嵌入式面试基础知识总结

目录

[**第一部分 C语言** 3](#_Toc160896935)

[1. 数组作为参数传递给函数 3](#_Toc160896936)

[2. 交换两个数的宏定义 3](#_Toc160896937)

[3. i+++j的结果 3](#_Toc160896938)

[4. 无符号整形和有符号整形进行运算 3](#_Toc160896939)

[5. 自己实现mystrcpy函数 3](#_Toc160896940)

[6. 动态申请内存问题 4](#_Toc160896941)

[7. 关键字const的作用 4](#_Toc160896942)

[8. 关键字static的作用 5](#_Toc160896943)

[9. 关键字volatile的作用 5](#_Toc160896944)

[10. 关于volatile的面试题 6](#_Toc160896945)

[11. 宏定义#define问题 7](#_Toc160896946)

[12. #error的目的 7](#_Toc160896947)

[13. typedef和define的区别 7](#_Toc160896948)

[14. 设置绝对地址为0x67a9的整型变量的值为0xaa66 7](#_Toc160896949)

[15. C语言定义的中断服务子程序 7](#_Toc160896950)

[16. 根据处理器的位数定义最大值 8](#_Toc160896951)

[17. 汇编、C语言和C++在嵌入式开发中的区别和特点 8](#_Toc160896952)

[18. 约瑟夫环问题算法（使用循环链表） 8](#_Toc160896953)

[**第二部分 单片机** 8](#_Toc160896954)

[1. 单片机IO开漏输出和推挽输出有什么区别 8](#_Toc160896955)

[**第三部分 Linux** 9](#_Toc160896956)

[1. 用VI编译器底行命令模式实现字符串替换 9](#_Toc160896957)

# **第一部分 C语言**

## 数组作为参数传递给函数

windows NT下32位程序，请计算sizeof 的值

void func(char str[100])

{

sizeof(str) = ？;

}

函数中数组作为参数时，[]里的数不起作用，传递的是首元素的地址，所以sizeof(str)等于地址的大小， 32位下地址大小为4。以下三种形式是等价的：

void Func(char str[100]);

void Func(char str[]);

void Func(char \*str);

## 交换两个数的宏定义

#define SWAP(a,b) a=a+b;b=a-b;a=a-b; /\* 通用性较高，但有数据溢出风险，不适合结构体 \*/

#define SWAP(a,b) a=a^b;b=a^b;a=a^b; /\* 避免了数据溢出，但只适用于计算整型，不适用于浮点和结构体 \*/

#define SWAP(a,b) {typeof(a) tmp = a;a = b;b = tmp;} /\* typeof()可自动获取类型，但只能在gnu编译器中使用 \*/

## i+++j的结果

int i = 1;

int j = 2;

int k = i +++ j;

printf("k = %d",k);

大多数的C编译系统都是尽可能的将多个字符结合成为一个运算符,所以 i+++j 等价于(i++)+j)，所以k=3。

## 无符号整形和有符号整形进行运算

void foo(void)

{

unsigned int a = 6;

int b = -20;

(a + b > 6) ? puts("> 6") : puts("<= 6");

}

答案输出是“>6”。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。嵌入式系统中频繁用到无符号整形的数据，务必注意。

## 自己实现mystrcpy函数

char \* mystrcpy( char \*strDest, const char \*strSrc ) //3 const修饰带拷贝的字符串，提高代码健壮性

{

assert( (strDest != NULL) && (strSrc != NULL) ); //2 检查指针的有效性，提高代码的健壮性

char \*address = strDest; //1

while( (\*strDest++ = \* strSrc++) != '\0' ); //3

return address; //1 支持链式表达式，提高函数可用性

}

如上，该函数将address返回了，目的是实现链式表达式，举例如下。

char str[20];

int length = [strlen](https://so.csdn.net/so/search?q=strlen&spm=1001.2101.3001.7020)( mystrcpy(str, “Hello World”) );

