  typedef用来声明一个别名，typedef后面的语法，是一个声明。本来笔者以为这里不会产生什么误解的，但结果却出乎意料，产生误解的人不在少数。罪魁祸首又是那些害人的教材。在这些教材中介绍typedef的时候通常会写出如下形式：   
    typedef int PARA;

    这种形式跟#define int PARA几乎一样，如前面几章所述，这些教材的宗旨是由浅入深，但实际做出来的行为却是以偏盖全。的确，这种形式在所有形式中是最简单的，但却没有对typedef进一步解释，使得不少人用#define的思维来看待typedef，把int与PARA分开来看，int是一部分，PARA是另一部分，但实际上根本就不是这么一回事。int与PARA是一个整体！就象int i:声明一样是一个整体声明，只不过int i定义了一个变量，而typedef定义了一个别名。这些人由于持有这种错误的观念，就会无法理解如下一些声明：

typedef int a[10];  
typedef void (\*p)(void);  
  
    他们会以为a[10]是int的别名，(\*p)(void)是void的别名，但这样的别名看起来又似乎不是合法的名字，于是陷入困惑之中。实际上，上面的语句把a声明为具有10个int元素的数组的类型别名，p是一种函数指针的类型别名。  
  
    虽然在功能上，typedef可以看作一个跟int PARA分离的动作，但语法上typedef属于存储类声明说明符，因此严格来说，typedef int PARA整个是一个完整的声明。

定义一个函数指针类型。  
   比如原函数是   void   func(void);  
    那么定义的函数指针类型就是typedef   void   (\*Fun)(void);  
    然后用此类型生成一个指向函数的指针：   Fun   func1;  
    当func1获取函数地址之后，那么你就可以向调用原函数那样来使用这个函数指针：   func1(void);