

C/C++面试题大汇总

发布之前先申明两点:

- 1 所有资料来自网络(主要是 CSDN)，本人只是收集和转发。
- 2 所有问题解答(尤其是代码)只是参考，不保证正确。

先发基本问题，再发编程问题.....

想成为嵌入式程序员应知道的 0x10 个基本问题:

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?

```
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
```

人们经常声称这里有几个问题是那种要翻一下书才能回答的问题，我同意这种说法。当我写这篇文章时，为了确定语法的正确性，我的确查了一下书。但是当我被面试的时候，我期望被问到这个问题（或者相近的问题）。因为在被面试的这段时间里，我确定我知道这个问题的答案。应试者如果不知道所有的答案（或至少大部分答案），那么也就没有为这次面试做准备，如果该面试者没有为这次面试做准备，那么他又能为什么出准备呢？

Static

6. 关键字 static 的作用是什么？

这个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。

Const

7. 关键字 const 有什么含意？

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，仔细阅读一下 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。

Volatile

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。搞嵌入式的家伙们经常同硬件、中断、RTOS 等等打交道，所有这些都要求

用到 volatile 变量。不懂得 volatile 的内容将会带来灾难。

假设被面试者正确地回答了这是问题（嗯，怀疑是否会是这样），我将稍微深究一下，

看一下这家伙是不是真正懂得 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) 这段代码有点变态。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，这段代码可能返回你所期望的平方值！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)

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;
```

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。

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("\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。

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。不管怎样，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非常重要的。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。不管怎样，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。晦涩的语法

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，这段代码执行后 $a = 6$, $b = 7$, $c = 12$ 。

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。

华为笔试题

1. 写出判断 ABCD 四个表达式的是否正确，若正确，写出经过表达式中 a 的值(3 分)

```
int a = 4;
```

(A) $a += (a++)$; (B) $a += (++a)$; (C) $(a++) += a$; (D) $(++a) += (a++)$;

$a = ?$

答：C 错误，左侧不是一个有效变量，不能赋值，可改为 $(++a) += a$;

改后答案依次为 9,10,10,11

2. 某 32 位系统下，C++ 程序，请计算 `sizeof` 的值(5 分).

```
char str[] = "http://www.ibegroup.com/";
```

```
char *p = str;
```

```
int n = 10;
```

请计算

`sizeof (str) = ?` (1)

`sizeof (p) = ?` (2)

`sizeof (n) = ?` (3)

```
void Foo ( char str[100]){
```

请计算

`sizeof(str) = ?` (4)

```
}
```

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

请计算

`sizeof (p) = ?` (5)

答: (1) 17 (2) 4 (3) 4 (4) 4 (5) 4

3. 回答下面的问题. (4 分)

(1).头文件中的 `ifndef/define/endif` 干什么用? 预处理

答: 防止头文件被重复引用

(2). `#include` 和 `#include "filename.h"` 有什么区别?

答: 前者用来包含开发环境提供的库头文件, 后者用来包含自己编写的头文件。

(3).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"` 声明?

答: 函数和变量被C++编译后在符号库中的名字与C语言的不同, 被 `extern "C"`修饰的变量和函数是按照C语言方式编译和连接的。由于编译后的名字不同, C++程序不能直接调用C 函数。C++提供了一个C 连接交换指定符号 `extern "C"` 来解决这个问题。

(4). switch()中不允许的数据类型是?

答: 实型

4. 回答下面的问题(6分)

(1).Void GetMemory(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: 输出 “hello”

(2). 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：输出 “world”

```
(3). 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：无效的指针，输出不确定

5. 编写 strcat 函数(6 分)

已知 strcat 函数的原型是 `char *strcat (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 strcat

答：

VC 源码：

```

char * __cdecl strcat (char * dst, const char * src)

{

char * cp = dst;

while( *cp )

cp++; /* find end of dst */

while( *cp++ = *src++ ); /* Copy src to end of dst */

return( dst ); /* return dst */

}