## 动态申请内存问题

|  |  |  |
| --- | --- | --- |
| **示例1** | **示例2** | **示例3** |
| void GetMemory(char \*p)  {  p = (char \*)malloc(100);  }  void Test(void)  {  char \*str = NULL;  GetMemory(str);  strcpy(str, "hello world");  printf(str);  } | char \*GetMemory(void)  {  char p[] = "hello world";  return p;  }  void Test(void)  {  char \*str = NULL;  str = GetMemory();  printf(str);  } | void GetMemory2(char \*\*p, int num)  {  \*p = (char \*)malloc(num);  }  void Test(void)  {  char \*str = NULL;  GetMemory(&str, 100);  strcpy(str, "hello");  printf(str);  } |

**示例**1：

传入函数GetMemory()的形参为字符串指针，函数内部修改形参并不会真正的改变传入形参的值，所以执行Test()函数后，str仍为NULL。

**示例2**：

函数GetMemory()内创建的p[]数组为局部变量，函数返回后，内存就已将被释放了，所以执行Test()函数后，str指向的内容可能是乱码。

**示例3**：

函数GetMemory()避免了示例1的问题，传入的参数为指针的地址，但在申请完内存后没判断内存是否申请成功，而且也没释放内存。注意在释放内存后一定要将str指向NULL，否则会导致其变成野指针。

**注意**：

1. 在申请内存后应该判断内存是否申请成功。即在p = (char \*)malloc(100);下面加上：if ( p ！= NULL ){ //进行申请内存成功处理 }。
2. 避免内存泄漏和野指针，在printf(str);下面加上：free(str); str = NULL;

## 关键字const的作用

const意味着只读，用来定义常量，如果一个变量被const修饰，那么它的值就不能再被改变。

* 1. **增强程序的健壮性**

与预编译指令#define相比，#define只是对值进行简单的替换，并不会进行类型检查，而const可以保护被修饰的

东西，防止意外修改，增强程序的健壮性。

* 1. **提高编译效率**

编译器通常不为普通const常量分配存储空间，而是将它们保存在符号表中，这使得它成为一个编译期间的常量，

没有了存储与读内存的操作，使得它的效率也很高。

* 1. **const的用法如下：**
  2. 修饰局部变量

const int n=5和int const n=5这两种写法一样，表示变量n的值不能被改变了，也可以修饰常量静态字符串，如const char\* str="fdsafdsa"，后续若对str进行了误修改，编译期间就会发现，避免程序的逻辑错误。

* 1. 修饰常量指针与指针常量
     1. 修饰常量指针表示不能使用这个指针改变变量的值，如：int a = 100; const int \* n = &a或int const \* n = &a，那么就不能使用\*n = 10将a的值改为10,但指针可以指向其他地址。

修饰函数的参数时也同理，如void StringCopy(char \*strDestination, const char \*strSource)，则在函数中就不能使用\*strSource改变指针strSource指向的内容，否则会报错。

如果函数的返回值是指针形式的，用const修饰后，则函数返回值（指针）指向的地址不能被修改，且只能赋值给由const修饰的同类型指针，如：const char \* GetString(void)，则只能使用const char \*str = GetString()，而不能使用char \*str = GetString()。（我用VS2022使用char \*str = GetString()没发现报错）

* + 1. 修饰指针常量表示指针指向的地址不能变，但地址中保存的数据是可以变的，如：int a = 100; int\* const c = &a; \*c = 10，即c只能指向变量a的地址，但可通过\*c改变a的值。修饰函数的参数时也同理，如void swap ( int \* const p1 , int \* const p2 )，则指针p1和p2指向的地址都不能修改，但是可以使用\*p1和\*p2对值进行修改。
    2. 修饰指向常量的常指针表示指针指向的地址不能改变，且不能通过这个指针改变变量的值，但是依然可以通过其他指针改变这个变量的值，如：const int\* const p。
  1. 修饰全局变量

全局变量的作用域是整个文件，我们应该尽量避免使用全局变量，因为一旦有一个函数改变了全局变量的

值，它也会影响到其他引用这个变量的函数，导致出了bug后很难发现，如果一定要用全局变量，我们应该尽量使用const进行修饰，以防止不必要的人为修改，使用的方法与局部变量是相同的。

