# 面试学习笔记

## 写一个“标准”宏

这个宏输入两个参数并返回较小的一个。

答案：

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

## 无限循环

10、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。

答案：while(1){}或者for(;;){}

## 关键字static

答案：定义静态变量

这里 static 作用主要影响着变量或函数的生命周期，作用域，以及存储位置。

当 static 修饰局部变量时：

 变量的存储区域由栈变为静态常量区。

 变量的生命周期由局部变为全局。

 变量的作用域不变。

而在static修饰函数局部变量的时，其修饰的静态局部变量只执行初始化一次，延长了局部变量的生命周期，直到程序运行结束以后才释放，但不改变作用域。

没有static时，函数每调用一次， 变量就会进行一次初始化值为 0，

当有static修饰时，初始化语句只会被执行一次所以值会一直累加。

作用域：作用域为局部作用域，当定义它的函数或者语句块结束的时候，作用域结束。但是当局部静态变量离开作用域后，并没有销毁，而是仍然驻留在内存当中，只不过不能再对它进行访问，直到该函数再次被调用 才可以访问，并且值不变。

静态函数

在函数返回类型前 加static，函数就定义为静态函数。函数的定义和声明在默认情况下都是extern的，但静态函数只在声明它的文件当中可见，不能被其他文件所用。

函数的实现使用static修饰，那么这个函数只可在本cpp内使用，不会同其他cpp中的同名函数引起冲突。

warning：不要在头文件中声明static的全局函数；不要在cpp内声明非static的全局函数，如果你要在多个cpp中复用该函数，就把它的声明提到头文件里去，否则cpp内部声明需加上static修饰。如果想在不同原文件里，使用全局变量，可以用extern。

类的静态成员

在类中，静态成员可以实现多个对象之间的数据共享，并且使用静态数据成员还不会破坏隐藏的原则，即保证了安全性。因此，静态成员是类的所有对象中共享的成员，而不是某个对象的成员。对多个对象来说，静态数据成员只存储一处，供所有对象共用。

静态成员变量属于整个类所有

-静态成员变量的生命期不依赖于任何对象，为程序的生命周期

-可以通过类名直接访问公有静态成员变量

-所有对象共享类的静态成员变量

-可以通过对象名访问公有静态成员变量

-静态成员变量需要在类外单独分配空间

-静态成员变量在程序内部位于全局数据区 (Type className::VarName = value)

**类的静态成员和非静态成员有何区别？**

答案：类的静态成员每个类只有一个，非静态成员每个对象一个

但是C++中讲究的是封装性，以上代码，有2个不妥之处

1、类名或对象名能直接访问成员变量，也就是说成员变量能直接被外界修改

2、我们使用了一个成员函数来获取当前的对象个数，看似没问题，但是必须要定义对象，通过对象去调用，但有时候我们不想定义对象，也能使用类中的成员函数，这就是我们要说的类的静态成员函数。

类的静态函数

静态成员函数和静态数据成员一样，它们都属于类的静态成员，它们都不是对象成员。因此，对静态成员的引用不需要用对象名。

在静态成员函数的实现中，不能直接引用类中说明的非静态成员，可以引用类中说明的静态成员（这点非常重要）。

如果静态成员函数中要引用非静态成员，可通过对象来引用。

从中可看出，调用静态成员函数使用如下格式：<类名>::<静态成员函数名>(<参数表>);

## 关键字const

const名叫常量限定符

const修饰一般常量及数组

int const a = 100;

const int a = 100; //与上面等价

int const arr [3] = {1,2,3};

const int arr [3] = {1,2,3};//与上面等价

输入参数为指针，加上const之后就会起保护指针意外修改的作用

可以起到保护作用的原因是：

实参中，指针会指向一段内存地址，调用函数之后，函数会产生一个临时指针变量，这个变量的地址与实参的地址不一样，但是这两个指针指向的内存是同一块。形参加上const 修饰之后，保护了这一块内存地址不被修改，如果刻意修改这一块内存，编译器会报错。

const修饰成员函数

const 修饰的成员函数为了保护成员变量，要求const 函数不能修改成员变量，否则编译会报错。

函数体内不能修改成员变量的值，增加程序的健壮性或鲁棒性。只有成员函数才可以在后面加const，普通函数后加const无意义。

const函数的规则

const 对象只能访问const 成员函数，非const 的对象可以访问任何成员函数，包括const 成员函数。

如果函数名、参数、返回值都相同的const成员函数和非const成员函数是可以构成重载，那么const对象调用const成员函数，非const对象默认调用非const的成员函数。

const成员函数可以访问所有成员变量，但是只能访问const的成员函数。

非const成员函数，可以访问任何成员，包括const成员成员函数。

const成员函数不能修改任何的成员变量，除非变量用mutable修饰。

类中实现常量，使用枚举类型

C++11仅不允许在类声明中初始化static非const类型的数据成员。

类中的const变量只能通过构造函数初始化列表进行初始化！

## 关键字volatile

[volatile](https://so.csdn.net/so/search?q=volatile&spm=1001.2101.3001.7020" \o "volatile)提醒编译器它后面所定义的变量随时都有可能改变

