```
1.main()
 {

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

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

    printf("%d,%d",*(a+1),*(ptr-1));

}
```

答：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].

2. 以下为 Windows NT 下的 32 位 C++程序，请计算 sizeof 的值

```
void Func ( char str[100] )
{
   sizeof( str ) = ?
}
void *p = malloc( 100 );
sizeof ( p ) = ?
```

这题 很常见了,Func (char str[100])函数中数组名作为函数形参时，在函数体内，数组名失去了本身的内涵，仅仅只是一个指针；在失去其内涵的同时，它还失去了其常量特性，可以作自增、自减等 操作，可以被修改。Windows NT 32 位平台下，指针的长度（占用内存的大小）为 4 字节，故 sizeof(str) 、sizeof (p) 都为 4。

3.还是考指针,不过我对 cocoa 的代码还是不太熟悉

大概是这样的

```
- （void）*getNSString(const NSString * inputString)

{

        inputString = @"This is a main test\n";

        return ;

}

-main(void)

{



NSString *a=@"Main";



  NSString *aString = [NSString stringWithString:@"%@",getNSString(a)];
```

```
  NSLog(@"%@\n", aString);

}
```

最后问输出的字符串:NULL,output 在 函数返回后，内存已经被释放。

4.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：
?; #define 语法的基本知识（例如：不能以分号结束，括号的使用，等等）
?; 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。

?; 意识到这个表达式将使一个 16 位机的整型数溢出-因此要用到长整型符号 L,告诉编译器这个常数是的长整型数。
?; 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。

写一个"标准"宏 MIN，这个宏输入两个参数并返回较小的一个。

```
#defineMIN(A,B) ((A) <= (B) ? (A) :(B))
```

这个测试是为下面的目的而设的： ?;标识#define 在宏中应用的基本知识。这是很重要的，因为直到嵌入(inline)操作符变为标准 C 的一部分，宏是方便产生嵌入代码的唯一方法，对于嵌入式系统来说，为了能达到要求的性能，嵌入代码经常是必须的方法。 ?;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。 ?; 懂得在宏中小心地把参数用括号括起来 ?; 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);

结果是：
((*p++) <= (b) ? (*p++) : (*p++))
这个表达式会产生副作用，指针 p 会作三次++自增操作。

5.写一个委托的 interface

```
@protocol MyDelegate;

@interface MyClass: NSObject
{
    id <MyDelegate> delegate;
}

// 委托方法
@protocol MyDelegate
- (void)didJobs:(NSArray *)args;
@end
```

6. 写一个 NSString 类的实现

```
+ (id)initWithCString:(const char *)nullTerminatedCString encoding:(NSStringEncoding)encoding;

+ (id) stringWithCString: (const char*)nullTerminatedCString
                encoding: (NSStringEncoding)encoding
{
```

```
    NSString    *obj;

    obj = [self allocWithZone: NSDefaultMallocZone()];
    obj = [obj initWithCString: nullTerminatedCString encoding: encoding];
    return AUTORELEASE(obj);
}
```

7.obj-c 有多重继承么?不是的话有什么替代方法?

cocoa 中所有的类都是 NSObject 的子类

多继承在这里是用 protocol 委托代理 来实现的
你不用去考虑繁琐的多继承 ,虚基类的概念.
ood 的多态特性 在 obj-c 中通过委托来实现.

8.obj-c 有私有方法么?私有变量呢

objective-c - 类里面的方法只有两种, 静态方法和实例方法. 这似乎就不是完整的面向对象了,按照 OO 的原则就是一个对象只暴露有用的东西. 如果没有了私有方法的话, 对于一些小范围的代码重用就不那么顺手了. 在类里面声名一个私有方法

```
@interface Controller : NSObject { NSString *something; }

+ (void)thisIsAStaticMethod;

- (void)thisIsAnInstanceMethod;

@end

@interface Controller (private) -

(void)thisIsAPrivateMethod;

@end
```

@private 可以用来修饰私有变量

在 Objective - C 中，所有实例变量默认都是私有的，所有实例方法默认都 是公有的

9.关键字 const 有什么含意？修饰类呢?static 的作 用,用于类呢?还有 extern c 的作用

const 意味着"只读"，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）。

结论：

?; 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。）
?; 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。
?; 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。

