

230704——数据结构

day1

```
1  数据结构：数据与数据之间的关系
2  逻辑结构：
3      根据数据的直接前驱数和直接后继数分为：
4      线性结构 和 非线性结构
5  存储结构：
6      顺序存储、链式存储 、 索引存储、 散列存储
7  运算：增、删、改、查、...
```

2.1

```
1  逻辑结构：
2      线性结构 和非线性结构
3  线性结构：
4      线性表、栈、队列
5      线性表：
6          顺序表、链表
7  非线性结构：
8      树、图
9  线性表的特点：
10     表头无前驱、表尾无后继；
11     其他元素有且仅有一个直接前驱和直接后继；
```

顺序表

```
1  逻辑结构：线性
2  存储结构：顺序存储
3
4  抽象顺序表：
5      #define SIZE 100
6      typedef int data_t;
7      typedef struct{
8          data_t buf[SIZE]; //数据
9          int index; //表尾指针
10     }sqlist_t;
11  运算：
12     创建表
13     sqlist_t *sqlistCreate();
14     判空
15     int sqlistIsEmpty(sqlist_t *sqlist);
16     判满：
```

```

17     int sqlistIsFull(sqlist_t *sqlist);
18     按照位置插入：
19     int sqlistInsertPos(sqlist_t *sqlist, int pos, data_t
    data)
20     求长度：
21     int sqlistGetLength(sqlist_t *sqlist);
22     打印顺序表
23     void SqlistShow(sqlist_t *sqlist);
24
25     //按位置删除
26     int sqlistDeletePos(sqlist_t *sqlist, int pos);
27     //按数据查找
28     int sqlistSearchData(sqlist_t *sqlist, data_t data);
29     //更新
30     int sqlistUpdataData(sqlist_t *sqlist, data_t olddata,
    data_t newdata);
31     //表的清空
32     int sqlistClear(sqlist_t *sqlist);
33     //表的销毁
34     int sqlistDestroy(sqlist_t **sqlist);
35
36     //按数据删除
37     void sqlistDeleteData(sqlist_t *sqlist, data_t data);
38
39     //按位置查找
40     data_t sqlistSearchPos(sqlist_t *sqlist, int pos);
41     //按位置更新
42     int sqlistUpdataPos(sqlist_t *sqlist, int pos, data_t
    data);
43     //排序
44     void sqlistSort(sqlist_t *sqlist);
45
46     顺序表的优缺点：
47     缺点：
48     1.可能会存在浪费空间的情况或者空间不够；
49     2.在删除或者插入数据的时候，数据会发生成片的移动，效率低；
50     有点：
51     查找很方便；
52
53

```

链表

```

1     逻辑线性；
2     存储：链式存储
3
4     抽象数据结构：
5     typedef int data_t;
6     typedef struct Node{
7         data_t data; //数据域
8         struct Node *next; //指针域

```

```

9  }link_t;
10
11  1.创建一个链表
12  link_t *linkCreate()
13  {
14      link_t * head = (link_t *)malloc(sizeof(link_t));
15      if(head == NULL){
16          return NULL;
17      }
18      head->data = -1;
19      head->next = NULL;
20
21      return head;
22  }
23  2.int linkGetLength(link_t *head)
24  {
25      if(head == NULL){
26          return -1;
27      }
28      int len = 0;
29      link_t *p = head->next;
30      while(p != NULL){
31          len++;
32          p = p->next;
33      }
34      return len;
35  }
36
37  3.判空
38  int linkIsEmpty(link_t *head)
39  {
40      if(head == NULL){
41          return -1;
42      }
43      return head->next == NULL;
44  }
45
46  4.遍历链表
47  void linkShow(link_t *head)
48  {
49      if(head == NULL){
50          return;
51      }
52      link_t *p = head->next;
53      while(p != NULL){
54          printf("%d ", p->data);
55          p = p->next;
56      }
57  }
58  5.按位置插入
59  int linkInsertPos(link_t *head, int pos, data_t
data)

```

```

60 {
61     //1. 判断链表是否存在
62     if(head == NULL){
63         return -1;
64     }
65     //2.判断位置是否合法
66     len = linkGetLength(head);
67     if(pos < 0 || pos > len){
68         printf("pos error\n");
69         return -1;
70     }
71     //3.找到要插入位置的前一个节点
72     link_t *p = head;
73     int i = 0;
74     for(; i < pos; i++){
75         p = p->next;
76     }
77     //4.创建新节点
78     link_t *new = (link_t *)malloc(sizeof(link_t));
79     if(new == NULL){
80         printf("malloc failed...\n");
81         return -1;
82     }
83     new->data = data;
84     //插入:
85     //让新节点指向后一个节点
86     new->next = p->next;
87     //让前一个节点指向新节点
88     p->next = new;
89
90     return 0;
91 }
92 }
93 6.按位置删除
94 int linkDeletePos(link_t *head, int pos)
95 {
96     //1.判断链表是否为空
97     if(linkIsEmpty(head)){
98         printf("head is empty\n");
99         return -1;
100     }
101     //2.判断位置是否合法
102     int len = linkGetLength(head);
103     if(pos < 0 || pos > len-1){
104         printf("pos error\n");
105         return -1;
106     }
107     //3.删除
108     //3.1 找到要删除的前一个节点
109     int i = 0;
110     link_t *p = head;
111     for(; i < pos; i++){

