include <iostream>

using namespace std;


void itochar(int num);


void itochar(int num)

{

int i = 0;

int j ;

char stra[10];

char strb[10];

while ( num )

{

stra[i++]=num%10+48;

num=num/10;

}

stra[i] = '\0';

for( j=0; j < i; j++)

{

strb[j] = stra[i-j-1];
```

```
}
strb[j] = '\0';
cout<<strb<<endl;

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

11、求组合数： 求 n 个数（1....n）中 k 个数的组合....

如：combination(5,3)

要求输出：543，542，541，532，531，521，432，431，421，321，

```
#include<stdio.h>

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

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

int main()
{
int n,m;
printf("Input two numbers:\n");
while( (2!=scanf("%d%*c%d",&n,&m)) )
{
fflush(stdin);
printf("Input error! Again:\n");
}
combination(n,m);
printf("\n");
}
void combination(int m,int n)
{
int temp=m;
push(temp);
while(1)
{
if(1==temp)
```

```
{
if(pop(&temp)&&stack[0]==n)  //当栈底元素弹出&&为可能取的最小值，循环退出
break;
}
else if( push(--temp))
{
printf("%d%d%d    ",stack[0],stack[1],stack[2]);//§&auml;¨ì¤@?
pop(&temp);
}
}
}
int push(int i)
{
stack[++top]=i;
if(top<2)
return 0;
else
return 1;
}
int pop(int *i)
{
```

```
*i=stack[top--];
if(top>=0)
return 0;
else
return 1;
}
```

12、用指针的方法，将字符串“ABCD1234efgh”前后对调显示

```
#include <stdio.h>
#include <string.h>
#include <dos.h>
int main()
{
    char str[] = "ABCD1234efgh";
    int length = strlen(str);
    char * p1 = str;
    char * p2 = str + length - 1;
    while(p1 < p2)
    {
        char c = *p1;
        *p1 = *p2;
        *p2 = c;
```

```
        ++p1;
        --p2;
    }
    printf("str now is %s\n",str);
    system("pause");
    return 0;
}
```

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

```
#include <stdio.h>
double getValue()
{
    double result = 0;
    int i = 2;
    while(i < 42)
    {
        result += 1.0 / i;//一定要使用 1.0 做除数，不能用 1，否则结果将自动转化成整数，即 0.000000
        i += 2;
    }
    return result;
```

```
}
int main()
{
	printf("result is %f\n", getValue());
	system("pause");
	return 0;
}
```

14、有一个数组 a[1000]存放 0--1000;要求每隔二个数删掉一个数，到末尾时循环至开头继续进行，求最后一个被删掉的数的原始下标位置。

以 7 个数为例：

{0,1,2,3,4,5,6,7} 0-->1-->2（删除）-->3-->4-->5(删除)-->6-->7-->0（删除），如此循环直到最后一个数被删除。

方法 1：数组

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

int main()
{
int arr[1000];
```

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

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

}方法 2：链表

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

```
}
head->next=p;
while(p!=p->next)
{
p->next->next=p->next->next->next;
p=p->next->next;
}
cout<<p->data;
return 0;
}
```

方法 3：通用算法

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

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

```
                }
                current_node=R[front_node];
            }
  index=front_node;
  values=a[front_node];

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

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

15、实现 strcmp

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

做是做对了，没有抄搞，比较乱

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

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

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

16、实现子串定位

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

做是做对了，没有抄搞，比较乱

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

//assert((MainStr!=NULL)&&(  SubStr!=NULL));//用断言对输入进行判
```

断

```
while(*MainStr) //内部进行递增
{
p = MainStr;
q = SubStr;
while(*q && *p && *p++ == *q++);
if(!*q )
```

```
{
return MainStr - u +1 ;//MainStr 指向当前起始位，u 指向
}
MainStr ++;
}
return -1;
}
```

17、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)
}
```

18、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;
}
```

19、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;
}
```

20、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;
}
```

21、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';
}
```

22、写一个函数，功能：完成内存之间的拷贝

memcpy source code:

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

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

```
307       }
308       return dst;
309 }
```

23、公司考试这种题目主要考你编写的代码是否考虑到各种情况，是否安全（不会溢出）

各种情况包括：

１、参数是指针，检查指针是否有效

２、检查复制的源目标和目的地是否为同一个，若为同一个，则直接跳出

３、读写权限检查

４、安全检查，是否会溢出

memcpy 拷贝一块内存，内存的大小你告诉它

strcpy 是字符串拷贝，遇到'\0'结束

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

指向的内存是否重叠

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

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

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

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

```
char * search(char *cpSource, char ch)
{
        char *cpTemp=NULL, *cpDest=NULL;
        int iTemp, iCount=0;
        while(*cpSource)
```

