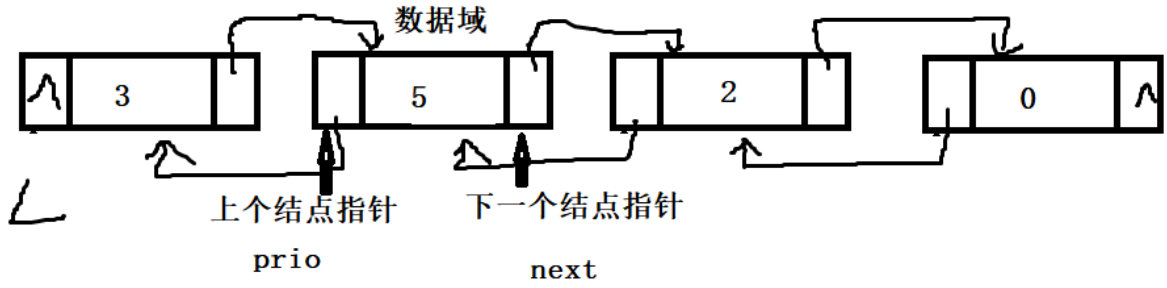
数据结构第三讲

一、双向链表

1> 因为单向链表只能通过前一个结点访问到后一个结点，不能通过后面的结点访问前驱节点，并且单链表最后一个节点指针域为空

2> 此时可以在指针域放两个指针，分别指向前驱节点和后继节点，那么通过前面的节点的next指针找到后继节点，也可以通过prio指针找到前驱节点



二、双向链表操作

2.1 节点结构体定义

除了数据域外，另外定义两个指针，分别指向前驱节点和后继节点

```
typedef int  datatype;
typedef struct Node
{
    union{
        int len;          //头节点数据域
        datatype data;    //普通节点数据域
    };

    struct Node *prio;          //指向前驱节点的指针
    struct Node *next;          //指向后继节点的指针
}doubleLink;
```

2.2 创建双向链表

创建成功后，需要将数据域len赋值为0，两个指针域全为空

```
doubleLink *list_create()
{
    doubleLink *D = (doubleLink*)malloc(sizeof(doubleLink));
```

```
    if(NULL==D)
    {
        printf("创建失败\n");
        return NULL;
    }

    //对节点初始化
    D->len = 0;
    D->prio = NULL;
    D->next = NULL;

    printf("创建成功\n");
    return D;            //将创建好的链表返回
}
```

2.3 头插法

1> 判断释放合法

2> 申请节点封装数据

3> 判断是否为空表，空表的话只需要链接两个指针域，非空要更新四个指针域

```
int list_insert_head(doubleLink *D, datatype e)
{
    //判断逻辑
    if(NULL == D)
    {
        printf("表不合法，插入失败\n");
        return -1;
    }

    //申请节点封装数据
    doubleLink *p = list_create();
    p->data = e;

    //插入逻辑
    if(D->next == NULL)
    {
        p->prio = D;
        D->next = p;
    }else
    {
        p->next = D->next;
        p->prio = D;
```

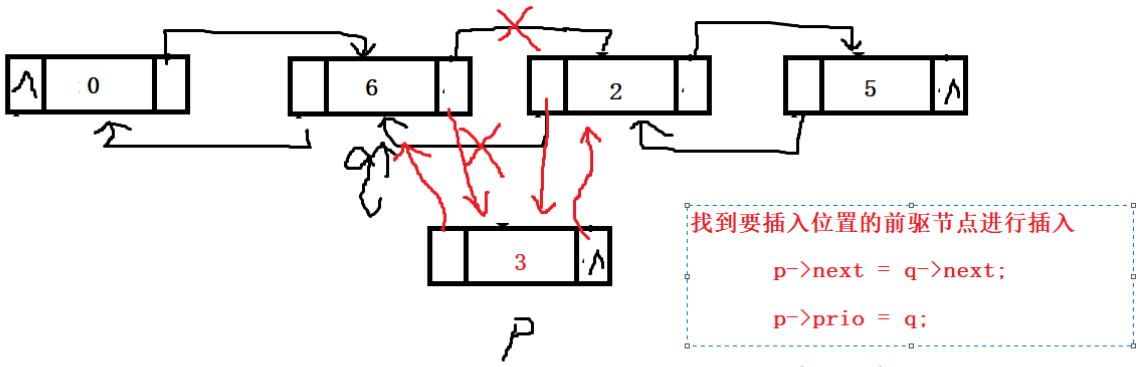
```
23
24          p->next->prio = p;
25          p->prio->next = p;
26      }
27
28      //表长变化
29      D->len++;
30      printf("插入成功\n");
31      return 0;
32  }
```

