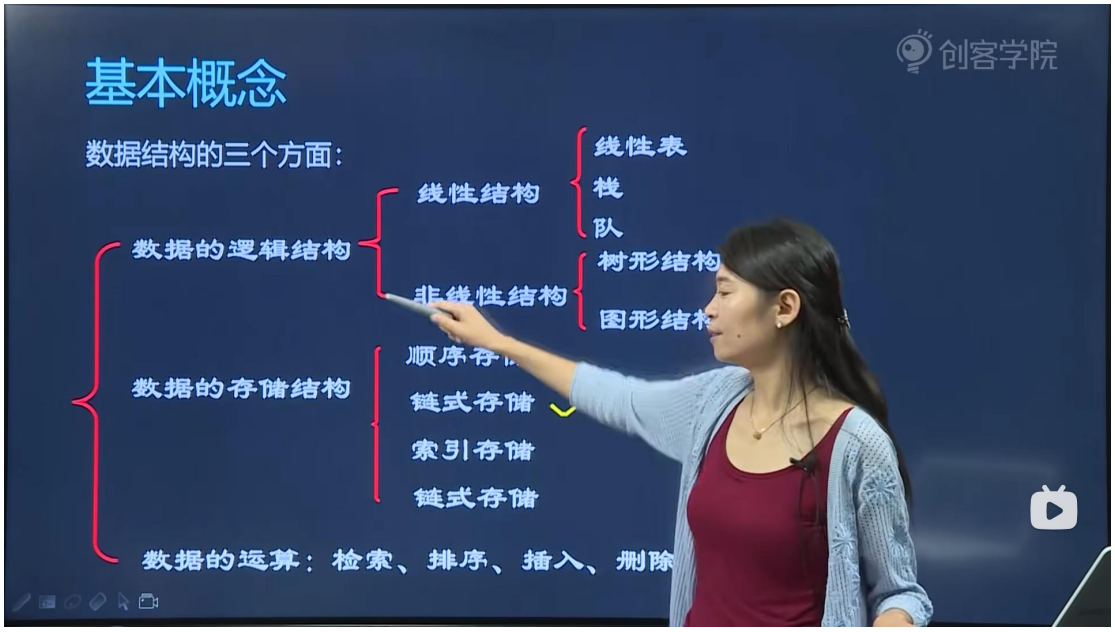
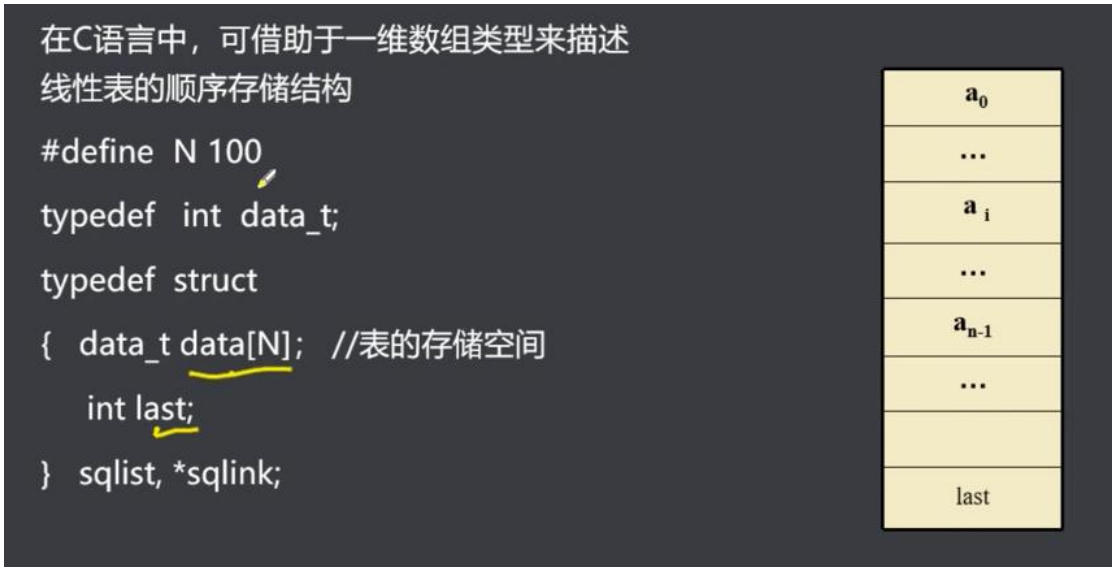


