

> 一研为定

@[toc]

第五次 直播 双向链表

双链表结点的定义:

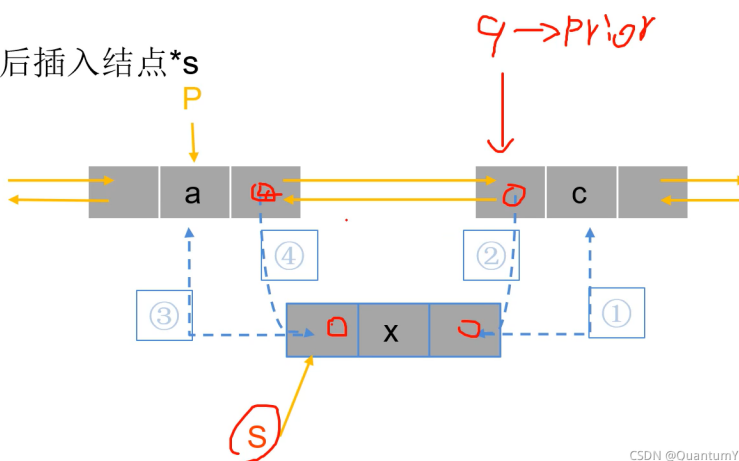
```
typedef struct DNode{           //双链表结点类型
    ElemType data;              //数据域
    struct DNode *prior;        //前驱指针
    struct DNode *next;         //后继指针
}DNode, *DLinkedList;
```

CSDN @QuantumYou

核心: 注意双向链表的插入次序 ①②③④ 标识

在双链表中p所指的结点之后插入结点*s

- 1、s->next=p->next
- 2、p->next->prior=s;
- 3、s->prior=p;
- 4、p->next=s;



CSDN @QuantumYou

注意: 赋值语句的解读 eg: `p->next = s;` 的意思为将 `s` 的值赋给 `p->next`

双向链表的增删改查

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef int ElemType;
5  typedef struct DNode{
6      ElemType data;
7      struct DNode *prior,*next;//前驱, 后继
8  }DNode,*DLinkedList;
9  //双向链表头插法
10 DLinkedList Dlist_head_insert(DLinkedList &DL)
11 {
12     DNode *s;int x;
13     DL=(DLinkedList)malloc(sizeof(DNode));//带头结点的链表,不带头结点
14     DL->next=NULL;
15     DL->prior=NULL;
16     scanf("%d",&x);//从标准输入读取数据
17     //3 4 5 6 7 9999
```

```

18     while(x!=9999){
19         s=(DLinkedList)malloc(sizeof(DNode)); //申请一个空间空间，强制类型转换
20         s->data=x;
21         s->next=DL->next;
22         if(DL->next!=NULL) //插入第一个结点时，不需要这一步操作
23         {
24             DL->next->prior=s;
25         }
26         s->prior=DL;
27         DL->next=s;
28         scanf("%d",&x); //读取标准输入
29     }
30     return DL;
31 }
32 //双向链表尾插法
33 DLinkedList Dlist_tail_insert(DLinkedList &DL)
34 {
35     int x;
36     DL=(DLinkedList)malloc(sizeof(DNode)); //带头节点的链表
37     DNode *s,*r=DL;
38     DL->prior=NULL;
39     //3 4 5 6 7 9999
40     scanf("%d",&x);
41     while(x!=9999){
42         s=(DNode*)malloc(sizeof(DNode));
43         s->data=x;
44         r->next=s;
45         s->prior=r;
46         r=s; //r指向新的表尾结点
47         scanf("%d",&x);
48     }
49     r->next=NULL; //尾结点的next指针赋值为NULL
50     return DL;
51 }
52 //按序号查找结点值
53 DNode *GetElem(DLinkedList DL,int i)
54 {
55     int j=1;
56     DNode *p=DL->next;
57     if(i==0)
58         return DL;
59     if(i<1)
60         return NULL;
61     while(p&&j<i)
62     {
63         p=p->next;
64         j++;
65     }
66     return p;
67 }
68 //新结点插入第i个位置
69 bool DlistFrontInsert(DLinkedList DL,int i,ElemType e)
70 {
71     DLinkedList p=GetElem(DL,i-1);
72     if(NULL==p)
73     {
74         return false;
75     }

```