2.4 遍历

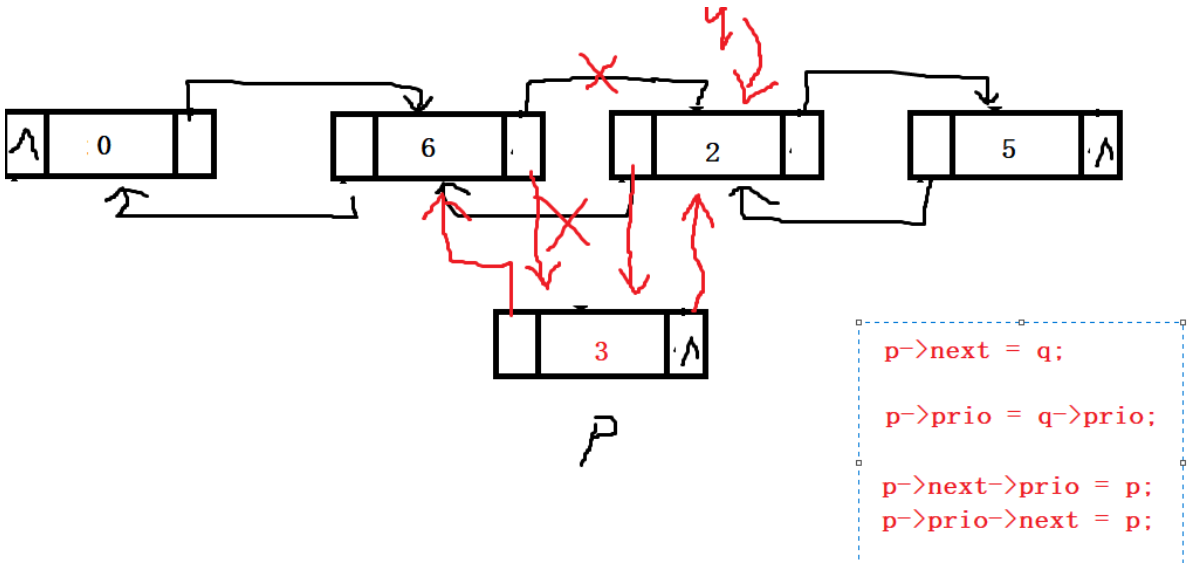
```
1   void list_show(doubleLink *D)
2   {
3       //判断逻辑
4       if(NULL==D || D->next==NULL)
5       {
6           printf("遍历失败\n");
7           return;
8       }
9
10      //定义遍历指针
11      printf("当前链表中的元素分别是: ");
12      doubleLink *q = D->next;
13      while(q!=NULL)
14      {
15          printf("%d\t", q->data);
16          q = q->next;
17      }
18      printf("\n");
19  }
```

2.5 任意位置插入函数

1> 可以找到要插入位置的前驱节点完成插入



2> 也可以找到要插入位置，直接进行插入




```c
int list_insert_pos(doubleLink *D, int pos, datatype e)
{
    //判断逻辑
    if(NULL==D || D->next==NULL || pos<1 || pos>D->len)
    {
        printf("插入失败\n");
        return -1;
    }

    //申请节点封装数据
    doubleLink *p = list_create();
    p->data = e;

    //找到当前位置
    doubleLink *q = D;
    for(int i=1; i<=pos; i++)
    {
        q = q->next;
    }
```

```
20
21      //插入逻辑
22      p->next = q;
23      p->prio = q->prio;
24      p->next->prio = p;
25      p->prio->next = p;
26
27      //表的变化
28      D->len++;
29      printf("插入成功\n");
30      return 0;
31  }
```

2.6 判空

```
1  int list_empty(doubleLink *D)
2  {
3      return D->next==NULL ? 1:0;      //判空   1表示空   0表示非空
4  }
```