```

(2) strcat 能把 strSrc 的内容连接到 strDest，为什么还要 char * 类型的返回值？

答：方便赋值给其他变量

6.MFC 中 CString 是类型安全类么？

答：不是，其它数据类型转换到 CString 可以使用 CString 的成员函数 Format 来转换

7.C++中为什么用模板类。

答：(1)可用来创建动态增长和减小的数据结构

(2) 它是类型无关的，因此具有很高的可复用性。

(3) 它在编译时而不是运行时检查数据类型，保证了类型安全

(4) 它是平台无关的，可移植性

(5) 可用于基本数据类型

8.CSingleLock 是干什么的。

答：同步多个线程对一个数据类的同时访问

9. NEWTEXTMETRIC 是什么。

答：物理字体结构，用来设置字体的高宽大小

10. 程序什么时候应该使用线程，什么时候单线程效率高。

答：1. 耗时的操作使用线程，提高应用程序响应

2. 并行操作时使用线程，如 C/S 架构的服务器端并发线程响应用户的请求。

3. 多 CPU 系统中，使用线程提高 CPU 利用率

4. 改善程序结构。一个既长又复杂的进程可以考虑分为多个线程，成为几个独立或半独立的运行部分，这样的程序会利于理解和修改。

其他情况都使用单线程。

11. Windows 是内核级线程么。

答：见下一题

12. Linux 有内核级线程么。

答：线程通常被定义为一个进程中代码的不同执行路线。从实现方式上划分，线程有两种类型：“用户级线程”和“内核级线程”。用户线程指不需要内核支持而在用户程序中实现的线程，其不依赖于操作系统核心，应用进程利用线程库提供创建、同步、调度和管理线程的函数来控制用户线程。这种线程甚至在象 DOS 这样的操作系统中也可实现，但线程的调度需要用户程序完成，这有些类似 Windows 3.x 的协作式多任务。另外一种则需要内核的参与，由内核完成线程的调度。其依赖于操作系统核心，由内核的内部需求进行创建和撤销，这两种模型各有其好处和缺点。用户线程不需要额外的内核开支，并且用户态线程的实现方式可以被定制或修改以适应特殊应用的要求，但是当个线

程因 I/O 而处于等待状态时，整个进程就会被调度程序切换为等待状态，其他线程得不到运行的机会；而内核线程则没有各个限制，有利于发挥多处理器的并发优势，但却占用了更多的系统开支。

Windows NT 和 OS/2 支持内核线程。Linux 支持内核级的多线程

13.C++中什么数据分配在栈或堆中，New 分配数据是在近堆还是远堆中？

答：栈：存放局部变量，函数调用参数,函数返回值，函数返回地址。由系统管理

堆：程序运行时动态申请，new 和 malloc 申请的内存就在堆上

14.使用线程是如何防止出现大的波峰。

答：意思是如何防止同时产生大量的线程，方法是使用线程池，线程池具有可以同时提高调度效率和限制资源使用的好处，线程池中的线程达到最大数时，其他线程就会排队等候。

15 函数模板与类模板有什么区别？

答：函数模板的实例化是由编译程序在处理函数调用时自动完成的，而类模板的实例化必须由程序员在程序中显式地指定。

16 一般数据库若出现日志满了，会出现什么情况，是否还能使用？

答：只能执行查询等读操作，不能执行更改，备份等写操作，原因是任何写操作都要记录日志。也就是说基本上处于不能使用的状态。

17 SQL Server 是否支持行级锁，有什么好处？

答：支持，设立封锁机制主要是为了对并发操作进行控制，对干扰进行封锁，保证数据

的一致性和准确性，行级封锁确保在用户取得被更新的行到该行进行更新这段时间内不被其它用户所修改。因而行级锁即可保证数据的一致性又能提高数据操作的进发性。

18 如果数据库满了会出现什么情况，是否还能使用？

答：见 16

19 关于内存对齐的问题以及 `sizeof()` 的输出

答：编译器自动对齐的原因：为了提高程序的性能，数据结构（尤其是栈）应该尽可能地在自然边界上对齐。原因在于，为了访问未对齐的内存，处理器需要作两次内存访问；然而，对齐的内存访问仅需要一次访问。

20 `int i=10, j=10, k=3; k*=i+j; k` 最后的值是？

答：60，此题考察优先级，实际写成：`k*=(i+j);`，赋值运算符优先级最低

21.对数据库的一张表进行操作,同时要对另一张表进行操作,如何实现?

