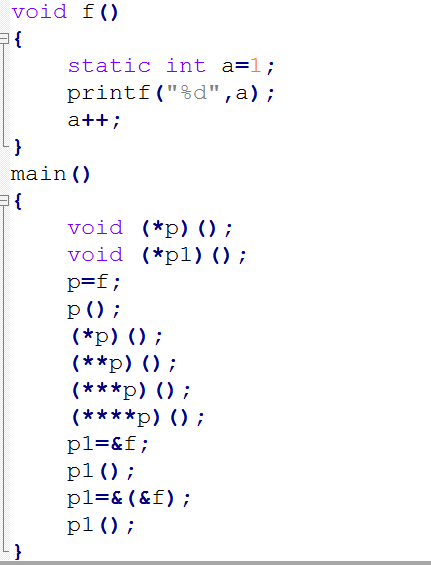
# 综合研究10问题记录

**问题**

1. **a(\_,\_)和(\*a)(\_,\_)有什么不同？**

答：对于标号 &a=a=\*a 所以没有区别。

验证如下：



编译链接生成exe文件 猜想结果应该为1234567

经运行得：



所以总结如下：

（&标号）是本身，（\*标号）也还是本身。

1. **int (far \*a)(char,char)和****int((far \*)a)(char,char)区别.**

答：int((far \*)a)(char,char) 这种定义函数指针是会出现错误。

错误1：declaration syntax error in function main

声明语法错误在主函数中

错误2：undefined symbol a in function main

在主函数中未定义符号 a

错误3：Non-portable pointer assignment in function main

不可移动的指针分配在主函数中

由错误1就可以得出一个结论，不能那样定义。(far \*)我们知道是强制类型转换，难道是在定义时不可以出现强制类型转换，不然会认为语法错误，从而导致其他的错误。

**3.指针函数实参void型，调用函数时实参能不能是NULL？**

答：经过查找头文件发现 

NULL其实就是0,所以当你的实参定义为void型 就不能有形参，也不能使NULL,因为NULL就是数字0呀。

验证结果如下：



意思是在调用的时候有了额外的参数，这说明NULL其实是数字0参数。

**对于使用函数指针调用函数的过程进行研究**

一定注意，char型参数压栈是将其扩展为字进行压栈的

读取的时候是按字大小一个一个读取，但是只会取一个字的低字节。

其他类型压栈还是通过实参类型（当你声明为不定参数）和你声明的实参类型进行压栈，读取的的时候会根据子函数的形参类型进行读取。返回是根据子函数的类型返回，得到的返回值最后由声明的返回类型决定，声明为char 则取al,声明为int则取ax,声明为long则取ax,dx. 最后你赋值给变量 如果和声明的类型一样则什么都不做，大于声明的类型，则会进行字扩展或双字扩展，小于声明类型则直接取低部分。

1. **如何向函数传递函数指针，这样做的意义是什么？**

答：要向函数传递函数指针，必须有函数指针类型的形参

比如函数为 f(void (\*p)()) 这个函数f就可以接受传递过来的函数的入口地址。 这样做的意义

1. 比如写一个创建任务的函数，它需要对每个函数创建任务，那么就需要向这个创建任务的函数传递函数指针。
2. 有的时候由于硬件原因 只会调用一个函数，但是我们又想调用其它函数，则可以通过向函数传递函数指针实现一个函数通过一些标志调用其它函数。这个函数就是专门为调用其它函数而生。
3. 很多时候用函数指针这种技巧，称为” **回调函数**”， 许多窗口系统使用回调函数连接多个动作。

**5、对返回值是函数指针的函数进行研究**

**研究方向提示**：函数指针类型的函数返回的函数指针在哪儿存放？

答：void (\*)()类型返回是偏移地址放在AX

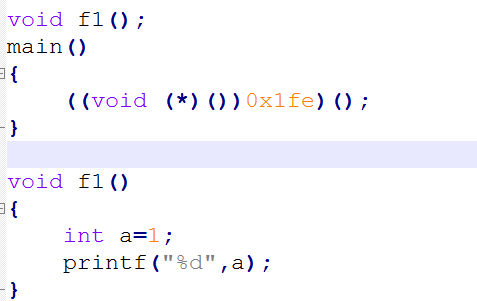
void (far \*)()类型返回时偏移地址加段地址，偏移放AX,段地址放DX

