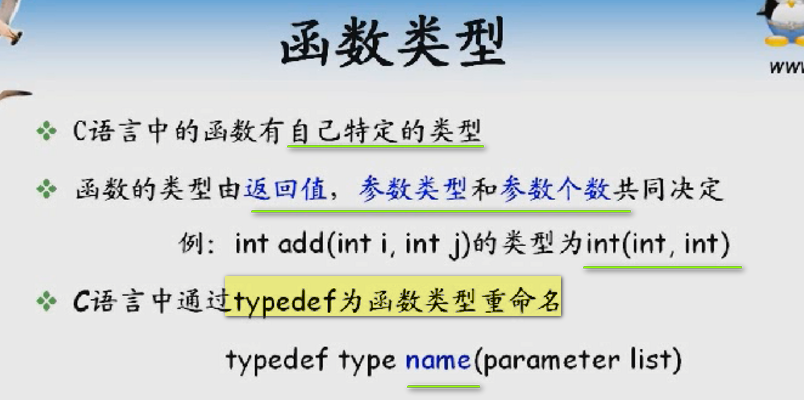
函数与指针分析



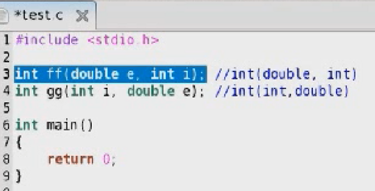
这个小节里面 讨论阅读代码非常重要的技巧 这之前介绍一个C语言中的新概念

函数类型**（Java是继承C++的 C++继承C的 所以 Java同样有函数类型 ----- Java C都有函数类型和数组类型）**



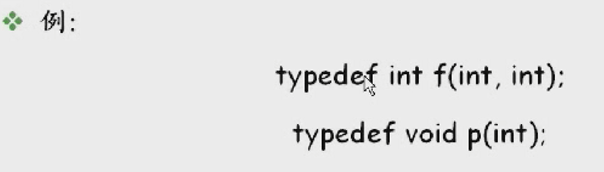
---- 函数类型还隐含了一个**参数出现的顺序**也是确定函数类型的一个重要的指标