（1）欲阻止一个变量被改变，可以使用 const 关键字。在定义该 const 变量时，通常需要对它进行初
始化，因为以后就没有机会再去改变它了；
（2）对指针来说，可以指定指针本身为 const，也可以指定指针所指的数据为 const，或二者同时指

定为 const；
（3）在一个函数声明中，const 可以修饰形参，表明它是一个输入参数，在函数内部不能改变其值；
（4）对于类的成员函数，若指定其为 const 类型，则表明其是一个常函数，不能修改类的成员变量；
（5）对于类的成员函数，有时候必须指定其返回值为 const 类型，以使得其返回值不为“左值”。

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：

?; 并行设备的硬件寄存器（如：状态寄存器）
?; 一个中断服务子程序中会访问到的非自动变量(Non-automatic variables)
?; 多线程应用中被几个任务共享的变量

?; 一个参数既可以是 const 还可以是 volatile 吗？解释为什么。
?; 一个指针可以是 volatile 吗？解释为什么。

下 面是答案：
?; 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。
?; 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。

static 关键字的作用：

（1）函数体内 static 变量的作用范围为该函数体，不同于 auto 变量，该变量的内存只被分配一次，
因此其值在下次调用时仍维持上次的值；

（2）在模块内的 static 全局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；
（3）在模块内的 static 函数只可被这一模块内的其它函数调用，这个函数的使用范围被限制在声明
它的模块内；
（4）在类中的 static 成员变量属于整个类所拥有，对类的所有对象只有一份拷贝；
（5）在类中的 static 成员函数属于整个类所拥有，这个函数不接收 this 指针，因而只能访问类的 static 成员变量。

extern "C" 的作用

（1）被 extern "C"限定的函数或变量是 extern 类型的；

extern 是 C/C++语言中表明函数和全局变量作用范围（可见性）的关键字，该关键字告诉编译器，
其声明的函数和变量可以在本模块或 其它模块中使用。

（2）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；

extern "C"的惯用法

（1）在 C++中引用 C 语言中的函数和变量，在包含 C 语言头文件（假设为 cExample.h）时，需进
行下列处理：

```
extern "C"
{
#include "cExample.h"
}
```

而在 C 语言的头文件中，对其外部函数只能指定为 extern 类型，C 语言中不支持 extern "C"声明，
在.c 文件中包含了 extern "C"时会出现编译语法错误。

（2）在 C 中引用 C++语言中的函数和变量时，C++的头文件需添加 extern "C"，但是在 C 语言中不
能直接引用声明了 extern "C"的该头文件，应该仅将 C 文件中将 C++中定义的 extern "C"函数声明为
extern 类型。

10.为什么标准头文件都有类似以下的结构？

```
#ifndef __INCvxWorksh
```

```
#define __INCvxWorksh
#ifdef __cplusplus
extern "C" {
#endif

#ifdef __cplusplus
}
#endif
#endif
```

显然，头文件中的编译宏“#ifndef __INCvxWorksh、#define __INCvxWorksh、#endif” 的作用
是防止该头文件被重复引用。

11.#import 跟#include 的区别,@class 呢?

@class 一般用于头文件中需要声明该类的某个实例变量的时候用到，在 m 文 件中还是需要使用#import

而#import 比起#include 的好处就是不会引起交叉编译

12.MVC 模式的理解

MVC 设计模式考虑三种对象：模型对象、视图对象、和控制器对象。模型对象代表 特别的知识和专业技能，它们负责保有应用程序的数据和定义操作数据的逻辑。视图对象知道如何显示应用程序的模型数据，而且可能允许用户对其进行编辑。控制 器对象是应用程序的视图对象和模型对象之间的协调者。

13.线程与进程的区别和联系?

