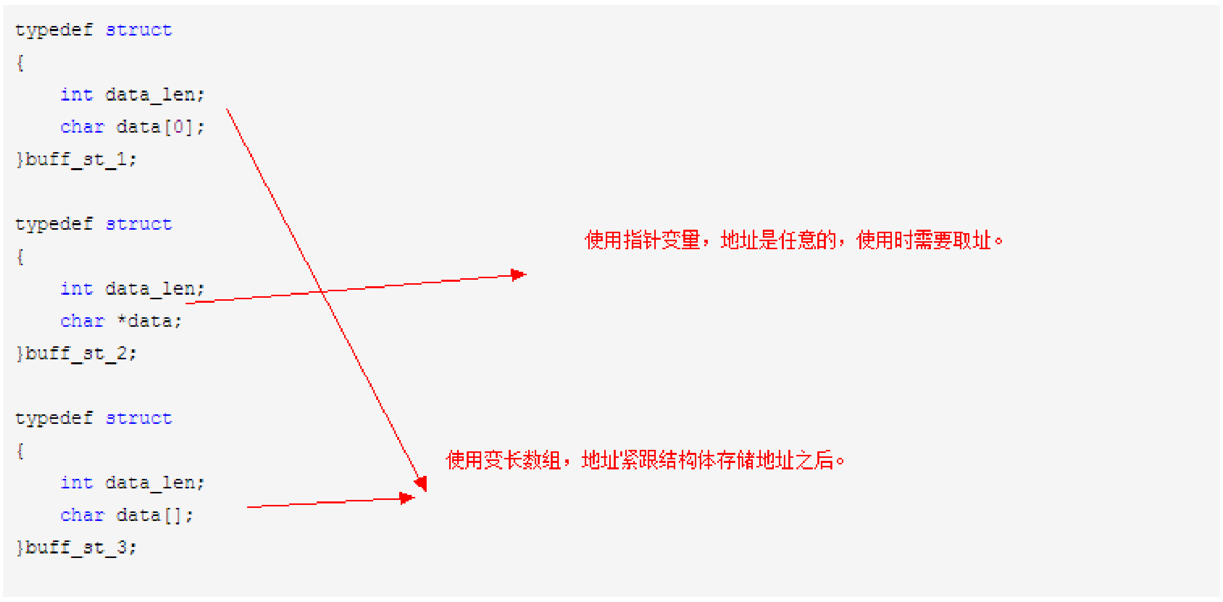
零长数组char[0]，char*，char[]

2、data[0]结构

经常遇到的结构形状如下：

```
struct buffer
{
    int data_len;   //长度
    char data[0];  //起始地址
};
```

在这个结构中，data是一个数组名；但该数组没有元素；该数组的真实地址紧随结构体buffer之后，而这个地址就是结构体后面数据的地址（如果给这个结构体分配的内容大于这个结构体实际大小，后面多余的部分就是这个data的内容）；这种声明方法可以巧妙的实现C语言里的数组扩展。