2.7 任意位置删除函数

1> 判断逻辑：是否合法、是否为空
2> 定义遍历指针遍历到当前节点
3> 判断当前位置是否是最后一个节点
4> 如果是最后一个，则将倒数第二个节点的后继指针置空，再将该节点释放
5> 如果不是最后一个节点，那么将指向自己的节点指针进行更新

```
1  int list_delete_pos(doubleLink *D, int pos)
2  {
3      //判断逻辑
4      if(NULL==D || list_empty(D))
5      {
6          printf("删除失败\n");
7          return -1;
8      }
9
10     //定义遍历指针找到要删除的节点
11     doubleLink *p = D;
12     for(int i=1; i<=pos; i++)
13     {
14         p = p->next;
15     }
16
```

```
    //删除逻辑
    if(p->next == NULL)
    {
        p->prio->next = NULL;     //将倒数第二个节点后继指针置空
        free(p);
        p = NULL;
    }else
    {
        p->prio->next = p->next;
        p->next->prio = p->prio;

        free(p);
        p = NULL;
    }

    //表的变化
    D->len--;
    printf("删除成功\n");
    return 0;
}
```

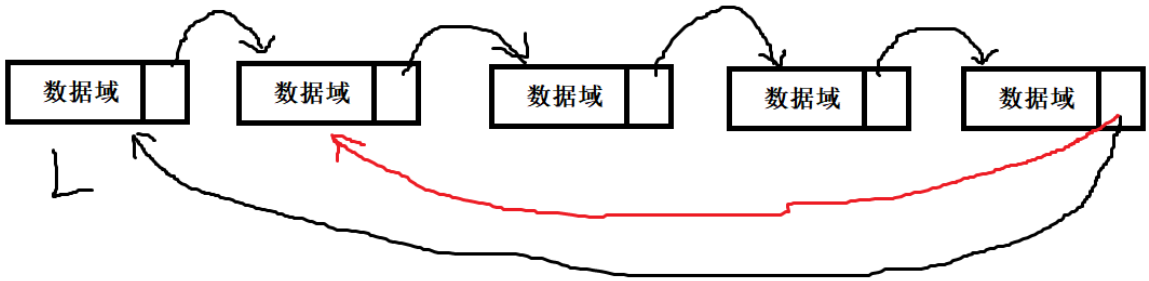
作业：双向链表尾插法、头删法、按位置查找返回节点、释放表

三、单向循环链表

1> 单向链表特点是，只能从前往后访问节点，一个节点一旦被访问后，就不能再次被访问了

2> 单向循环链表，整条链表访问到最后一个节点后，能回到第一个节点继续往后访问

3> 特点：单向循环链表的最后一个节点的指针域指向第一个节点
在循环链表中，没有一个节点的指针域是空的



3.1 节点结构体定义

```c
typedef char datatype;

typedef struct Node
{
    union{
        int len;     //头结点数据域
        datatype data;       //普通节点数据域
    };

    struct Node *next;     //指针域
}loopLink;
```

3.2 创建函数

创建成功后，头节点数据域为0，指针域指向自己

```c
loopLink *list_create()
{
    loopLink *L = (loopLink*)malloc(sizeof(loopLink));
    if(NULL==L)
    {
        printf("创建失败\n");
        return NULL;
    }

    //初始化
    L->len = 0;
    L->next = L;    //自己指向自己

    printf("创建成功\n");
    return L;
}
```

3.3 判空

判断头节点指针域是否指向自己

```c
int list_empty()
{
    return L->next==L ? 1: 0;     //1表示空   0表示非空
}
```

3.4 头插

```c
int list_insert_head(loopLink *L, datatype e)
{
    //判断逻辑
    if(NULL==L)
    {
        printf("所给链表不合法\n");
        return -1;
    }

    //申请节点封装数据
    loopLink *p = list_create();
    p->data = e;

    //插入逻辑
    p->next = L->next;
    L->next = p;

    //表的变化
    L->len++;
    printf("插入成功\n");
    return 0;
}
```

3.5 遍历

```c
void list_show(loopLink *L)
{
    //判断逻辑
    if(NULL==L || list_empty(L))
    {
        printf("遍历失败\n");
        return;
    }

    //遍历
    loopLink *q = L->next;
    while(q != L)
    {
        printf("%c\t", q->data);
        q = q->next;
    }
    printf("\n");
```

```
18
19 }
```

3.6 尾插

注意：找到最后一个节点后，将新节点连接到表尾，将新节点的指针域指向头节点

```c
int list_insert_tail(loopLink *L, datatype e)
{
    //判断逻辑
    if(NULL==L)
    {
        printf("所给链表不合法\n");
        return -1;
    }

    //申请节点封装数据
    loopLink *p = list_create();
    p->data = e;

    //定义遍历指针找到最后一个节点
    loopLink *q = L;
    while(q->next != L)
    {
        q = q->next;
    }

    //插入逻辑
    q->next = p;
    p->next = L;

    //表的变化
    L->len++;
    printf("插入成功\n");
    return 0;
}
```