【typedef里面 使用的第二种 就是 未知+已知融合】

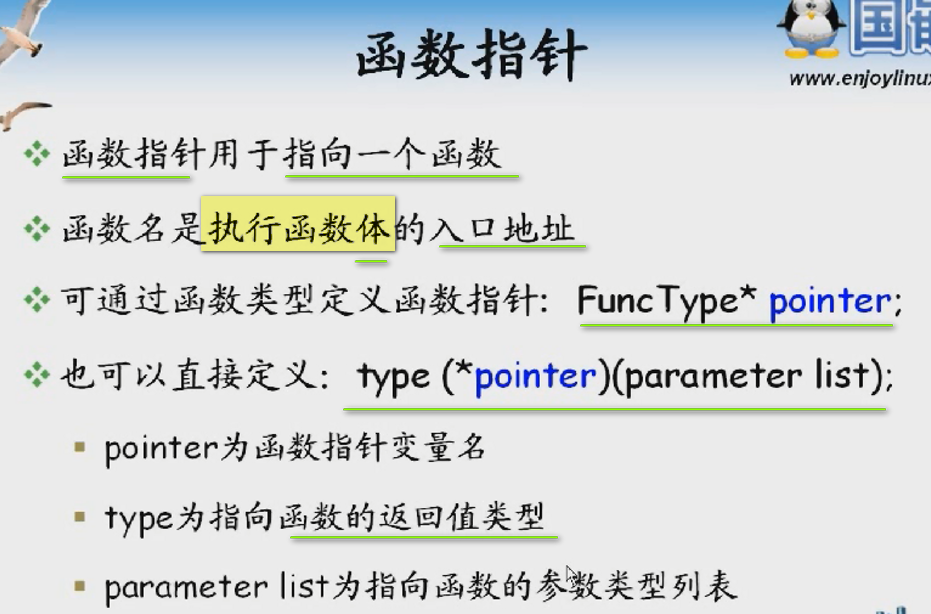


注意 从ff和gg的类型是截然不同的 double和int的顺序不一样 ---- 可以知道 这两个函数的类型是不一样的

C语言还是通过typedef重命名

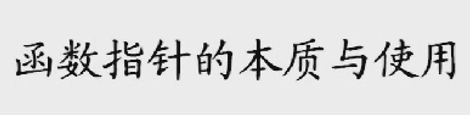


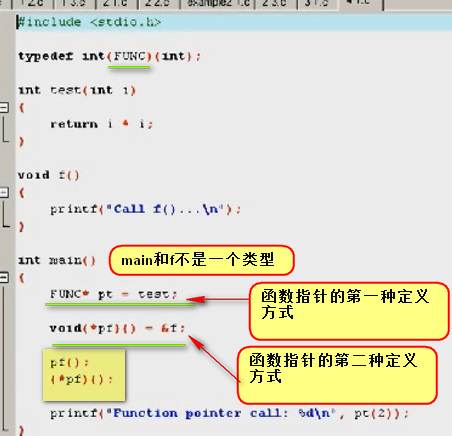
如果不看typedef 以为是函数声明 --- 加上了typedef 就可以知道f是函数别名



调用函数 就是到某个程序代码段执行代码 ---- 函数体代码的入口 就是函数名 --- C语言中函数名就是执行函数体的入口地址

数组名就是数组首元素的地址





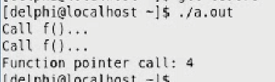
注意 f和test分别是两个函数体执行的入口地址 但是一个传test本身 另一个传的是&f ----- 是不是和数组a, &a类似 含义不一样呢？

------ 函数比较特殊 f和&f是一样的 给哪一个都可以 写没有写& 都一样 ---- 为什么要两个？老的编译器里面支持&f --- 这种方式 新的编译器变成了f （为了方便）---- 兼容 两种都有

------ 因为pf保存了函数的地址 所以 直接可以调用pf(); 也可以使用老式的方法 (\*pf)(); 没有本质的区别

pt(2)等价于直接调用test

运行

 没有本质区别

反映的问题：函数名是函数的入口地址 ----- 直接使用函数名

其实函数指针比数组指针还简单

====== 回调函数 ======

在linux内核 和 驱动的编写非常多 下面讲解回调函数的原理

场景：

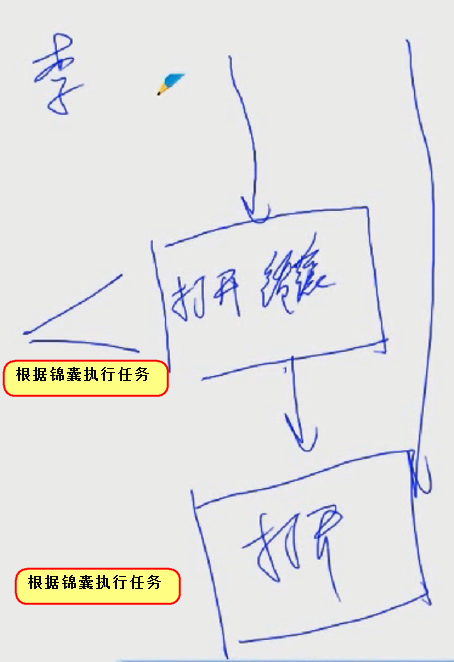
李逍遥要执行一个任务 一路上遇到很多艰险 但是不知道艰险是什么 ---- 所以 求助高人剑仙 ---- 告诉李逍遥 不用怕 我这有三个锦囊妙计 ---- 遇到艰险的时候 打开这些锦囊 告诉你怎么做