## 学长补充

1. **有这样一句代码“（\*（void（\*）（））0）（）;”这句代码的具体含义是什么？**

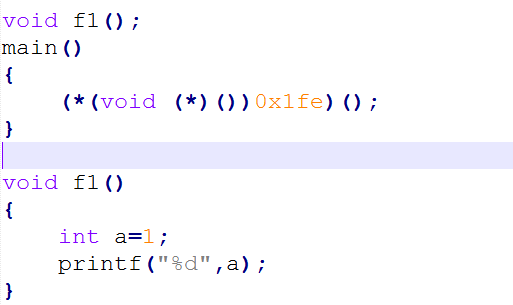
答：和（（void（\*）（））0）（）一样的含义

验证如下：



主函数对应的汇编：



****

主函数对应的汇编：



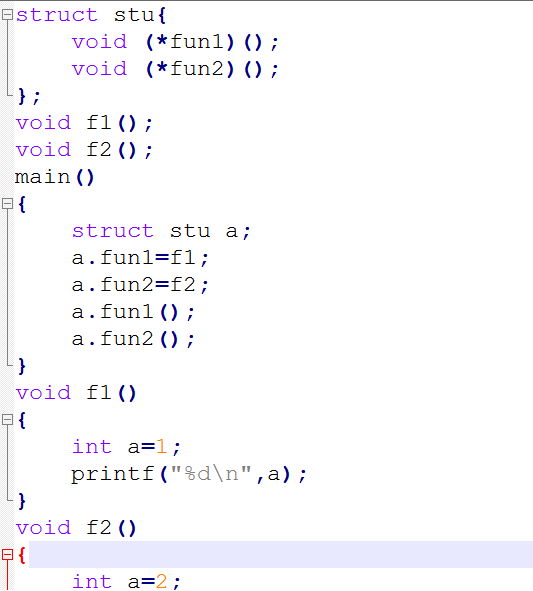
一模一样。

即：（\*（void（\*）（））0）（）和（（void（\*）（））0）（）一样的含义

都是将0强制转换为函数指针，最右的（）就是访问函数体。

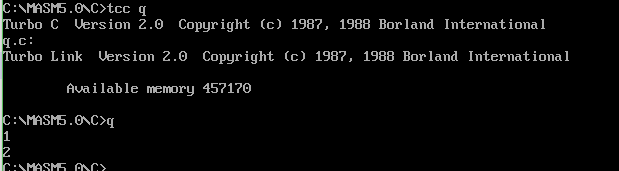
1. **函数指针可以作为结构体成员吗？**

答：我的猜想是当然可以。





编译链接结果：



1. **当下标不在合法界限内时，对数组进行越界访问（小于0或大于数组长度，如char a[3]={'a','b','c'};char b[3]={'d','e','f'}; a[3]=？b[-1]=？）会出现什么现象？深入理解数组机制。**

答：经发现数组里定义的内容开始就被放在 0194处，不管定义的是局部数组还是全局数组，但是局部数组会从0194处拷贝到堆栈里，如果你定义的数组大小是都是偶数，则两个数组在堆栈中没有空隙，如果是奇数，则两个数组在堆栈中存放间会有一个随机数。

a就是数组名 代表首地址，也是一种结构，大小为3个字节，但是a[3]就会是a[2]地址后的一个地址空间的内容，b[-1]就是b[0]地址前的一个地址空间的内容。

1. **函数指针的用途。**

答：

1. **对静态指针、常量指针与指针常量的理解。**

**答：**

常量指针，就是指向常量的指针，就是这个指针指向的空间为常量，是不允许被赋值的。 const int \* 或 int const \* 比如：

int b=1;

const int \*a=&b;

表示 \*a指向的是变量b的空间，且为常量，不允许再赋值

int c=1;

a=&c;这是合法的

\*a=2; 这就不合法了。

指针常量，就是一个常量，它是指针，开始赋值后就不能再进行赋值。

\*const int 或 int \*const 比如：

int b=1;

\*const int a=&b;

表示 a为常量且值为b变量的地址 这个常量是指针，指向变量b的空间

int c=1;

a=&c;这是不合法的

\*a=2; 这是合法的。

1. **可以将指针数组理解为二维数组或将函数指针数组名理解为二级指针吗，为什么？**

答：可以。

因为指针数组名是指针，指向的空间放的也是指针，就可以把其理解为二级指针。

1. **typedef的用法有哪些，这些用法之间有什么异同？**

答：

1. 定义一种类型的别名，而不只是简单的宏替换。可以用作同时声明指针型的多个对象
2. 可以定义double型为REAL 即 typedef double REAL; 有的平台只有float,我们移植只需要改typedef flaot REAL ,这样做就表现出强的移植性。
3. 我们定义结构体类型后，定义结构体变量 需要struct 结构体名 对象名

使用typedef 定义结构体新类型，定义结构体变量时只需要 新类型名 对象名。

1. **C++中引用的概念及其本质。**

答：引用就是目标变量的别名，对引用的操作与对变量直接操作效果完全相同。 引用申明完毕后，就相当一目标变量有两个名称。两个目标变量不能同时使用一个引用名。引用本身不占存储单元，系统也不给引用分配存储单元。

特点：

1. 传递引用给函数与传递指针的效果是一样的。
2. 使用引用传递函数的参数，在内存中并没有对形参分配空间，它是直接对实参操作
3. 使用指针作为函数的参数虽然也能达到与使用引用的效果，但是，在被调函数中同样要给形参分配存储单元

引用声明方式：类型标识符 &引用名=目标变量名；

引用本质就是指针常量