3.7 尾删函数

```c
int list_delete_tail(loopLink *L)
{
    //判断逻辑
    if(NULL==L || list_empty(L))
```

```
    {
        printf("删除失败\n");
        return -1;
    }

    //定义遍历指针找到倒数第二个节点
    loopLink *q = L;
    while(q->next->next != L)
    {
        q = q->next;
    }

    //删除最后一个节点
    free(q->next);

    //将倒数第二个节点的指针指向头结点
    q->next = L;

    //表的变化
    L->len--;
    printf("删除成功\n");
    return 0;
}
```

3.8 删除头节点

```
loopLink *head_delete(loopLink *L)
{
    //判断逻辑
    if(NULL==L || list_empty(L))
    {
        printf("删除失败\n");
        return NULL;
    }

    //定义遍历指针找到最后一个节点
    loopLink *q = L;
    while(q->next != L)
    {
        q = q->next;
    }

    //将最后一个节点的指针域指向第一个节点
    q->next = L->next;
```

```
19
20      //将头结点释放
21      free(L);
22      L = NULL;
23
24      printf("头结点删除成功\n");
25
26      return q->next;     //将第一个节点地址返回
27  }
```

3.9 删除头节点后的遍历

```
1   void list_show_2(loopLink *L)
2   {
3       //判断逻辑
4       if(NULL==L)
5       {
6           printf("遍历失败\n");
7       }
8
9       //定义遍历指针将所有节点遍历一遍
10      printf("去头后的循环链表为：");
11      loopLink *q = L;
12
13      do
14      {
15          printf("%c\t", q->data);
16          q = q->next;
17      }while(q != L);
18
19      printf("\n");
20
21  }
```

四、栈

4.1 栈的概念

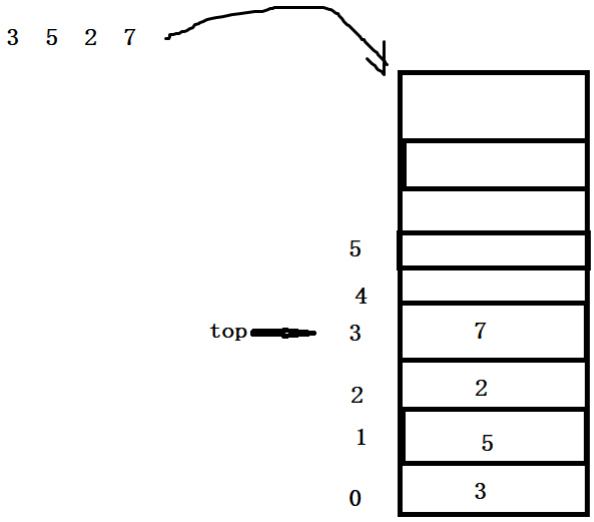
1> 栈是操作受限的线性表：只能在同一端进行插入和删除操作的线性表
2> 特点：先进后出（FILO）、后进先出（LIFO）
3> 允许操作的一端被称为栈顶，不能操作的一段称为栈底
4> 注意：区分数据结构中的栈，跟内存分配的栈不是一个概念
5> 存储结构有两种：
顺序栈：顺序存储的栈

链式栈：链式存储的栈

6>　基本操作：创建、判空、判满、入栈、出栈、遍历栈

4.2 顺序栈结构体类型

提供一个数组，存放栈；还要提供一个变量记录栈顶元素的下标




```
#define MAX 8
typedef int datatype;
typedef struct
{
    datatype data[MAX];          //存放栈的数组
    int top;                     //记录栈顶元素的下标
}seqStack;

```

作业：

作业1：作业：双向链表尾插法、头删法、按位置查找返回节点、释放表

作业2：循环链表实现约瑟夫环

■ **案例：** 用循环链表编程实现约瑟夫问题

◆n个人围成一圈，从某人开始报数1, 2, ..., m，数到m的人出圈，然后从出圈的下一个人(m+1)开始重复此过程，直到全部人出圈，于是得到一个出圈人员的新序列

◆如当n=8，m=4时，若从第一个位置数起，则所得到的新的序列为4, 8, 5, 2, 1, 3, 7, 6。