---- 李逍遥并不知道什么事件 什么地点遇到什么事情 ---- 但是知道打开锦囊查找办法

---- 剑仙他不会遇到这些事情 但是知道碰到事情该怎么处理 ---- 剑仙同志不会直接碰到事情 但是知道碰到这些事情该做什么事情 ----- 这样就分开了

李逍遥遇到第一个困难 --- 打开锦囊 就知道怎么做了

**回调机制就出来了**

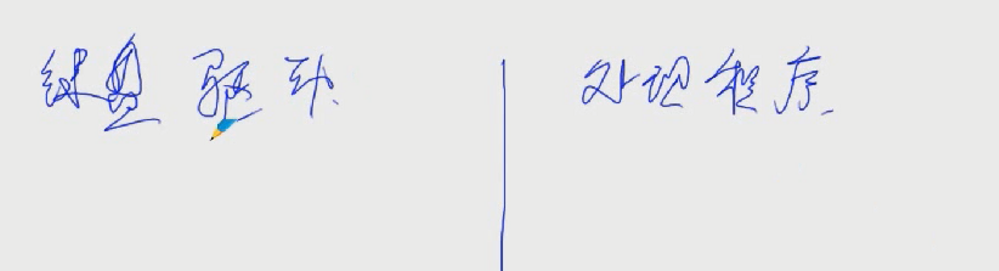


剑仙的任务就是出锦囊 写锦囊



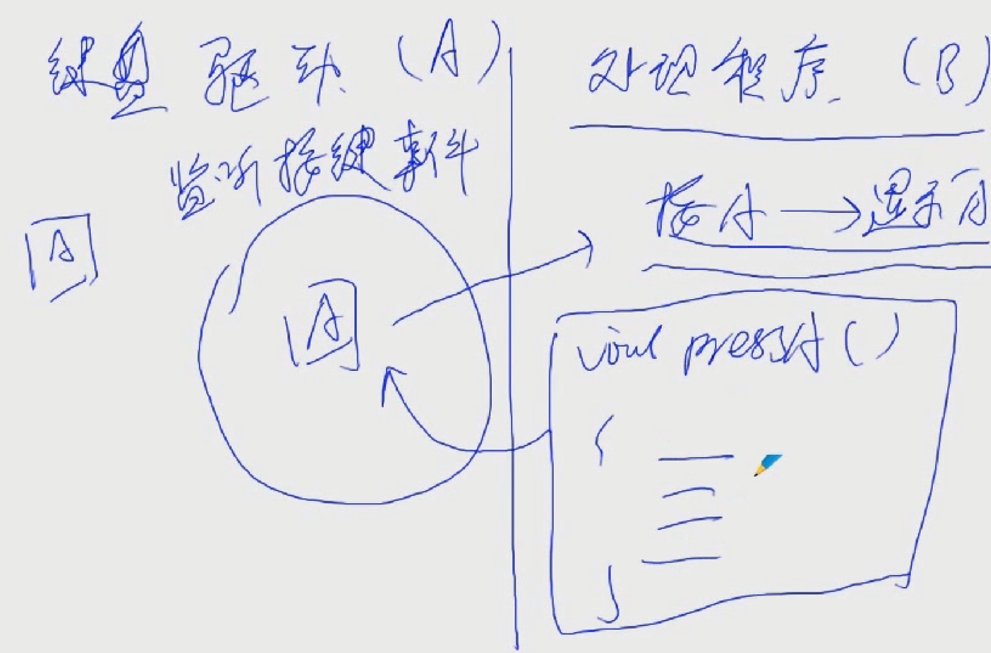
回调一定两个部分来完成 ----- 遇到事情+解决（不知道怎么解决） ----- 求助剑仙

