第六课

> c-abridged
>
> 104 -

1. 动态数组和内存释放

> 看 PPT 和 回放

malloc

realloc

free

2. struct 结构体

结构体定义：

```c
// 方式1
/* type definition */
struct point {
    int x;
    int y;
}; /* the end ';' is required */
/* point declaration (& alloc!) */
struct point pt;
/* pointer to a point */
struct point *pp;


// 方式2
/* combined definition & decls */
struct point {
    int x;
    int y;
} pt, *pp;

```

结构体的指针的使用：

```c
pp = &pt;
(*pp).x = 351; /* comp. access via pointer */
// 注意：*pp.x是错误的，这个的本质为*(pp.x)

// 或者

pp = &pt;
pp->x = 10;
pp->y = -5;
```

动态结构体以及内存释放：

```c
/* Dynamically allocating structs: */
struct point *parr1 = malloc(N * sizeof(struct point));
for (i=0; i<N; i++) {
 parr1[i].x = parr1[i].y = 0;
}
/* or, equivalently, with calloc (which zero-inits) */
struct point *parr2 = calloc(N, sizeof(struct point));
/* do stuff with parr1, parr2 ... */
free(parr1);
free(parr2);
```

函数的参数如果是结构体，传值不会改变原本结构体的元素，也就是说：

> struct 不是地址传参

除非是使用结构体指针，下面的操作会改变内部值：

```c
void foo(struct point *pp) {
 pp -> x = pp -> y = 10;
}
```

链表/节点：

```c
// 不同于python，链表的构建不能直接通过class node中储存一个node(next)实现
// c的结构体实现节点/链表 需要指针

// 下面是一个错误实现：
struct ll_node {
 char *data;
 struct ll_node next;
};
// 该实现会导致内存出错

// 正确实现：
struct ll_node {
 char *data;
 struct ll_node *next; /* need a pointer! */
};
```

完整的链表实现：

```c
struct ll_node {
 char *data;
 struct ll_node *next; /* need a pointer! */
};


struct ll_node *prepend(char *data, struct ll_node *next) {
    // 请注意！！！！！！！！
    // 由于指针必须要有指向，不能为空
    // 在 C 语言中，局部变量不会自动初始化
    // n 指针需要被初始化指向有效的内存区域
    // 所以这里通过分配内存的方式，对指针进行赋值
    struct ll_node *n = malloc(sizeof(struct ll_node));
    n->data = data;
    n->next = next;
    return n;
}

// 这里注意，free需要释放所有创建的节点
void free_llist(struct ll_node *head) {
 struct ll_node *p=head, *q;
 while (p) {
     q = p->next;
     free(p);
     p = q;
 }
}
```

链表赋值与输出：

```c
main() {
    struct ll_node *head = 0;
    head = prepend("reverse.", head);
    head = prepend("in", head);
    head = prepend("display", head);
    head = prepend("will", head);
    head = prepend("These", head);
    struct ll_node *p;
    for (p=head; p; p=p->next) {
     printf("%s ", p->data);
    }
    printf("\n");
    free_llist(head);
}
```

3. typeof

ppt 没有，看回放

4. debugg 工具

> very handy tool for detecting/debugging memory leaks: **valgrind**