答：将操作多个表的操作放入到事务中进行处理

22.TCP/IP 建立连接的过程?(3-way shake)

答：在 TCP/IP 协议中，TCP 协议提供可靠的连接服务，采用三次握手建立一个连接。

第一次握手：建立连接时，客户端发送 `syn` 包(`syn=j`)到服务器，并进入 `SYN_SEND` 状态

，等待服务器确认；

第二次握手：服务器收到 `syn` 包，必须确认客户的 `SYN` (`ack=j+1`)，同时自己也发送一个

SYN 包 (syn=k) , 即 SYN+ACK 包, 此时服务器进入 SYN_RECV 状态;

第三次握手: 客户端收到服务器的 SYN+ACK 包, 向服务器发送确认包 ACK(ack=k+1)
 , 此包发送完毕, 客户端和服务器进入 ESTABLISHED 状态, 完成三次握手。

23.ICMP 是什么协议,处于哪一层?

答: Internet 控制报文协议, 处于网络层 (IP 层)

24.触发器怎么工作的?

答: 触发器主要是通过事件进行触发而被执行的, 当对某一表进行诸如 UPDATE、INSERT
 、 DELETE 这些操作时, 数据库就会自动执行触发器所定义的 SQL 语句, 从而确保对数
据的处理必须符合由这些 SQL 语句所定义的规则。

25.winsock 建立连接的主要实现步骤?

答: 服务器端: socket()建立套接字, 绑定 (bind) 并监听 (listen) , 用 accept ()
等待客户端连接。

客户端: socket()建立套接字, 连接 (connect) 服务器, 连接上后使用 send()和 recv (
) , 在套接字上写读数据, 直至数据交换完毕, closesocket()关闭套接字。

服务器端: accept () 发现有客户端连接, 建立一个新的套接字, 自身重新开始等待连
接。该新产生的套接字使用 send()和 recv () 写读数据, 直至数据交换完毕, closesock
et()关闭套接字。

26.动态连接库的两种方式?

答: 调用一个 DLL 中的函数有两种方法:

1. 载入时动态链接 (load-time dynamic linking) , 模块非常明确调用某个导出函数

，使得他们就像本地函数一样。这需要链接时链接那些函数所在 DLL 的导入库，导入库向系统提供了载入 DLL 时所需的信息及 DLL 函数定位。

2. 运行时动态链接 (run-time dynamic linking)，运行时可以通过 LoadLibrary 或 LoadLibraryEx 函数载入 DLL。DLL 载入后，模块可以通过调用 GetProcAddress 获取 DLL 函数的

出口地址，然后就可以通过返回的函数指针调用 DLL 函数了。如此即可避免导入库文件了。

。

27.IP 组播有那些好处?

答：Internet 上产生的许多新的应用，特别是高带宽的多媒体应用，带来了带宽的急剧消耗和网络拥挤问题。组播是一种允许一个或多个发送者（组播源）发送单一的数据包到多个接收者（一次的，同时的）的网络技术。组播可以大大的节省网络带宽，因为无论有多少个目标地址，在整个网络的任何一条链路上只传送单一的数据包。所以说组播技术的核心就是针对如何节约网络资源的前提下保证服务质量。

1.static 有什么用途？（请至少说明两种）

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

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，局部变量在堆栈。

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 函数不能声明为虚函数。

8.冒泡排序算法的时间复杂度是什么？

时间复杂度是 $O(n^2)$ 。

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 协议

主要层次结构为：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)

12. IP 地址的编码分为哪两部分？

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。

13. 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

14. 不能做 switch() 的参数类型是：

switch 的参数不能为实型。

華為

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：无限循环，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<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

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?

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

2、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。

3、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：__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 $\text{int}4 + \text{DATE}20 + \text{double}8 = 32$ 。

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 $\text{int}2 + \text{DATE}10 + \text{double}8 =$

4、队列和栈有什么区别？

队列先进先出，栈后进先出

5、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;

}

```

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 <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);
}
```

2.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

3.直接链接两个信令点的一组链路称作什么?

PPP 点到点连接

4.接入网用的是什么接口?

5.VoIP 都用了那些协议?

6.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

7.确定模块的功能和模块的接口是在软件设计的那个阶段完成的?

概要设计阶段

8.enum string

{

x1,

x2,

x3=10,

x4,

x5,

}x;

问 x= 0x801005, 0x8010f4 ;

9.unsigned char *p1;

unsigned long *p2;

p1=(unsigned char *)0x801000;

```
p2=(unsigned long *)0x810000;
```

请问 p1+5= ；

p2+5= ；

三.选择题:

1.Ethternet 链接到 Internet 用到以下那个协议?