进程和线程都是由操作系统所体会的程序运行的基本 单元，系统利用该基本单元实现系统对应用的并发性。

程和线程的主要差别在于它们是不同的操作系统资源 管理方式。进程有独立的地址空间，一个进程崩溃后，在保护模式下不会对其它进程产生影响，而线程只是一个进程中的不同执行路径。线程有自己的堆栈和局部变 量，但线程之间没有单独的地址空间，一个线程死掉就等于整个进程死掉，所以多进程的程序要比多线程的程序健壮，但在进程切换时，耗费资

源较大，效率要差一 些。但对于一些要求同时进行并且又要共享某些变量的并发操作，只能用线程，不能用进程。

14.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
进程之间通信的途径

答案：共享存储系统消息传递系统管道：以文件系统为基础
进 程死锁的原因

答案：资源竞争及进程推进顺序非法
死锁的 4 个必要条 件

答案：互斥、请求保持、不可剥夺、环路
死锁的处理

答案：鸵鸟策略、预防策略、避免策略、检测与解除死锁

15.堆和栈的区别

管理方式：对于栈来讲，是由编译器自动管理，无需我们手工控制；对于堆来说，释放工作由程序员控制，容易产生 memory leak。

申请大小：
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。

碎片问题：对于堆来讲，频繁的 new/delete 势必会造成内存空间的不连续，从而造成大量的碎片，使程序效率降低。对于栈来讲，则不会存在这个 问题，因为栈是先进后出的队列，他们是如此的一一对应，以至于永远都不可能有一个内存块从栈中间弹出

分配方式：堆都是动态分配的，没有静态分配的堆。栈有 2 种分配方式：静态分配和动态分配。静态分配是编译器完成的，比如局部变量的分配。动态分配由 alloca 函数进行分配，但是栈的动态分配和堆是不同的，他的动态分配是由编译器进行释放，无需我们手工实现。

分配效率：栈是机器系统提供的数据结构，计算机会在底层对栈提供支持：分配专门的寄存器存放栈的地址，压栈出栈都有专门的指令执行，这就决定了栈的 效率比较高。堆则是 C/C++函数库提供的，它的机制是很复杂的。

16.什么是键-值,键路径是什么

模型的性质是通过一个简单的键（通常是个字符串）来指定的。视图和控制器通过键 来查找相应的属性值。在一个给定的实体中，同一个属性的所有值具有相同的数据类型。键-值编码技术用于进行这样的查找—它是一种间接访问对象属性的机制。

键路径是一个由用点作分隔符的键组成的字符串，用于指定一个连接在一起的对象性 质序列。第一个键的
性质是由先前的性质决定的，接下来每个键的值也是相对于其前面的性质。键路径使您可以以独立于模型
实现的方式指定相关 对象的性质。通过键路径，您可以指定对象图中的一个任意深度的路径，使其指向相
关对象的特定属性。

For example, the key path address.streetwould get the value of the address property from the receiving

object, and then determine the street property relative to the address object.

17.c 和 obj-c 如何混用

1）obj-c 的编译器处理后缀为 m 的文件时，可以识别 obj-c 和 c 的代码， 处理 mm 文件可以识别 obj-c,c,c++代码，但 cpp 文件必须只能用 c/c++代码，而且 cpp 文件 include 的头文件中，也不能出现 obj- c 的代码，因为 cpp 只是 cpp
2) 在 mm 文件中混用 cpp 直接使用即可，所以 obj-c 混 cpp 不是问题
3）在 cpp 中混用 obj- c 其实就是使用 obj-c 编写的模块是我们想要的。
如果模块以类实现，那么要按照 cpp class 的标准写类的定义，头文件中不能出现 obj-c 的东西，包括#import cocoa 的。实现文件中，即类的实现代码中可以使用 obj-c 的东西，可以 import,

只是后缀是 mm。
如果模块以函数实现，那么头文件要按 c 的格式声明函数，实现文件中，c++函数内部可以用 obj-c，但后缀还是 mm 或 m。

总结：只要 cpp 文件和 cpp include 的文件中不包含 obj-c 的东西就可以用了，cpp 混用 obj-c 的关键是使用接口，而不能直接使用实现代码，实际上 cpp 混用的是 obj-c 编译后的 o 文件，这个东西其实是无差别的，所以可以用。obj-c 的编译器支持 cpp.

