## 170806\_研讨会9、10、11

1. 定义复杂声明，把各自想到的列出来，从语法上分析，大家分享一下想法。

例：定义一个函数指针数组指针

同学A: void (\*( (\*a)[SIZE]) )(void);

同学B: int \* (\*) (char char) (\*p)[];

分步解释：

1> “\*p;” 声明p是一个指针 ；

2> “（\*p）[]; ” ，声明p是一个指针数组；

3> “\* （\*p）[];”，声明p指向数组，数组存放指针；

4> “int \* (\*) (char char) (\*p)[];”，声明p是一个函数指针数组指针，p指向数组，数组内存放函数指针。

但是第四步错了，正确的应该参考“（\* (（\*p）[] )）();”

1. 递归定义的局限性：不能定义有递归性质的变量。如：如何定义一个指针，“这个指针”指向的函数返回的指针也同“这个指针”一样？

例： “这个指针” 指向 “ (“这个指针”) f() ”

3、写一个“返回值为int型的数组指针的函数指针数组”

Int (\* (\*(p[]))() ) [] ;

递归定义。

开始分析的东西，元，不可分割，实现类型意义的最最基本载体。

基本类型 ：

1. P 已经不可以分割了，由p可以产生其他，比如指针：\*p
2. P[] 分割以后就不能表示数组了，由他可产生其他，比如指针数组（数组里存放的是指针）： \*p[]
3. P() 函数的表征形式，可以生成函数，如返回指针的函数：\*p()

所以在定义的时候我们应该是先要明白被定义的是那种基本类型，然后在由此递归的生成，即给基本类型赋予实际应用的意义。

而分析一个定义是什么（表示什么意思）的时候，应该先找到类型的承载体，即标识符，递归的分析对这个标识符的定义。即常说的左右法则：首先从未定义的标识符看起，然后往右看，再往左看。每当遇到圆括号时，就应该掉转阅读方向。一旦解析完圆括号里面所有的东西，就跳出圆括号。重复这个过程直到整个声明解析完毕。例：识别定义Int (\* (\*(p[]))() ) [];的语义（自己实践，及假设你是第一次拿到这个语句）。

4、不定参数宏

不定参数用到如下几个宏：

#define va\_start(ap, parmN) (ap = ...)

#define va\_arg(ap, type) (\*((type \*)(ap))++)

#define va\_end(ap)

#define \_va\_ptr (...)

#endif

理解这几个宏的定义，其中va\_arg(ap,type) 中先做的是指针的\*操作（取值），再做指针++操作。

5、重写9号中断

把处理的程序生成汇编程序：“tcc -S temp.c”。其中在temp.文件中的调用框架为：

Void installInt9h();

Main(){

……

installInth();

}

Void installInt9h(){

}

在temp.asm文件中，编写9号中断,即找到temp.asm文件中对应的installInt9h()函数的定义（也就是函数实现部分）往里头加需要的实现重装9号中断的代码。

1. 作业：用汇编重装9号中断，子函数是实验7，第二个子函数是实验11内容就是实验7（复制过去），意即实现用实验十一的函数调用方式调用实验七的功能函数；用exit()返回。装完9号中断开始打印，打印部分是将函数拿数组调用（实验11）。并实现以下对实验七的扩展要求：

3种情况：

中断里：

1>.“a”：结束循环打印，返回到主函数，再打印printf（“hello world!”）；

2>.“q”：退出程序；

3>.其他按键，继续循环打印；