A.HDLC;B.ARP;C.UDP;D.TCP;E.ID

2.属于网络层协议的是:

A.TCP;B.IP;C.ICMP;D.X.25

3.Windows 消息调度机制是:

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>>)%256  
  
}
```

请问 hash(16),hash(256)的值分别是:

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

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;
```

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

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;
```

```
}
```

```
.....
```

```
}
```

五.问答题:

1.IP Phone 的原理是什么?

IPV6

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

2. 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 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' ？

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：

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 invert(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 invert(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。

12. 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。

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。

```
const char* s="AAA";
```

然后又因为是常量，所以对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)` 表示的是什么呢

`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 <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

```

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

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

15. 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

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

链表：数据可以随机存储，大小可动态改变

12.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: (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

2: 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？

main 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

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)();
```

2，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]))
```

面试题：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

```
unsigned short A = 10;
```

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

```
char c=128;
```

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

输出多少？并分析过程

第一题， $\sim A = 0xffffffff5$, 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。

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。。。

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。*/
```

```
long a=0x801010;
```

```
a+5=?
```

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 罗

1)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

```

2)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？

```

// 帮忙分析一下

#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

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

$$9999 = 9 \times 1024 + 512 + 256 + 15$$

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。

int a,b,c 请写函数实现 C=a+b ,不可以改变数据类型,如将 c 改为 long int,关键是如何处理溢出问题

```
bool add (int a, int b,int *c)

{

*c=a+b;

return (a>0 && b>0 &&(*c<a || *c<b) || (a<0 && b<0 &&(*c>a || *c>b)));

}
```

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 为有符号数时, 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 为有符号数时, c = 100,即 c = 4,同理 b = 3

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, 用一位二进制位即可。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 位二进制。

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制分配的。

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;
```

```
}
```

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.

```
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。

```

intel:

```

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）？

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。

```

struct sl

{

    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

上面是基本问题，接下来是编程问题：

本人很弱，这几个题也搞不定，特来求救：

1) 读文件 file1.txt 的内容（例如）：

12

34

56

输出到 file2.txt:

56

34

12

(逆序)

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。

望高手赐教！！

第一题,注意可增长数组的应用.

```
#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;
```

```
}
```

第二题.

```
#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);

}

```

兄弟,这样的题目若是做不出来实在是有些不应该, 给你一个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;

}

```

借鉴了楼上的“递归反向输出”

```

#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;
}

```

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`

你可以参考一下指针运算的细节

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;

return low;

}

```

```

void QSort (int*L, int low, int high)

{

if (low < high)

{

int p1 = Partition (L, low, high);

QSort (L, low, p1 - 1);

QSort (L, p1 + 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';

}

```

```

return 0;

}

```

1、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);

}

}

```

2.

```

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

3. 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,

4.

```
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。

5:

```
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

今天早上的面试题 9 道， 比较难， 向牛人请教， 国内的一牛公司， 坐落在北京北四环某大厦：

1、 线形表 a、 b 为两个有序升序的线形表， 编写一程序， 使两个有序线形表合并成一个有序升序线形表 h；

答案在 请化大学严锐敏《数据结构第二版》第二章例题， 数据结构当中， 这个叫做： 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;

}

```

2、运用四色定理，为 N 个局域举行配色，颜色为 1、2、3、4 四种，另有数组 $adj[i][N]$ ，如 $adj[i][j]=1$ 则表示 i 区域与 j 区域相邻，数组 $color[N]$ ，如 $color[i]=1$ ，表示 i 区域的颜色为 1 号颜色。

四色填充

3、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] );

}

```

4、编写算法，从10亿个浮点数当中，选出其中最大的10000个。

用外部排序，在《数据结构》书上有

《计算方法导论》在找到第n大的数的算法上加工

5、编写一unix程序，防止僵尸进程的出现。

同学的4道面试题，应聘的职位是搜索引擎工程师，后两道超级难，（希望大家多给一些算发）

1.给两个数组和他们的大小，还有一动态开辟的内存，求交集，把交集放到动态内存

dongtai，并且返回交集个数

```
long jiaoji(long* a[],long b[],long* alength,long blength,long* dongtai[])
```

2.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.data=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 length = 'z' - 'b';

int i=0;

```

```

while ( i<=length )