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

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

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

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

27、给定字符串 A 和 B,输出 A 和 B 中的最大公共子串。

比如 A="aocdfe" B="pmcdfa" 则输出"cdf"

```
*/
//Author: azhen
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

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

char *substring=malloc(256);

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

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


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

gets(str1);
gets(str2);
```

```
if(strlen(str1)>strlen(str2))                    //将短的字符
```
串放前面

```
comman=commanstring(str2, str1);
else
comman=commanstring(str1, str2);


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

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

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

```
                ret = -1 ;
        else if ( ret > 0 )
                ret = 1 ;
        return( ret );
}
```

29、求 1000！的未尾有几个 0（用素数相乘的方法来做，如 72=2*2*2*3*3）；

求出 1->1000 里,能被 5 整除的数的个数 n1,能被 25 整除的数的个数 n2,能被 125 整除的数的个数 n3,

能被 625 整除的数的个数 n4.

1000!末尾的零的个数=n1+n2+n3+n4;

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

int find5(int num){
int ret=0;
while(num%5==0){
num/=5;
ret++;
}
return ret;
```

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

30、有双向循环链表结点定义为：

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

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

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

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

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

```
delete pNode;

pNode = pNextNode;


bRet = TRUE;

//不要 break 或 return，删除所有

}

else

{

pNode = pNode->next;

}

}


return bRet;

}


void DE(Node *pHeadA, Node *pHeadB)

{

if (pHeadA == NULL || pHeadB == NULL)

{

return;

}
```

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

```
else
{
pNode = pNode->next;
}
}
}
```

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

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

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

```
while(as + 1 < len1 && bs + 1 < len2 && s1[++as] == s2[++bs])

count++;


if(count > maxlen)

{

maxlen = count;

*r1 = s1 + i;

*r2 = s2 + j;

}

}

}

}
```

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

```
char* test3(long num) {

char* buffer = (char*)malloc(11);

buffer[0] = '0';

buffer[1] = 'x';

buffer[10] = '\0';


char* temp = buffer + 2;
```

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

33、输入 N，打印 N*N 矩阵

比如 **N = 3**，打印：

1 2 3

8 9 4

7 6 5

N = 4，打印：

1 2 3 4

12 13 14 5

11 16 15 6

10 9 8 7

解答：

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

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

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

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

	for (i = x+n-1; i > x; i--)	//底部
```

```
            matrix[y+n-1][i] = start++;

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

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

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

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

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

34、斐波拉契数列递归实现的方法如下：

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

请问，如何不使用递归，来实现上述函数？

请教各位高手！

解答：

```
int    Funct( int n )    //    n  为非负整数
{
    int a=0;
    int b=1;
    int c;
    if(n==0) c=1;
    else if(n==1) c=1;
    else for(int i=2;i<=n;i++)    //应该 n 从 2 开始算起
    {
       c=a+b;
       a=b;
       b=c;
```

```
    }
    return c;
}
```

解答：

现在大多数系统都是将低字位放在前面，而结构体中位域的申明一般是先声明高位。

100 的二进制是 **001 100 100**

低位在前　　高位在后

001----s3

100----s2

100----s1

所以结果应该是 **1**

如果先申明的在低位则：

001----s1

100----s2

100----s3

结果是 **4**

1、原题跟 **little-endian**，**big-endian** 没有关系

2、原题跟位域的存储空间分配有关，到底是从低字节分配还是从高字节分配，从 **Dev C++**和 **VC7.1** 上看，都是从低字节开始分配，并且连续分配，中间不空，不像谭的书那样会留空位

3、原题跟编译器有关，编译器在未用堆栈空间的默认值分配上有所

不同，Dev C++未用空间分配为

01110111b，VC7.1 下为 11001100b,所以在 Dev C++下的结果为 5，在 VC7.1 下为 1。

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

35、判断一个字符串是不是回文

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

```
return found;
}
```

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

数组实现：

```
#include <stdio.h>
#include <malloc.h>
int Josephu(int n, int m)
{
   int flag, i, j = 0;
   int *arr = (int *)malloc(n * sizeof(int));
   for (i = 0; i < n; ++i)
      arr[i] = 1;
   for (i = 1; i < n; ++i)
   {
      flag = 0;
      while (flag < m)
      {
```

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

链表实现：

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

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

    for (i = 1; tail != head; ++i)
    {
```

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

37、已知 strcpy 函数的原型是：

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

1.不调用库函数，实现 strcpy 函数。

2.解释为什么要返回 char *。

解说：

1.strcpy 的实现代码