## 关键字static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其值维持不变。静态变量的生命周期和程序相同，在main函数之前初始化，在程序退出时销毁。
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。即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。

## 关键字volatile的作用

volatile是一个类型修饰符，用来修饰被不同线程访问和修改的变量。volatile提醒编译器它后面所定义的变量随时都有可能改变，因此编译后的程序每次需要存储或读取这个变量的时候，告诉编译器对该变量不做优化，而是直接从变量内存地址中读取数据，从而可以提供对特殊地址的稳定访问。如果没有volatile关键字，则编译器可能优化读取和存储，可能暂时使用寄存器中的值，如果这个变量由别的程序更新了的话，将出现不一致的现象。简洁地说，volatile关键词影响编译器编译的结果，用volatile声明的变量表示该变量随时可能发生变化，与该变量有关的运算，不要进行编译优化，以免出错。

用法：

1. **并行设备的硬件寄存器**（如状态寄存器）

存储器映射的硬件寄存器通常要加volatile，因为每次对他读写都可能有不同的意义，示例如下。

如对一个设备进行初始化，此设备的某个寄存器为0xff800000。

int\* output = (unsigned int\*)0xff800000; //定义一个IO端口；

int init(void)

{

int i;

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

\*output = i;

}

如上所示，编译器优化后代码相当于：

int init(void)

{

int i;

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

\*output = i;

}

如果对设备进行初始化的过程必须向最上面的代码一样对寄存器顺序赋值，显然编译器优化后对此地址的读操作只做了一次，然而从代码角度看是没有任何问题的，并不能达到预期目的，所以可使用volatile通知编译器这个变量是一个不稳定的，在遇到此变量时候不要优化。

1. 中断服务程序中修改的供其他程序检测的变量

中断服务函数中对某变量进行修改时，若主程序中没有修改该变量，则编译器优化后可能只从内存中读取到寄存器中一

次，而后每次只从寄存器中读取变量的副本，导致中断服务程序的操作短路，所以需要使用volatile对变量进行修饰，告诉编译器不对其优化。

1. 多任务环境下个任务间共享标志，应该加volatile

本次线程内读取一个变量时，编译器优化后会把变量读取到寄存器中，之后再取变量值时都是从寄存器读取，当内存变

量或寄存器变量因别的线程而改变了值时，该寄存器的值不会改变，会导致应用程序读取的值与实际的变量值不一致。

## 关于volatile的面试题

1. **一个参数既可以是const还可以是volatile吗？**

可以，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。

1. **一个指针可以是volatile吗？**

可以，当一个中服务子程序修改一个指向buffer的指针时。

1. **下面的函数有什么错误？**

int square(volatile int\*ptr)  
{

return\*ptr \* \*ptr;  
}

该程序的目的是返回指针ptr指向的值的平方，但ptr指向的是volatile型的变量，所以编译器将产生类似于下述代码：

int square(volatile int \*ptr)

{

int a,b;

a = \*ptr;

b = \*ptr;

return a \* b;

}

由于\*ptr的值可能会被意想不到的改变，所以a和b的值可能不同，导致结果与预期不一致，正确的代码如下：

long square(volatile int \*ptr)

{

int a;

a = \*ptr;

return a \* a;

}

## 宏定义#define问题

用预处理指令#define声明一个常数，用以表明1年中有多少秒。

#define SECONDS\_PER\_YEAR (60 \* 60 \* 24 \* 365)UL

1. #define基础知识：不能以分好结束；括号的使用等。
2. 清晰的表达计算的方法，而不是直接使用计算后的结果。
3. 意识到这个表达式将导致一个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，UL表示无符号长整型。

## #error的目的

生成编译错误提示信息，并停止编译。编译程序时，只要遇到#error就会跳出一个错误，目的就是保证程序按照自己所设想的那样进行编译。

## typedef和define的区别

typedef和define都是替一个对象取一个别名，以此增强程序的可读性，区别如下：

* 1. 原理不同

#define是预处理指令，预处理时只进行简单而机械的字符串替换，不做正确性检查；typedef是关键字，在编译处理时会做正确性检查，它在自己的作用域内给一个已经存在的类型一个别名，但不能在函数定义里面使用typedef。用typedef定义数组、指针、结构等类型会带来很大的方便，不仅使程序书写简单，也使意义明确，增强可读性。