18.目标-动作机制

目标是动作消息的接收者。一个控件，或者更为常见的是它的单元，以插座变量（参 见"插座变量"部分）
的形式保有其动作消息的目标。

动作是控件发送给目标的消息，或者从目标的角度看，它是目标为了响应动作而实现 的方法。

程序需要某些机制来进行事件和指令的翻译。这个机制就是目标-动作机制。

19.cocoa touch 框架

iPhone OS 应用程序的基础 Cocoa Touch 框架重用了许多 Mac 系统的成熟模式，但是它更多地专注于触摸的接口和优化。UIKit 为您提供了在 iPhone OS 上实现图形，事件驱动程序的基本工具，其建立在和 Mac OS X 中一样的 Foundation 框架上，包括文件处理，网络，字符串操作等。

Cocoa Touch 具有和 iPhone 用户接口一致的特殊设计。有了 UIKit，您可以使用 iPhone OS 上的独特的图形接口控件，按钮，以及全屏视图的功能，您还可以使用加速仪和多点触摸手势来控制您的应用。

各色俱全的框架 除了 UIKit 外，Cocoa Touch 包含了创建世界一流 iPhone 应用程序需要的所有框架，从三维图形，到专业音效，甚至提供设备访问 API 以控制摄像头，或通过 GPS 获知当前位置。Cocoa Touch 既包含只需要几行代码就可以完成全部任务的强大的 Objective-C 框架，也在需要时提供基础的 C 语言 API 来直接访问系统。这些框架包括：

Core Animation

通过 Core Animation，您就可以通过一个基于组合独立图层的简单的编程模型来创建丰富的用户体验。

Core Audio

Core Audio 是播放，处理和录制音频的专业技术，能够轻松为您的应用程序添加强大的音频功能。

Core Data

提供了一个面向对象的数据管理解决方案，它易于使用和理解，甚至可处理任何应用 或大或小的数据模型。

功能列表：框架分类
下面是 Cocoa Touch 中一小部分可用的框架：

音频和视频
Core Audio
OpenAL
Media Library
AV Foundation
数据管理
Core Data
SQLite
图形和动画
Core Animation
OpenGL ES
Quartz 2D
网络/li>
Bonjour
WebKit
BSD Sockets
用户应用
Address Book
Core Location
Map Kit
Store Kit

20.objc 的内存管理

? 如果您通过分配和初始化（比如[[MyClass alloc] init]）的方式来创建对象，您就拥有这个对象，需要负责该对象的释放。这个规则在使用 NSObject 的便利方法 new 时也同样适用。
? 如果您拷贝一个对象，您也拥有拷贝得到的对象，需要负责该对象的释放。
? 如果您保持一个对象，您就部分拥有这个对象，需要在不再使用时释放该对象。
反过来，
? 如果您从其它对象那里接收到一个对象，则您不拥有该对象，也不应该释放它（这个规则有少数的例外，在参考文档中有显式的说明）。

21.自动释放池是什么,如何工作

当您向一个对象发送一个 autorelease 消息时，Cocoa 就会将该对 象的一个引用放入到最新的自动释放池。它仍然是个正当的对象，因此自动释放池定义的作用域内的其它对象可以向它发送消息。当程序执行到作用域结束的位置 时，自动释放池就会被释放，池中的所有对象也就被释放。

1. ojc-c 是通过一种"referring counting"(引用计数)的方式来管理内存的, 对象在开始分配内存(alloc)的时候引用计数为一,以后每当碰到有 copy,retain 的时候引用计数都会加一, 每当碰到 release 和 autorelease 的时候引用计数就会减一,如果此对象的计数变为了 0, 就会被系统销毁.
2. NSAutoreleasePool 就是用来做引用计数的管理工作的,这个东西一般不用你管的.
3. autorelease 和 release 没什么区别,只是引用计数减一的时机不同而已,autorelease 会在对象的使用真正结束的时候才做引用计数 减一.

