结构体零长度数组的意义

在一些 C 语言编写的代码中，有时可以看到如下定义的结构：

typedef struct user\_def  
{  
    char \* name;  
    int length;  
    char bytes[0];  
} user\_def\_t;

这个 bytes 是什么意思？我们知道 0 == sizeof(bytes)，那么 bytes 仅仅是为了定义结构的尾地址吗？

不是的。这里的 bytes 是作为扩展数组用的。请看如下代码：

int alloc\_user\_def\_t(user\_def\_t \* p, int length)  
{  
    p = (user\_def\_t)malloc(sizeof(user\_def\_t) + length);  
    if (NULL == p)  
    {  
        return -1;  
    }  
  
    p->name = NULL;  
    p->length = length;  
    memset(p->bytes, 0, length);  
    return 0;  
}

是不是很酷？同样，也可以把 name 域的值附在结构后面：

int alloc\_user\_def\_t(user\_def\_t \* p, char \* name, int length)  
{  
    p = (user\_def\_t)malloc(sizeof(user\_def\_t) + strlen(name) + length + 1);  
    if (NULL == p)  
    {  
        return -1;  
    }

    p->name = p + sizeof(user\_def\_t) + length;  
    memcpy(p->name, name, strlen(name) + 1);   /\* 别忘了'/0' \*/  
    p->length = length;  
    memset(p->bytes, 0, length);  
    return 0;  
}

总结：在某一结构末尾如定义类似 char bytes[0] 的零长数组，表示该结构不定长，可通过数组的方式进行扩展。结构中必包含一个长度信息。结构本身类似于一个信息头。同时，此结构只能通过堆方式分配内存。

========================

1.零长度数组不是所有的c标准都支持，gnu c支持，ansi c不支持，c++不支持。

2.可以把结构体中定义了零长度数组的地方 视为结构体的结尾，在它之后最好不要再定义任何字段。因为零长度数组用来动态的添加数据，一旦添加数据后，零长度数组字段之后定义的字段如果之前赋了值，那么这个值就会被改掉。（结构体的里面的数据在内存中按顺序存储的。）

3.如何扩展：

可以直接使用数组方式添加信息：

p->bytes[0] = 'a';

p->bytes[1] = 'b';

也可以通过memcpy，strcpy等函数拷贝方式来添加。

4.等价写法：

typedef struct user\_def{

    char\* name;

    int      length;

    char   bytes[];

}user\_def\_t;

把char bytes[0] 写成 char bytes[]也可以，但是这样写的话在slickedit里面调试时，查看结构体的成员时是看不到bytes[]这个成员的，写成bytes[0]的话就可以。

5.注意的问题：

因为结构体是可以扩展的，所以在给结构体分配内存时必须预先分配足够的内存，保证后面扩展时不会超过预先分配的内存的大小。比如

*01***struct** arr{  
*02*    **int** a;  
*03*    **int** buff[0];  
*04*};  
05   
*06***int** main (**int** argc, **char** \*argv[])  
*07*{  
*08*    **struct** arr \*ptr;  
*09*    ptr = (**struct** arr\*)malloc(**sizeof**(**struct** arr));  
10     **int**\* q=(**int**\*)malloc(**sizeof**(**int**));  
*11*    **int**\* k=(**int**\*)malloc(**sizeof**(**int**));  
*12*  
*13*    printf("%x %x %x /r/n",ptr,q,k);  
*14*    \*q = 1;  
15     ptr->buff[0] = 2;  
*16*    ptr->buff[1] = 2;  
*17*    ptr->buff[2] = 2;  
*18*    ptr->buff[3] = 2;  
*19*    printf("%d/r/n",\*q);  
20 }

ptr指向结构体，q和k指向的int都是在堆里面分配的（ptr,q,k本身是在栈中分配的），分配的空间是16个字节（malloc好像默认是最小分16字节，如果大于16字节则按实际的分，在Slickedit和gcc下是这样，至于malloc的具体内存分配方法暂时没研究） 。

printf("%x %x %x /r/n",ptr,q,k);的结果是6b0208 6b0218 6b0228，可以看出分了16个字节的空间。

当执行到ptr->buff[2] = 2;时，struct的最后四个字节的内容被置成了2. 接下来ptr->buff[3] = 2就把原来q指向的空间的内容置成了2，所以最后printf("%d/r/n",\*q)的结果就是2了。