```

76     DLinkedList s=(DLinkedList)malloc(sizeof(DNode)); //为新插入的结点申请空间
77     s->data=e;
78     s->next=p->next;
79     p->next->prior=s;
80     s->prior=p;
81     p->next=s;
82     return true;
83 }
84 //删除第i个结点
85 bool DListDelete(DLinkedList DL,int i)
86 {
87     DLinkedList p=GetElem(DL,i-1);
88     if(NULL==p)
89     {
90         return false;
91     }
92     DLinkedList q;
93     q=p->next;
94     if(q==NULL)//删除的元素不存在
95         return false;
96     p->next=q->next;//断链
97
98     // 下面注意要进行判断
99     if(q->next!=NULL)
100     {
101         q->next->prior=p;
102     }
103     free(q); //释放对应结点的空间
104     return true;
105 }
106 //链表打印
107 void PrintDList(DLinkedList DL)
108 {
109     DL=DL->next;
110     while(DL!=NULL)
111     {
112         printf("%3d",DL->data);
113         DL=DL->next;
114     }
115     printf("\n");
116 }
117
118
119 //2.3.3 双链表增删查
120 int main()
121 {
122     DLinkedList DL;
123     DLinkedList search;
124     Dlist_head_insert(DL);
125     //Dlist_tail_insert(DL);
126     //3 4 5 6 7 9999
127     PrintDList(DL);
128     search=GetElem(DL,2);
129     if(search!=NULL)
130     {
131         printf("按序号查找成功\n");
132         printf("%d\n",search->data);
133     }

```

```

134     DListFrontInsert(DL,3,99);
135     PrintDList(DL);
136     DListDelete(DL,2);
137     PrintDList(DL);
138     system("pause");
139 }

```

双向链表的删除

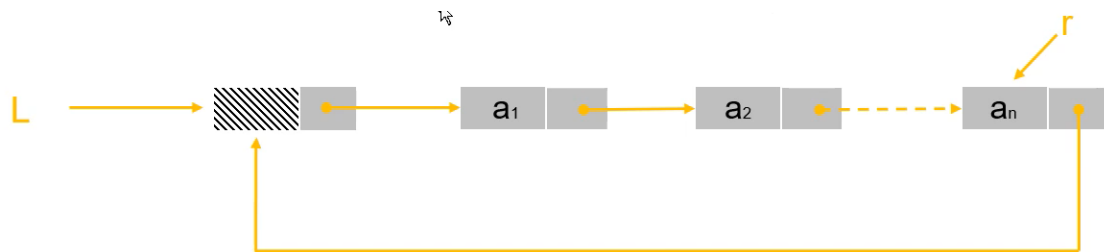
= 删除双链表中结点 *p 的后继结点 *q

- 1、 p->next=q->next ;
- 2、 q->next-> prior=p;
- 3、 free(q) ;

第五次 直播 其他链表

循环单链表

- 循环单链表与单链表的区别在于，表中最后一个结点的next指针不是NULL,而是指向头结点L,从而整个链表形成一个环



r->next=L;

while(pcur->next!=L)

CSDN @QuantumYou

静态链表

- 静态链表是借助数组来描述线性表的链式存储结构，结构类型如下