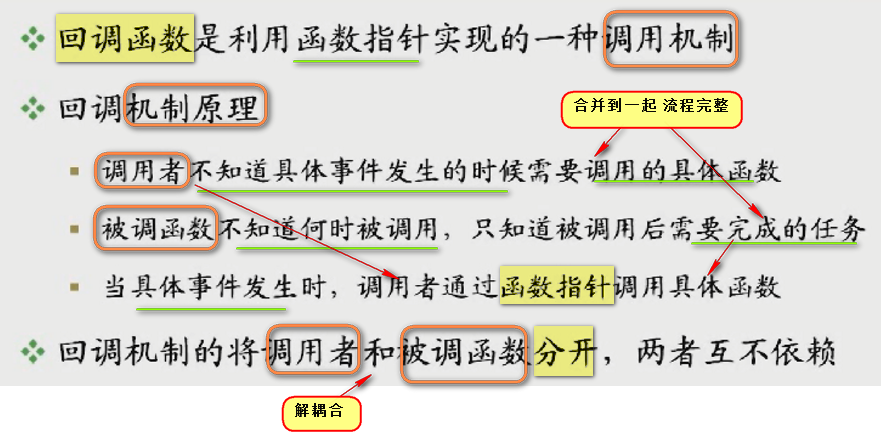
剑仙不会接触到这些事件 ----- 完成锦囊的编写

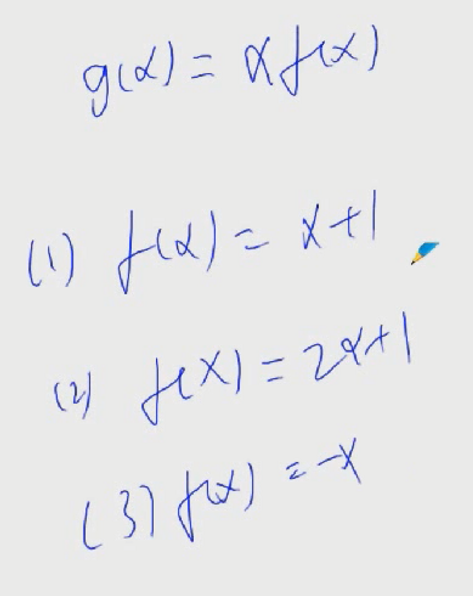


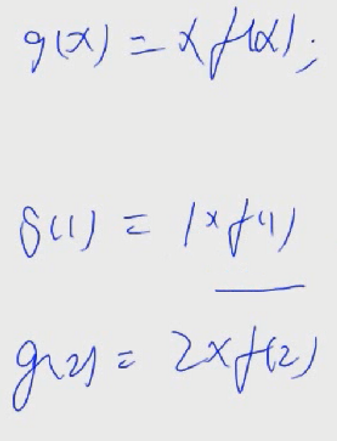
驱动和处理程序是两个人写的

驱动不知道a按键按下会发生什么事情 怎么处理 ---- 驱动程序就只能不停地监听按键按下这个事件 ----- 按键之后 就不知道该怎么做 ------ 类似于李逍遥遇到事情不知道怎么处理 ---- 这个时候 求助剑仙 ---- 处理程序无法得到按键事件 ------ 但是 处理程序知道该怎么做 比如按下a按键 显示a字符到屏幕 ----- 现在把显示字符a到屏幕 写成函数 做成锦囊 通过注册的方式 送给驱动程序

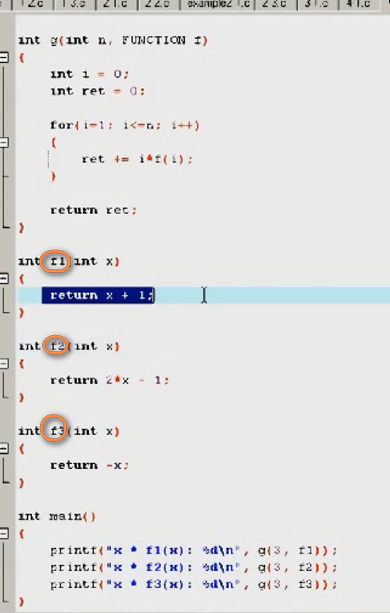




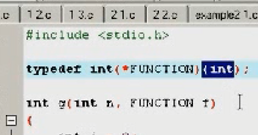
数学里面  ---- 这样f(x)可以取出无限多 但是g(x)不管哪个 就只有一个x\*f(x)

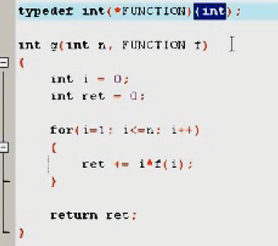
 f(x)不知道 只要给一个f(x) 就可以算出g(x)