```
char * strcpy(char * strDest,const char * strSrc)
{
        if ((strDest==NULL)||(strSrc==NULL)) file://[/1]
                throw "Invalid argument(s)"; //[2]
        char * strDestCopy=strDest;    file://[/3]
        while ((*strDest++=*strSrc++)!='\0'); file://[/4]
        return strDestCopy;
}
```

错误的做法：

[1]

(A)不检查指针的有效性，说明答题者不注重代码的健壮性。

(B) 检查指针的有效性时使用 ((!strDest)||(!strSrc)) 或 (!(strDest&&strSrc))，说明答题者对 C 语言中类型的隐式转换没有深刻认识。在本例中 char *转换为 bool 即是类型隐式转换，这种功能虽然灵活，但更多的是导致出错概率增大和维护成本升高。所以 C++

专门增加了 bool、true、false 三个关键字以提供更安全的条件表达式。

(C)检查指针的有效性时使用((strDest==0)||(strSrc==0))，说明答题者不知道使用常量的好处。直接使用字面常量（如本例中的 0）会减少程序的可维护性。0 虽然简单，但程序中可能出现很多处对指针的检查，万一出现笔误，编译器不能发现，生成的程序内含逻辑错误，很难排除。而使用 NULL 代替 0，如果出现拼写错误，编译器就会检查出来。

[2]

(A)return new string("Invalid argument(s)");，说明答题者根本不知道返回值的用途，并且他对内存泄漏也没有警惕心。从函数中返回函数体内分配的内存是十分危险的做法，他把释放内存的义务抛给不知情的调用者，绝大多数情况下，调用者不会释放内存，这导致内存泄漏。

(B)return 0;，说明答题者没有掌握异常机制。调用者有可能忘记检查返回值，调用者还可能无法检查返回值（见后面的链式表达式）。妄想让返回值肩负返回正确值和异常值的双重功能，其结果往往是两种功能都失效。应该以抛出异常来代替返回值，这样可以减轻调用者的负担、使错误不会被忽略、增强程序的可维护性。

[3]

(A)忘记保存原始的 strDest 值，说明答题者逻辑思维不严密。

[4]

(A)循环写成 while (*strDest++=*strSrc++);，同[1](B)。

(B)循环写成 while (*strSrc!='\0') *strDest++=*strSrc++;，说明答题者对边界条件的检查不力。循环体结束后，strDest 字符串的末尾没有正确地加上'\0'。

第四部分：附加部分

1、位域 ：

有些信息在存储时，并不需要占用一个完整的字节， 而只需占几个或一个二进制位。例如在存放一个开关量时，只有 0 和 1 两种状态，用一位二进位即可。为了节省存储空间，并使处理简便，C 语言又提供了一种数据结构，称为“位域”或“位段”。所谓“位域”是把一个字节中的二进位划分为几个不同的区域，并说明每个区域的位数。每个域有一个域名，允许在程序中按域名进行操作。 这样就可以把几个不同的对象用一个字节的二进制位域来表示。一、位域的定义和位域变量的说明位域定义与结构定义相仿，其形式为：

```
struct 位域结构名
{ 位域列表 };
```

其中位域列表的形式为： 类型说明符 位域名：位域长度

例如：

```
struct bs
{
int a:8;
int b:2;
```

```
int c:6;
};
```

位域变量的说明与结构变量说明的方式相同。可采用先定义后说明，同时定义说明或者直接说明这三种方式。例如：

```
struct bs
{
int a:8;
int b:2;
int c:6;
}data;
```

说明 data 为 bs 变量，共占两个字节。其中位域 a 占 8 位，位域 b 占 2 位，位域 c 占 6 位。对于位域的定义尚有以下几点说明：

1. 一个位域必须存储在同一个字节中，不能跨两个字节。如一个字节所剩空间不够存放另一位域时，应从下一单元起存放该位域。也可以有意使某位域从下一单元开始。例如：

```
struct bs
{
unsigned a:4
unsigned :0 /*空域*/
unsigned b:4 /*从下一单元开始存放*/
unsigned c:4
```

}

在这个位域定义中，a 占第一字节的 4 位，后 4 位填 0 表示不使用，b 从第二字节开始，占用 4 位，c 占用 4 位。

2. 由于位域不允许跨两个字节，因此位域的长度不能大于一个字节的长度，也就是说不能超过 8 位二进位。

3. 位域可以无位域名，这时它只用来作填充或调整位置。无名的位域是不能使用的。例如：

```
struct k
{
int a:1
int :2 /*该 2 位不能使用*/
int b:3
int c:2
};
```

从以上分析可以看出，位域在本质上就是一种结构类型， 不过其成员是按二进位分配的。

二、位域的使用位域的使用和结构成员的使用相同，其一般形式为：位域变量名•位域名 位域允许用各种格式输出。

```
main(){
```

```
struct bs
{
unsigned a:1;
unsigned b:3;
unsigned c:4;
} bit,*pbit;
bit.a=1;
bit.b=7;
bit.c=15;
pri
```

改错：

```
#include <stdio.h>

int main(void) {

    int **p;
    int arr[100];

    p = &arr;

    return 0;
```

}

解答：

搞错了,是指针类型不同,

int **p; //二级指针

&arr; //得到的是指向第一维为 100 的数组的指针

#include <stdio.h>

int main(void) {

int **p, *q;

int arr[100];

q = arr;

p = &q;

return 0;

}