```

1  #define Maxsize 50
2  typedef struct {
3      ElemType data ;
4      int next ;
5  } Slinklist[Maxsize] ;

```

0		2
1	b	6
2	a	1
3	d	-1
4		
5		
6	c	3



CSDN @QuantumYou

常见问题

- 在链表中插入第 i 个位置和删除第 i 个元素不需要用引用的原因在于：不需要改变头节点

第六次 直播 引用解析、栈与队列

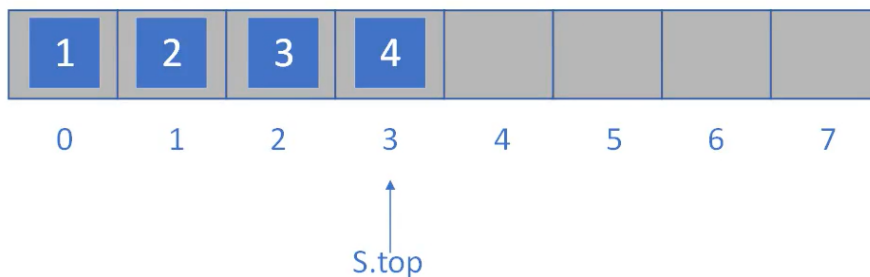
为什么我们需要在形参的地方使用引用？

- 在子函数中去给对应的形参赋值后，子函数结束，主函数中对应的实参就发生了变化，如果没有使用引用，那么在子函数中给形参赋值后，子函数结束，主函数中对应的实参不会变化的

栈的实现

顺序存储实现栈

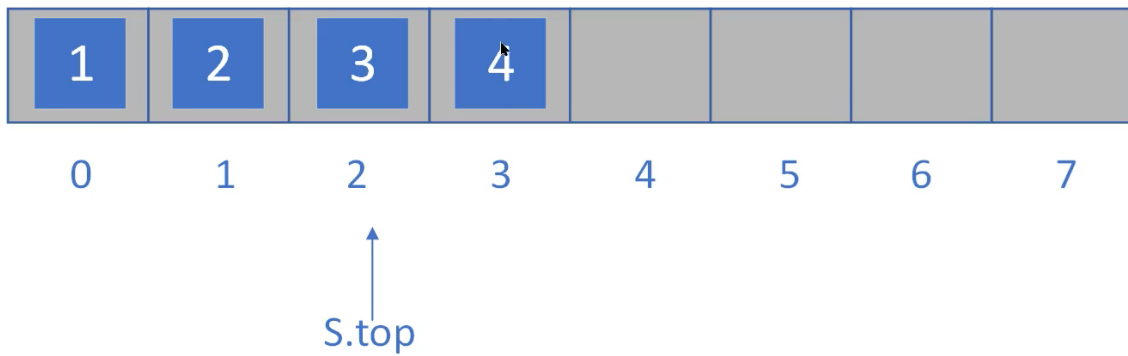
```
typedef struct{
    Elemtype data[50];
    int top;
}SqStack;
SqStack S;
```



`S.data[++S.top]=4;`

元素入栈

CSDN @QuantumYou



$x = S.data[S.top--]$

元素出栈

CSDN @QuantumYou

- 入栈：Top 栈顶指针先加加 ； 出栈：Top 栈顶指针后减减

栈的基本操作

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define MaxSize 50
5  typedef int ElemType;
6  typedef struct {
7      ElemType data[MaxSize]; //数组
8      int top;
9  } SqStack;
10
11 void InitStack(SqStack &S)
12 {
13     S.top = -1; //代表栈为空
14 }
15
16 bool StackEmpty(SqStack S)
17 {
18     if (-1 == S.top)
19     {
20         return true;
21     }
22     return false;
23 }
24 bool Push(SqStack& S, ElemType x)
25 {
26     if (S.top == MaxSize - 1)
27     {
28         return false; //栈满了
29     }
30     S.data[++S.top] = x;
31     return true; //返回true就是入栈成功
32 }
33 //获取栈顶元素
34 bool GetTop(SqStack S, ElemType &x)
35 {
36     if (StackEmpty(S)) //栈为空
```

```

37     {
38         return false;
39     }
40     x = S.data[S.top];
41     return true;
42 }
43 bool Pop(SqStack& S, ElemType& x)
44 {
45     if (StackEmpty(S))//栈为空
46     {
47         return false;
48     }
49     x = S.data[S.top--];//等价于x = S.data[S.top]; 再做    S.top--;
50     return true;
51 }
52 int main()
53 {
54     SqStack S;
55     bool flag;
56     ElemType m;//存储拿出来的栈顶元素的
57     InitStack(S);//初始化
58     flag = StackEmpty(S);
59     if (flag)
60     {
61         printf("栈是空的\n");
62     }
63     Push(S, 3);//入栈元素3
64     Push(S, 4);//入栈元素4
65     Push(S, 5);
66     flag = GetTop(S, m);//获取栈顶元素,但是S.top值不变
67     if (flag)
68     {
69         printf("获取栈顶元素为 %d\n", m);
70     }
71     flag = Pop(S, m);//弹出栈顶元素
72     if (flag)
73     {
74         printf("弹出元素为 %d\n", m);
75     }
76     return 0;
77 }