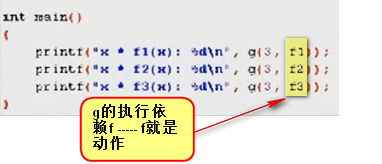
既然g(x)可以类比于李逍遥 剑仙就是3个f(x)的编写者 f(x)是三个锦囊



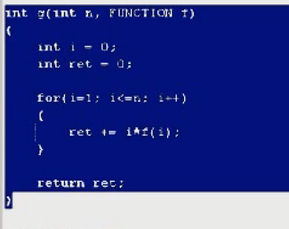
再看typedef

 这个FUNCTION是指向函数的指针





现在的g函数 铁定不能运行的

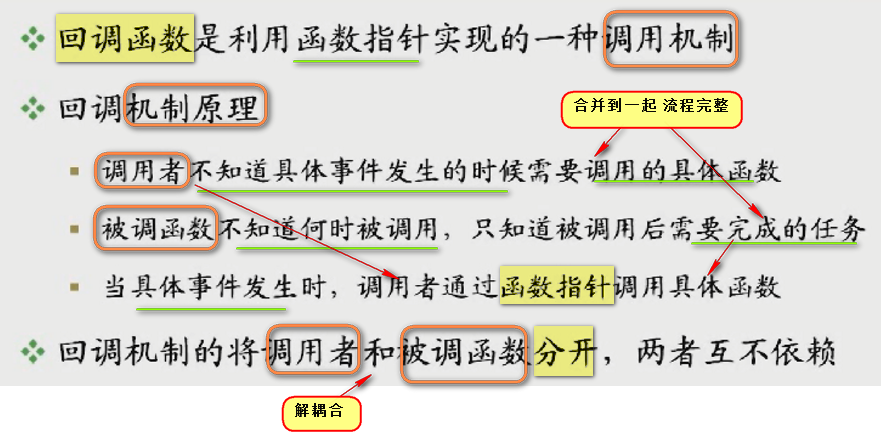
 因为不知道f是什么 这就要求助f

什么是回调函数的注册？

这个地方 这么写  有个学术的名字 是注册 ---- 也就是把f1作为参数 注册给g

【**什么是回调 什么是注册？**】

这样学会了回调函数 以及指导了什么是回调函数



【java在jdk8之前是使用接口和匿名内部类实现的】

F1 f2 f3何时被调用 自己不知道

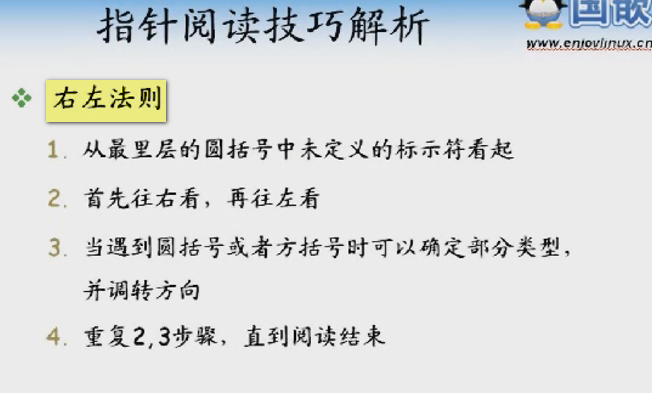
 这个就是第一个事件发生了 ---- g知道调用f1 等等

这是回调机制的原理 ---- 意义就是解耦合

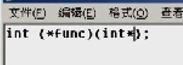
【spring中有很多回调 callback】

============== C语言代码的阅读技巧 ===========

随便打开一个文件



写一个函数指针



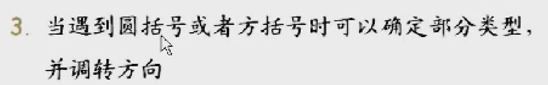
但是 C语言编译器怎么知道这个是函数指针？



这个是func 因为不是关键字 也没有事先定义 就是自定义的标识符

从这个看起  为什么不是从(int\*)看起 因为int是关键字 C已经定义好了



先往右看 

从这个func这个未定义的标识符---- 向右看 碰到了圆括号  并调整方向 ---- 那就向左看 看到 有\*之后 又碰到圆括号 可以停止一小步 ----- 发现我们的有一个\* 说明 func是一个指针