编译后的程序每次需要存储或读取这个变量的时候，都会直接从变量地址中读取数据。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。下面举例说明。

在DSP开发中，经常需要等待某个事件的触发，所以经常会写出这样的程序：

一般说来，volatile用在如下的几个地方：

1、中断服务程序中修改的供其它程序检测的变量需要加volatile；

2、多任务环境下各任务间共享的标志应该加volatile；

3、存储器映射的硬件寄存器通常也要加volatile说明，因为每次对它的读写都可能由不同意义；

## int (\*s[10])(int) 表示的是什么啊

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

## 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;

答案：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;

## c和c++中的struct有什么不同？

答案：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

## malloc

#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;

}

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

函数形参传递的是数值，而且是无法返回动态分配的内存。

即函数生成了个值和p相同的char\*临时变量,为这个临时指针分配了空间,而p并没有改变.  所以,可以把p改成引用传递,char \*&p.

还有一种思路就是：传指针的地址值

如上例，在调用子函数前，取char \*p地址值，即

addr = (int)p;

函数的第一个形参改为int addr

函数内部先将addr转为指针，即

ptr = (char \*)addr

这样同样可以实现子函数内部动态申请内存

两种思路本质上是一样的，都是传址而非传值；第二种思路其实是人手完成了引用要做的事情

关于C++中指针的引用\*&

指针的引用，相当于传递的是：指针的指针，这样指针的数值是可以改变的

而单传递指针，不传递指针的引用，那么指针指向的数据是可以改变，而指针本身是不可以改变的

void InitStack(LNode\*& HS)

{

HS = NULL; //函数返回后HS就是NULL了

}

void InitStack(LNode\* HS)

{

HS = NULL; //函数返回后，HS依然是传递进来的数值

}

简单讲，\*&指针本身可变；\*指针本身不变，仅指向的内容可变。

## 进程的同步机制

列举几种进程的同步机制，并比较其优缺点。

答案： 原子操作 信号量机制 自旋锁 管程，会合，分布式系统

### 信号量机制

一个信号量只能置一次初值，以后只能对之进行p操作或v操作。由此也可以看到，信号量机制必须有公共内存，不能用于分布式操作系统，这是它最大的弱点。

（1）信号量（Semaphore）及PV操作

 优：PV操作能够实现对临区的管理要求；实现简单；允许使用它的代码休眠，持有锁的时间可相对较长。

缺：信号量机制必须有公共内存，不能用于分布式操作系统，这是它最大的弱点。信号量机制功能强大，但使用时对信号量的操作分散，而且难以控制，读写和维护都很困难。加重了程序员的编码负担；核心操作P-V分散在各用户程序的代码中，不易控制和管理；一旦错误，后果严重，且不易发现和纠正。

### 自旋锁

自旋锁是为了保护共享资源提出的一种锁机制。调用者申请的资源如果被占用，即自旋锁被已经被别的执行单元保持，则调用者一直循环在那里看是否该自旋锁的保持着已经释放了锁，自旋锁是一种比较低级的保护数据结构和代码片段的原始方式，可能会引起以下两个问题;

（1）死锁

（2）过多地占用CPU资源

优：旋锁是为了保护共享资源提出的一种锁机制；调用者申请的资源如果被占用，即自旋锁已经被别的执行单元保持，则调用者一直循环在那里看是否该自旋锁的保持者已经释放了锁；低开销；安全和高效；

缺：自旋锁是一种比较低级的保护数据结构和代码片段的原始方式，可能会引起以下两个问题;

（1）死锁

（2）过多地占用CPU资源

传统自旋锁由于无序竞争会导致“公平性”问题

### 管程

信号量机制功能强大，但使用时对信号量的操作分散，而且难以控制，读写和维护都很困难。因此后来又提出了一种集中式同步进程——管程。其基本思想是将共享变量和对它们的操作集中在一个模块中，操作系统或并发程序就由这样的模块构成。这样模块之间联系清晰，便于维护和修改，易于保证正确性。

优：集中式同步进程——管程。其基本思想是将共享变量和对它们的操作集中在一个模块中，操作系统或并发程序就由这样的模块构成。这样模块之间联系清晰，便于维护和修改，易于保证正确性。

缺：如果一个分布式系统具有多个CPU，并且每个CPU拥有自己的私有内存，它们通过一个局域网相连，那么这些原语将失效。而管程在少数几种编程语言之外又无法使用，并且，这些原语均未提供机器间的信息交换方法。