```

拓展知识：链表实现的栈是头部插入与头部删除

链式存储实现栈(相对不重要)

```
LiStack Lhead=(LiStack)malloc(sizeof(struct Linknode))
Lhead->next=NULL;
LiStack top=NULL;
```



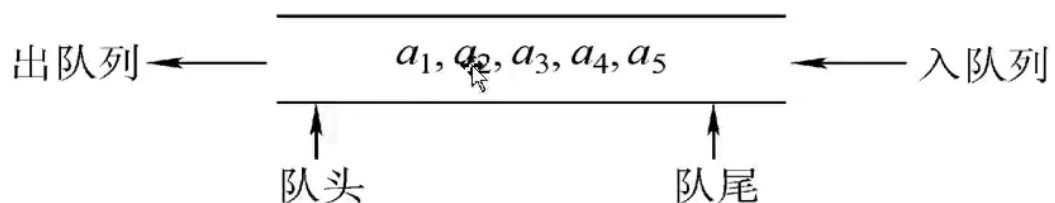
```
top=(LiStack)malloc(sizeof(struct Linknode));
top->next=NULL;
top->data=1;
top->next=Lhead->next;
Lhead->next=top;
```

CSDN @QuantumYou

队列

队头（Front）。允许删除的一端，又称队首。

队尾（Rear）。允许插入的一端。

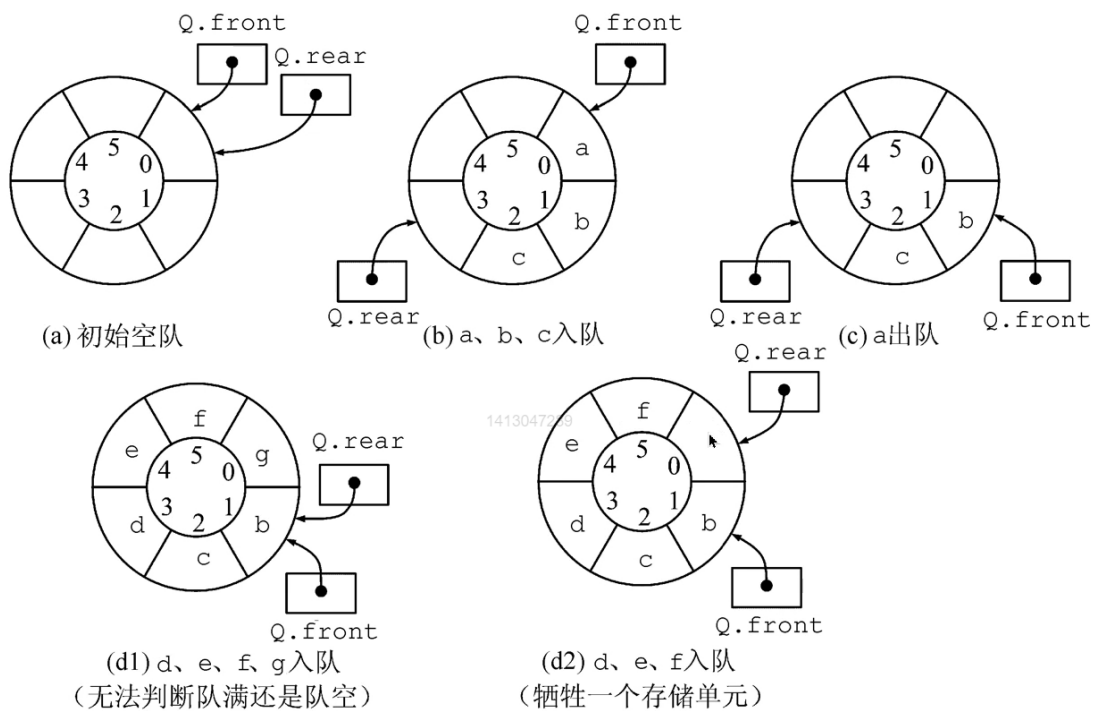


特性是先进先出（First In First Out, FIFO）

CSDN @QuantumYou

循环队列

循环队列图示



CSDN @QuantumYou

为了区分队空还是队满的情况:

- 1、以牺牲一个单元区分队空和队满
- 2、类型中增设表示数据元素个数的数据单元
- 3、类型中增设tag 数据成员

入队

```
bool EnQueue(SqQueue &Q, ElemType x)
{
    if((Q.rear+1)%MaxSize==Q.front) //判断是否队满
        return false;
    Q.data[Q.rear]=x; //放入元素
    Q.rear=(Q.rear+1)%MaxSize; //改变队尾标记
    return true;
}
```

CSDN @QuantumYou

出队

```
bool DeQueue(SqQueue &Q, ElemType &x)
{
    if(Q.rear==Q.front)
        return false;
    x=Q.data[Q.front]; //先进先出
    Q.front=(Q.front+1)%MaxSize;
    return true;
}
```



CSDN @QuantumYou

队列全局代码

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define MaxSize 5
5  typedef int ElemType;
6  typedef struct{
7      ElemType data[MaxSize]; //数组, 存储MaxSize-1个元素
8      int front, rear; //队列头 队列尾
9  }SqQueue;
10
11 void InitQueue(SqQueue &Q)
12 {
13     Q.rear=Q.front=0;
14 }
15 //判空
16 bool isEmpty(SqQueue &Q)
17 {
18     if(Q.rear==Q.front) //不需要为零
19         return true;
20     else
```

```

21         return false;
22     }
23     //入队
24     bool EnQueue(SqQueue &Q, ElemType x)
25     {