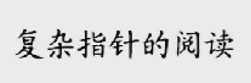
Func是指向什么类型的指针？ ------ 那么 把已经知道的信息的部分挖掉 

从挖掉的位置开始看 向右看 碰到圆括号 --- 就可以确定部分类型

 碰到左圆括号 可以确定 一定是一个函数 ---- 指针是指向函数的 是int\* ---- 返回值是什么 向左看 就是int

这样C语言编译器就知道 func就是定义了一个指针 指向了一个函数 参数是int\* 返回一个int

----- 这个是右左法则



 非常变态的东西 是什么呢？

右左法则





最里层的圆括号中未定义的标识符看起 ----- 老师直接说看p2 (我认为应该先看int(\*f)(int\*))

P2看起 右边) 反向\* 再看是( ---- 那么就是\*p2 ---- 说明p2是指针

----- 指向什么的指针？ 去掉(\*p2)

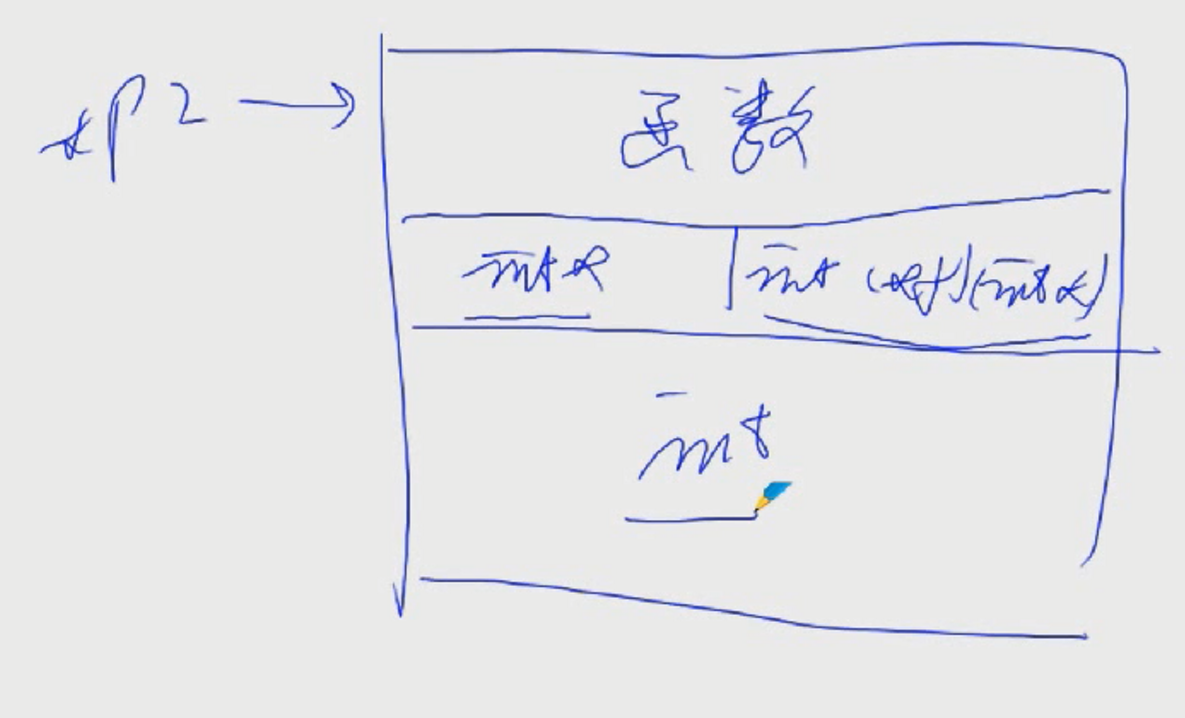
Int(int\*, int(\*f)(int \*))

------ 看到未定义的标识符f

同样方法 第二个参数f也是一个指针 指向了一个函数 函数类型是int(int\*)