* 1. 功能不同

typedef用来定义类型的别名，起到类型易于记忆的功能。另一个功能是定义与机器无关的类型，如定义目标机器

上最高精度的浮点类型REAL：在支持long double的机器上：typedef long double REAL；在不支持long double的机器上：typedef double REAL；在不支持double的机器上：typedef float REAL。#define不只是可以为类型取别名，还可以定义常量、变量、编译开关等。

* 1. 作用域不同

#define没有作用域的限制，只要是之前预定义过的宏，在以后的程序中都可以使用，而typedef有自己的作用域。

* 1. 对指针的操作不同

#define INTPTR1 int\*

typedef int\* INTPTR2;

INTPTR1 p1,p2; //声明一个指针变量p1和一个整型变量p2

INTPTR2 p3,p4; //声明两个指针变量p3、p4

int a = 1;

int b = 2;

int c = 3;

const INTPTR1 p1 = &a; //定义一个常量指针p1，即不能通过p1改变其指向的内容，但p1可以指向其他位置

const INTPTR2 p2 = &b; //定义一个指针常量p2，即p2不能再指向其他位置，

INTPTR2 const p3 = &c; //定义一个指针常量p3

## 设置绝对地址为0x67a9的整型变量的值为0xaa66

int\* ptr = NULL;

ptr = (int\*) 0x67a9;

\*ptr = 0xaa66;

## C语言定义的中断服务子程序

中断在嵌入式系统中很重要，导致很多开发厂商提供一种扩展，让C语言支持中断，使用新的关键字\_\_interrupt。找出

下面中断服务子程序（ISR）的错误。

\_\_interrupt double compute\_area (double radius)

{

double area = PI \* radius \* radius;

printf(" Area = %f", area);

return area;

}

1. ISR不能有返回值，也不能传递参数。
2. ISR应该是短而精悍的，所以不推荐在ISR内进行浮点运算，且在许多处理器编译器中浮点一般都是不可重入的。有些编译器需要额外的寄存器入栈，有些编译器就不允许在ISR中进行浮点运算。
3. printf()函数会导致重入和性能问题，不能再ISR中使用。printf()函数采用的缓冲机制，这个缓冲区是共享的，相当于一个全局变量，第一层中断来时，它向缓冲里面写入一些部分内容，恰好这时来了个优先级更高的中断，它同样调用了printf，也向缓冲里面写入一些内容，这样缓冲区的内容就错乱了。

## 根据处理器的位数定义最大值

unsigned int zero = 0;

unsigned int compzero = 0xFFFF;

如上所示，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。应编写为unsigned int compzero = ~0，对于32位的处理器可以写成unsigned int compzero = 0xFFFFFFFF，但unsigned int compzero = ~0;更安全，不管有多少位，直接取反，把所有的0都变成1了。

## 汇编、C语言和C++在嵌入式开发中的区别和特点

越低级的语言，执行效率越高，越高级的语言，开发周期越短。比如启动代码必须用汇编编写，为C建立好堆栈，分配好内存后使用C进行开发就可以了，如果C语言的结构体满足不了一些功能，比如窗口事件等操作，自然会用到C++。

## 约瑟夫环问题算法（使用循环链表）

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程序法。具体代码见

c语言代码例程.txt第56个示例。

# **第二部分 单片机**

## 单片机IO开漏输出和推挽输出有什么区别

推挽输出既可以输出高电平，也可以输出低电平，在两种电平下都具有驱动能力；开漏输出只可以输出低电平，输出高电平时需要外接上拉电阻。

# **第三部分 Linux**

## 用VI编译器底行命令模式实现字符串替换

把当前文件中的“xiaodai”全部替换成“banzhang”替换： :%s/xiaodai/banzhang/g

替换指令格式：

s/旧词/新词/ 替换一个

s/旧词/新词/g 替换一行

1,3s/旧词/新词/g 替换第一到第三行

%s/旧词/新词/g 替换所有