26         if((Q.rear+1)%MaxSize==Q.front) //判断是否队满
27             return false;
28         Q.data[Q.rear]=x; //3 4 5 6
29         Q.rear=(Q.rear+1)%MaxSize;
30         return true;
31     }
32     //出队
33     bool DeQueue(SqQueue &Q, ElemType &x)
34     {
35         if(Q.rear==Q.front)
36             return false;
37         x=Q.data[Q.front]; //先进先出
38         Q.front=(Q.front+1)%MaxSize;
39         return true;
40     }
41     //《王道C督学营》课程
42     //王道数据结构 3.2 循环队列
43     int main()
44     {
45         SqQueue Q;
46         bool ret; //存储返回值
47         ElemType element; //存储出队元素
48         InitQueue(Q);
49         ret=isEmpty(Q);
50         if(ret)
51         {
52             printf("队列为空\n");
53         }else{
54             printf("队列不为空\n");
55         }
56         EnQueue(Q, 3);
57         EnQueue(Q, 4);
58         EnQueue(Q, 5);
59         ret=EnQueue(Q, 6);
60         ret=EnQueue(Q, 7);
61         if(ret)
62         {
63             printf("入队成功\n");
64         }else{
65             printf("入队失败\n");
66         }
67         ret=DeQueue(Q, element);
68         if(ret)
69         {
70             printf("出队成功, 元素值为 %d\n", element);
71         }else{
72             printf("出队失败\n");
73         }
74         ret=DeQueue(Q, element);
75         if(ret)
76         {
77             printf("出队成功, 元素值为 %d\n", element);
78         }else{

```

```
79         printf("出队失败\n");
80     }
81     ret=EnQueue(Q,8);
82     if(ret)
83     {
84         printf("入队成功\n");
85     }else{
86         printf("入队失败\n");
87     }
88     system("pause");
89 }
```