这样 p2就是函数指针 --- 指向的函数的类型是int(int\*, int(\*f)(int\*)) ---第二个参数还是一个函数指针

如图



--------------------

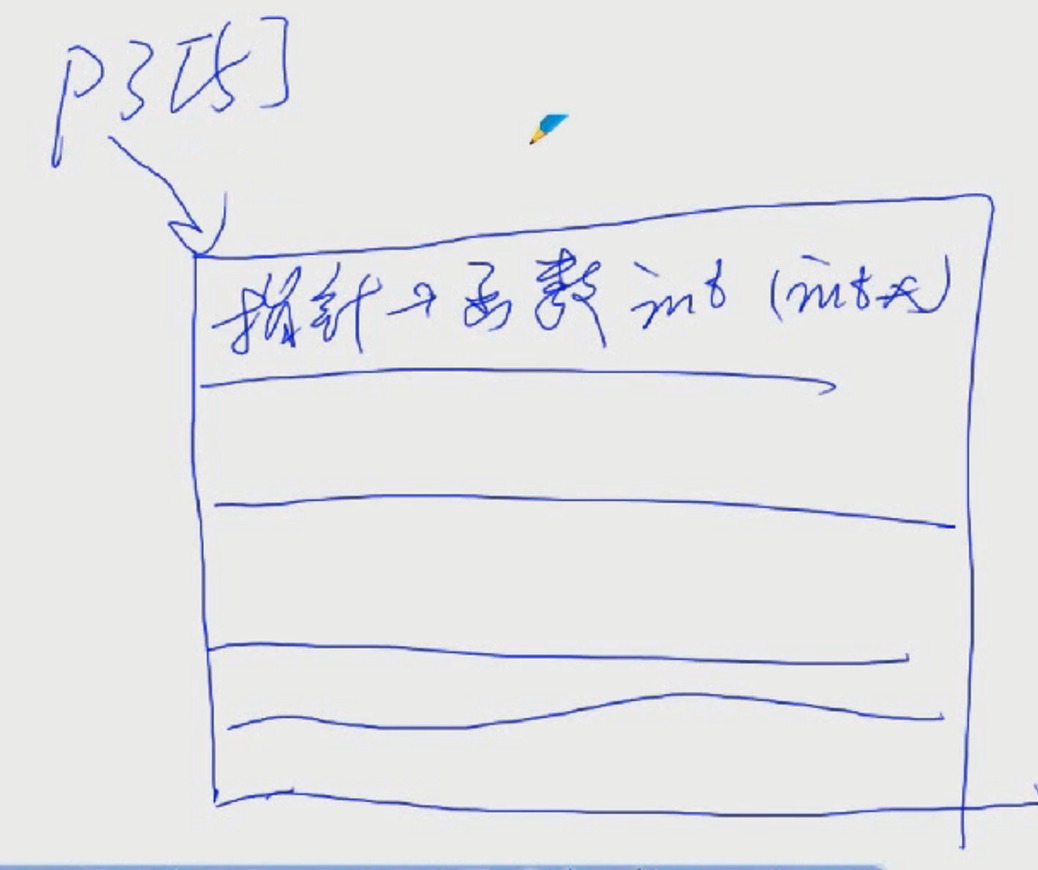


Int (\*p3[5])(int\*);

找到未定义的标识符 p3 --- p3 --- 右看看到[ ---- p3[5]是一个有五个元素的数组 ----- 数组大小确定了 但是 数组元素的类型是什么 --- 继续右左法则 但是 p3[5]去掉

Int(\*)(int\*) ---- 右看) 回头左看( ---- 说明 p3[5]中每一个元素是指针类型 ----- 还是没有确定 这个指针是指向什么类型的 剩下的右左法则