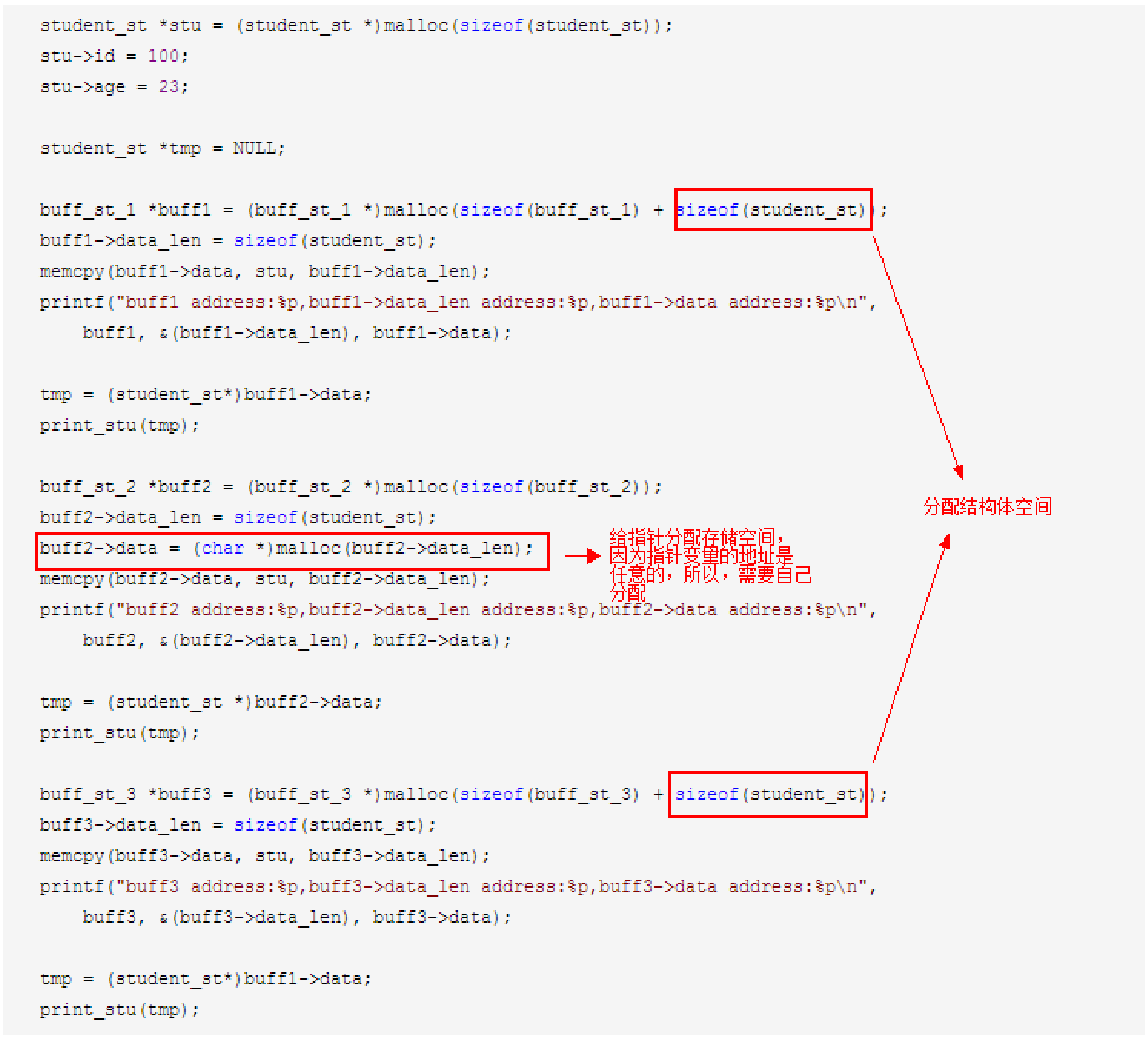
```
typedef struct
{
    int data_len;
    char data[0];
}buff_st_1;

typedef struct
{
    int data_len;
    char *data;
}buff_st_2;

typedef struct
{
    int data_len;
    char data[];
}buff_st_3;
```

使用指针变量，地址是任意的，使用时需要取址。

使用变长数组，地址紧跟结构体存储地址之后。



```c
student_st *stu = (student_st *)malloc(sizeof(student_st));
stu->id = 100;
stu->age = 23;

student_st *tmp = NULL;

buff_st_1 *buff1 = (buff_st_1 *)malloc(sizeof(buff_st_1) + sizeof(student_st));
buff1->data_len = sizeof(student_st);
memcpy(buff1->data, stu, buff1->data_len);
printf("buff1 address:%p,buff1->data_len address:%p,buff1->data address:%p\n",
    buff1, &(buff1->data_len), buff1->data);

tmp = (student_st*)buff1->data;
print_stu(tmp);

buff_st_2 *buff2 = (buff_st_2 *)malloc(sizeof(buff_st_2));
buff2->data_len = sizeof(student_st);
buff2->data = (char *)malloc(buff2->data_len);
memcpy(buff2->data, stu, buff2->data_len);
printf("buff2 address:%p,buff2->data_len address:%p,buff2->data address:%p\n",
    buff2, &(buff2->data_len), buff2->data);

tmp = (student_st *)buff2->data;
print_stu(tmp);

buff_st_3 *buff3 = (buff_st_3 *)malloc(sizeof(buff_st_3) + sizeof(student_st));
buff3->data_len = sizeof(student_st);
memcpy(buff3->data, stu, buff3->data_len);
printf("buff3 address:%p,buff3->data_len address:%p,buff3->data address:%p\n",
    buff3, &(buff3->data_len), buff3->data);

tmp = (student_st*)buff1->data;
print_stu(tmp);
```

给指针分配存储空间，因为指针变量的地址是任意的，所以，需要自己分配

分配结构体空间

参考链接：C语言变长数组data[0]