线性表之顺序表

表头无前驱，表尾无后继
中间每个元素 a(i)，有且只有一个直接前驱 a(i-1)，一个直接后继 a(i+1)

对应着的存储结构：顺序存储->依次存储在一片连续的空间
线性表的顺序存储特点：逻辑相邻物理存储也是相邻的、存储密度高
不足：对表的插入删除运算时间复杂度高
C 语言中借助一维数组类型来描述线性表的顺序存储结构



一个数组 data[N]来存储表的数据
last 来记录最后一个元素的下标

三个文件：sqlist.h, sqlist.c, test.c

① 结构清晰

② 可复用性

sqlist.h

```
15 typedef int data_t;
16 #define N 128
17
18 typedef struct {
19     data_t data[N];
20     int last;
21 }sqlist, *sqlink;
22
```

sqlist.c

实现.h（以我自己的命名习惯书写）

sqlist *list_create();

①申请内存

```
sqlist *L = (sqlist *)malloc(sizeof(sqlist));
```

malloc 到的内存是一段空间的起始地址

②初始化

```
memset(L, 0, sizeof(sqlist)); 写 0
L->last = -1; 空表标记
```

③返回

```
return L;
```

int list_insert_at(sqlist *L, int pos, data_t value);

①检查线性表是否满？

```
if(L->last == N-1){
    printf("list is full\n");
    return -1;
}
```

②检查传入位置是否合法 0<pos<=last+1

```
if(pos < 0 || pos > L->last+1){
    printf("pos is invalid\n");
    return -1;
}
```

③移动（从后往前）

```
int i;
for(i = L->last; i>= pos; i--){
    L->data[i+1] = L->data[i]
}
```

④存新值

```
L->data[pos] = value;
L->last++
return 0;
```

```
//清除表内容
int list_delete(sqlist *L){
        if(L == NULL){
        printf("list is NULL\n");
        return -1;
    }
    memset(L, 0, sizeof(sqlist));
    L->last = -1;
    return 0;
}

int list_delete_at(sqlist *L, int pos){
    if(L->last == -1){
        printf("list is empty\n");
        return -1;
    }
        if(pos < 0 || pos > L->last+1){
        printf("pos is invalid\n");
        return -1;
    }
    int i;
    for(i = pos+1; i<= L->last; i++){
        L->data[i-1] = L->data[i];
    }
    L->last--;
    return 0;
}

//删表
int list_destroy(sqlist *L){
    if(L == NULL){
        printf("list is NULL\n");
        return -1;
    }
    free(L);
    L = NULL;
    return 0;
}


int list_isempty(sqlist *L){
    if(L->last == -1)
        return 1;
```

```
    else
        return 0;
}

int list_show(sqlist *L){
    int i;
    if(L == NULL){
        printf("list is NULL\n");
        return -1;
    }
    if(L->last == -1){
        printf("list is empty\n");
        return -1;
    }
    for(i = 0; i<= L->last, i++){
        printf("%d ", L->data);
    printf("\n\n");
    return 0;
}

int list_locate(sqlist *L, data_t value){
    if(L == NULL){
        printf("list is NULL\n");
        return -1;
    }
    if(L->last == -1){
        printf("list is empty\n");
        return -1;
    }
    int i;
    for(i = 0; i<= L->last; i++){
            if(L->data[i] == value){
                //printf("element %d in %d\n", value, i);
                return i;
            }
        }
        printf("element %d not found in list\n", value);
        return -1;
}
```

思考

实现两个线性表的合并（求并集）：list_locate()和list_insert_at()

```
int list_merge(sqlist *L1, sqlist *L2){
    if(L1 == NULL || L2 == NULL){
        return -1;
    }
    int i,ret;
    for(i = 0; i <= L2->late; i++){
        ret = list_locate(L1, L2->data[i]);
        if(ret == -1){
            if(list_insert_at(L1, L1->last[i+1], L2->data[i]) == -1){
                printf(“merge faile!\n”);
                return -1;
            }
        }
    }
    return 0;
}
```

清除线性表的重复元素：

```
int list_purge(sqlist *L){
    if(L == NULL){
        return -1;
    }
    if(L->last == 0){
        return 0;
    }
    int i = 1;
    int j;
    while(i <= L->last){
        j = i-1;
        while(j>=0){
            if(L->data[i] == L->data[j]){
                list_delete_at(L, i);
                break;
            }else{
                j--;
            }
        }

        if(j < 0){
            i++;
        }
```

```
    }
    return 0;
}
```