```

```

112     p = p->next;
113 }
114 //保存要删除的节点的地址
115 link_t *q = p->next;
116 p->next = q->next;
117 free(q);
118 q = NULL;
119 return 0;
120 }
121 7.按数据查找
122 link_t *linkSearchData(link_t *head, data_t data)
123 {
124     if(head == NULL){
125         return -1;
126     }
127     link_t *p = head->next;
128     while(p != NULL){
129         if(p->data == data){
130             return p;
131         }
132         p = p->next;
133     }
134     return NULL;
135 }
136 8.int linkUpdateData(link_t *head, \
137                     data_t olddata, data_t
newdata)
138 {
139     if(head == NULL){
140         return -1;
141     }
142     link_t *p = linkSearchData(head, olddata);
143     p->data = newdata;
144     return 0;
145 }
146 9.void linkClear(link_t *head)
147 {
148     while(!linkIsEmpty(head)){
149         linkDeletePos(head, 0);
150     }
151     return;
152 }
153 10.void linkDestroy(link_t **head)
154 {
155     linkClear(*head);
156     free(*head);
157     *head = NULL;
158
159     return;
160 }
161
162 链表的优缺点:

```

```
163 插入删除比较方便，不用考虑预分配空间的问题
164 缺点：
165 访问效率低，单独分配一个指针的空间保存下一个节点的地址
```

单向循环链表

```
1  typedef struct Node{
2      data_t data;
3      struct Node *next;
4  }link_t;
5  link_t * cir_linkCreate()
6  {
7      link_t *head = (link_t *)malloc(sizeof(link_t));
8      if(head == NULL)
9      {
10         printf("malloc failed...\n");
11         head->data = -1;
12         head->next = head;
13     }
14     return head;
15 }
16
17 约瑟夫环问题?
```

双向链表

```
1  typedef int data_t;
2  typedef struct Node{
3      data_t data;
4      struct Node *next;
5      struct Node *prior;
6  }link_t;
7  //1.创建一个双向链表
8  link_t * D_linkCreate()
9  {
10     link_t *head = (link_t *)malloc(sizeof(link_t));
11     if(head == NULL){
12         printf("malloc failed...\n");
13         return NULL;
14     }
15     head->data = -1;
16     head->next = NULL;
17     head->prior = NULL;
18
19     return head;
20 }
21 }
```

栈

```
1 堆栈简称栈；
2  线性结构；
3  定义：只允许一端进行插入或者删除的线性表；允许插入或者删除的一
   端
4      称为栈顶，另一端称为栈底；
5  特点：后进先出；
6  抽象数据结构：
7  顺序栈：
8      #define SIZE 100
9      typedef int data_t;
10     typedef struct{
11         data_t data[SIZE];
12         int top; // 栈顶"指针"
13     }stack_t;
14
15 运算：
16     //1.创建栈
17     //2.入栈
18     //3.出栈
19     //4.获取栈顶元素
20     //5.判空
21     //6.判满
22     //7.获取长度
23
24     // 遍历
25     //清空
26     //销毁
```

链式栈

```
1  typedef int data_t;
2  typedef struct Node{
3      data_t data;
4      struct Node *next;
5  }lstack_t;
6  将链表的头节点作为栈顶，方便入栈或者出栈；
7
8  运算：
9      //1.创建栈
10     //2.入栈
11     //3.出栈
12     //4.获取栈顶元素
13     //5.判空
14     //6.判满
15     //7.获取长度
16
```

```
17 // 遍历
18 //清空
19 //销毁
```

队列

```
1 允许在两端进行插入或者删除的线性表称为队列;允许插入的一端称为队
   尾;
2 允许删除的一端称为队头;
3 特点:
4     先进先出;
5
6     逻辑关系: 线性
7     存储:
8         顺序存储 顺序队
9         链式存储 链队
10    顺序队列:
11    抽象数据结构:
12        typedef int data_t;
13        typedef struct{
14            data_t data[100];
15            int front; // 队头"指针"
16            int rear;  //队尾 "指针"
17        }sq_t;
18
19 //1.创建队列
20 //2.判空
21 //3.判满
22 //4.队列的长度
23 //5.入队
24 //6.出队
25 //7.清空队列
26 //8.销毁
27
```

链式队列

```
1 抽象数据结构:
2     typedef int data_t ;
3     typedef struct Node{
4         data_t data;
5         struct Node *next;
6     }link_t;
7
8     typedef struct{
9         link_t *front;
10        link_t *rear;
11    }lqueue_t;
```