{

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);

```

3.可怕的题目终于来了

象搜索的输入信息是一个字符串，统计 300 万输入信息中的最热门的前十条，我们每次输入的一个字符串为不超过 255byte,内存使用只有 1G,

请描述思想，写出算发（c 语言），空间和时间复杂度，

4.国内的一些贴吧，如 baidu,有几十万个主题，假设每一个主题都有上亿的跟帖子，怎么样设计这个系统速度最好，请描述思想，写出算发（c 语言），空间和时间复杂度，

```

#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);

    }

```

1. 简述一个 Linux 驱动程序的主要流程与功能。
2. 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;

}

```

6. 请问一下程序将输出什么结果?

```

char *RetMemory(void)

{

    char p[] = "hellow world" ;

    return p;

}

void Test(void)

{

```

```

char *str = NULL;

str = RetMemory();

printf(str);

}

```

RetMemory 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

写一个函数,它的原形是 `int continumax(char *outputstr,char *inputstr)`

功能:

在字符串中找出连续最长的数字串，并把这个串的长度返回，并把这个最长数字串付给

其中一个函数参数 outputstr 所指内存。例如： "abcd12345ed125ss123456789"的首地址传给

inputstr 后，函数将返回

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;

}
```

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;

}

```

前几天面试，有一题想不明白,请教大家！

```
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);
```

谢谢!

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

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",&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(l==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]);//

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;

}

```

1、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;

}

```

2、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;

}

]]>

```

</Content>

<PostDateTime>2006-4-17 10:17:34</PostDateTime>

</Reply>

<Reply>

<PostUserNickName>白日?做梦!</PostUserNickName>

<rank>一级(初级)</rank>

<ranknum>user1</ranknum>

<credit>100</credit>

<ReplyID>34231324</ReplyID>

<TopicID>4691482</TopicID>

<PostUserId>695883</PostUserId>

<PostUserName>free131</PostUserName>

<Point>0</Point>

<Content>

<![CDATA[

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;
```

```
}
```

试题:

```
void test2()
```

```
{
```

```
    char string[10], str1[10];
```

```
    int i;
```

```
    for(i=0; i<10; i++)
```

```
    {
```

```
        str1[i] = 'a';
```

```
    }
```

```
    strcpy( string, str1 );
```

```
}
```

解答：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；

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},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 );

}
```

第二个 code 题是实现 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 && str2);

    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;

}

```

第三个 code 题是实现子串定位

```

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;

}

```

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

华为全套完整试题

高级题

6、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)
```

```
}
```

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;

}

```

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;

}

```

(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';
```

```
}
```

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,则不同谢谢!

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 }

```

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);

}

```

华为面试题：怎么判断链表中是否有环？

```

bool CircleInList(Link* pHead)

{

    if(pHead == NULL || pHead->next == NULL) //无节点或只有一个节点并且无自环

        return (false);

    if(pHead->next == pHead) //自环

        return (true);

    Link *pTempl = pHead; //step 1

    Link *pTemp = pHead->next; //step 2

    while(pTemp != pTempl && pTemp != NULL && pTemp->next != NULL)

    {

        pTempl = pTempl->next;

        pTemp = pTemp->next->next;
    }
}

```

```

}

if(pTemp == pTemp1)

return (true);

return (false);

}

```

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';

}

```

1.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;

}

```

2。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;

}

```

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？

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。

```
#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 and 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。

2.写一函数，实现删除字符串 str1 中含有的字符串 str2.

第二个就是利用一个 KMP 匹配算法找到 str2 然后删除（用链表实现的话，便捷于数组）

/*雅虎笔试题(字符串操作)

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);

}

```

11.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 );

}

```

3,求 1000! 的末尾有几个 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;

}

```

1. 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 (DeleteNode(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;

}

}

}

```

2. 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;
```

```
}
```

```
}
```

```
}
```

```
}
```

3. 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;

}

```

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");

    }

}

```

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

如果先声明的在低位则:

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

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;

}
```

Josephu 问题为：设编号为 1, 2, \dots n 的 n 个人围坐一圈，约定编号为 k ($1 \leq k \leq n$) 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;
}

```

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`。多谢楼主 辛苦了

正方形 4 个顶点;

每边的中点,共 4 个;

两条对角线的交点, 1 个。

总共 9 个点。请用一笔将 9 个点连起来?