22.类工厂方法是什么

类工厂方法的实现是为了向客户提供方便，它们将分配和初始化合在一个步骤中， 返回被创建的对象，并进行自动释放处理。这些方法的形式是+ (type)className...（其中 className 不包括任何前缀）。

工厂方法可能不仅仅为了方便使用。它们不但可以将分配和初始化合在一起，还可以 为初始化过程提供对象的分配信息。

类工厂方法的另一个目的是使类（比如 NSWorkspace）提供单件实例。虽 然 init...方法可以确认一个类在每次程序运行过程只存在一个实例，但它需要首先分配一个“生的”实例，然后还必须释放该实例。

工厂 方法则可以避免为可能没有用的对象盲目分配内存。

23.单件实例是什么

Foundation 和 Application Kit 框架中的一些类只允许创建单件对象，即这些类在当前进程中的唯一实例。举例来说，NSFileManager 和 NSWorkspace 类在使用时都是基于进程进行单件对象的实例化。当向这些类请求实例的时候，它们会向您传递单一实例的一个引用，如果该实例还不存在，则首先进行实例的分配 和初始化。 单件对象充当控制中心的角色，负责指引或协调类的各种服务。如果类在概念上只有一个实例（比如
NSWorkspace），就应该产生 一个单件实例，而不是多个实例；如果将来某一天可能有多个实例，您可
以使用单件实例机制，而不是工厂方法或函数。

24.动态绑定

—在运行时确定要调用的方法

动态绑定将调用方法的确定也推迟到运行时。在编译时，方法的调用并不和代码绑定 在一起，只有在消实发送出来之后，才确定被调用的代码。通过动态类型和动态绑定技术，您的代码每次执行都可以得到不同的结果。运行时因子负责确定消息的接 收者和被调用的方法。运行时的消息分发机制为动态绑定提供支持。当您向一个动态类型确定了的对象发送消息时，运行环境系统会通过接收者的 isa 指针定位对象的类，并以此为起点 确定被调用的方法，方法和消息是动态绑定的。而且，您不必在 Objective-C 代码中做任何工作，就可以自动获取动态绑定的好处。您在每次发送消息时，

特别是当消息的接收者是动态类型已经确定的对象时，动态绑定就会例行而 透明地发生。

25.obj-c 的优缺点

objc 优点：

1) Cateogies
2) Posing

3) 动态识别
4) 指标计算
5）弹性讯息传递
6) 不是一个过度复杂的 C 衍生语言
7) Objective-C 与 C++ 可混合编程
缺点:
1) 不支援命名空間
2) 不支持运算符重载

3） 不支持多重继承

4） 使用动态运行时类型，所有的方法都是函数调用，所以很多编译时优化方法都用不到。（如内联函数等），性能低劣。

26.sprintf,strcpy,memcpy 使用上有什 么要注意的地方

strcpy 是一个字符串拷贝的函数，它的函数原型为 strcpy(char *dst, const char *src);

将 src 开始的一段字符串拷贝到 dst 开始的内存中去，结束的标志符号为 '\0'，由于拷贝的长度不是由我们自己控制的，所以这个字符串拷贝很容易出错。具备字符串拷贝功能的函数有 memcpy，这是一个内存拷贝函数，它的函 数原型为 memcpy(char *dst, const char* src, unsigned int len);

将长度为 len 的一段内存，从 src 拷贝到 dst 中去，这个函数的长度可控。但 是会有内存叠加的问题。

sprintf 是格式化函数。将一段数据通过特定的格式，格式化到一个字符串缓 冲区中去。sprintf 格式化的函数的长度不可控，有可能格式化后的字符串会超出缓冲区的大小，造成溢出。

27. 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 intege）r
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 t
hat 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

28.readwrite，readonly，assign，retain，copy，nonatomic 属性的作用

@property 是 一个属性访问声明，扩号内支持以下几个属性：
1，getter=getterName，setter=setterName，设置 setter 与 getter 的方法名
2，readwrite,readonly，设置可供访问级别
2，assign，setter 方法直接赋值，不进行 任何 retain 操作，为了解决原类型与环循引用问题
3，retain，setter 方法对参数进行 release 旧值再 retain 新值，所有 实现都是这个顺序(CC 上有相关资料)
4，copy，setter 方法进行 Copy 操作，与 retain 处理流程一样，先旧值 release，再 Copy 出新的对象，retainCount 为 1。这是为了减少对上下文的依赖而引入的机制。
5，nonatomic，非原子性访问，不加同步， 多线程并发访问会提高性能。注意，如果不加此属性，则默认是两个访问方法都为原子型事务访问。锁被加到所属对象实例级(我是这么理解的...)。