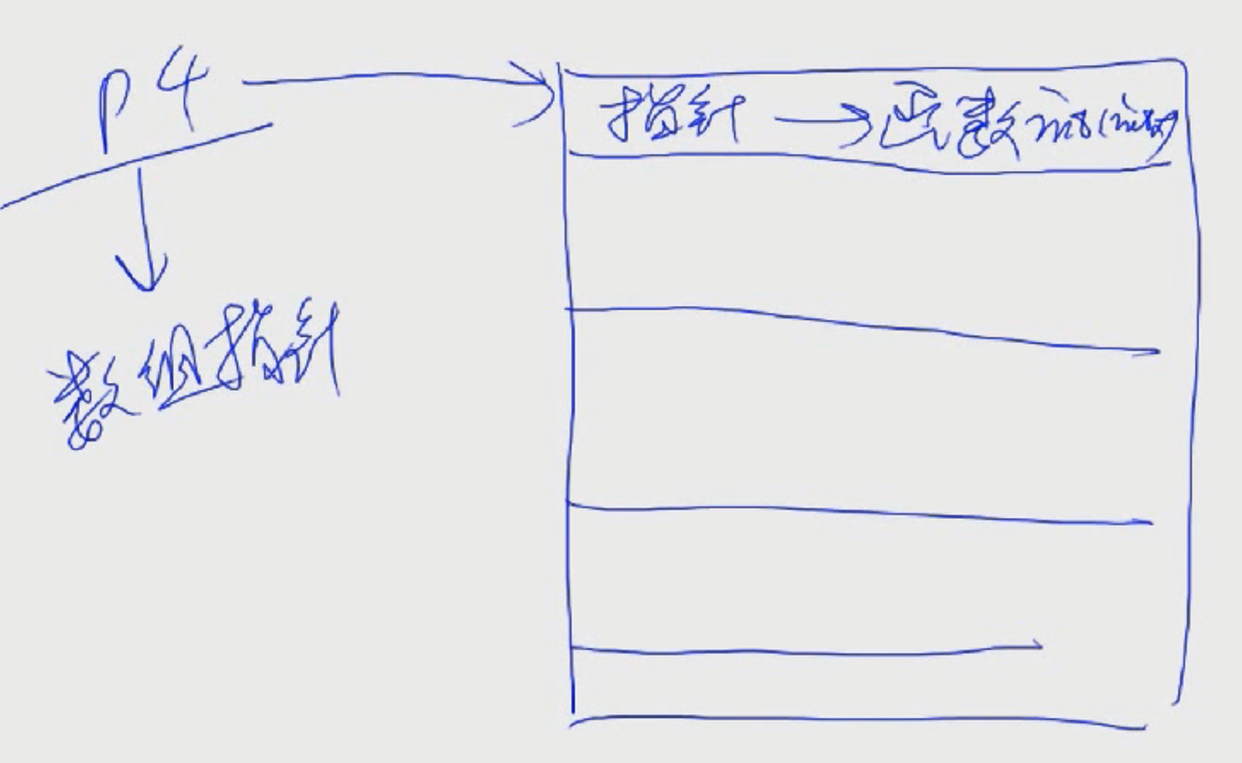
Int(int\*) ----- 所以 这个是一个指针数组 ---- 每个元素指向一个类型是的函数 ----函数类型是 int(int\*)

 这个p3是数组名 是首元素的地址 指向首元素 不是整个数组

---------------



P4是指向五个元素的指针 ---- 每个元素的类型是什么？ ---- 每个元素是指针 ---- 指向什么？ Int(int\*) ----指向int(int\*)类型的函数



P4是指向整个数组的指针 ---- 不是像上面p3仅仅指向数组的首元素 画法和意义都是不一样的

【老师说答疑课给答案】

首先看到p5未定义的标识符 右看) ----不知道 然后 ---- 左看\* ---- 然后( 知道p5是一个指针

---- 追问 指向什么类型的指针？

剩余部分int(\*(int\*))[5] ---- 从拿掉的位置 右看( ----- 说明p5是一个指向函数的指针 ---- 函数的返回值 参数都是什么呢？

剩余部分int(\*(int\*))[5] ---- 参数容易 就是(int\*) ---- 此时遇到了) 左看 看到了\* 说明返回值还是一个指针 ----- 追问 指向什么的指针？

剩余部分 int[5] ------ 所以 这个返回值的类型是一个数组指针 ---- 指向的数组大小是5 数组元素类型是int

这样 p5是指向函数的指针 ------ 函数的类型：参数int\* 返回值又是一个指针 ---返回值的类型是指针 --- 指向了5个元素大小的Int数组

**数组和指针是C语言的核心地位 ----- 以后的工作和学习才会比较顺利**