队列：

与栈一样，它属于一种逻辑， 队列的逻辑为： 先进先出， 后进后出。

插入一个新节点，必须插入到指定的一端，而删除一个已有节点，则必须在另一端进行。

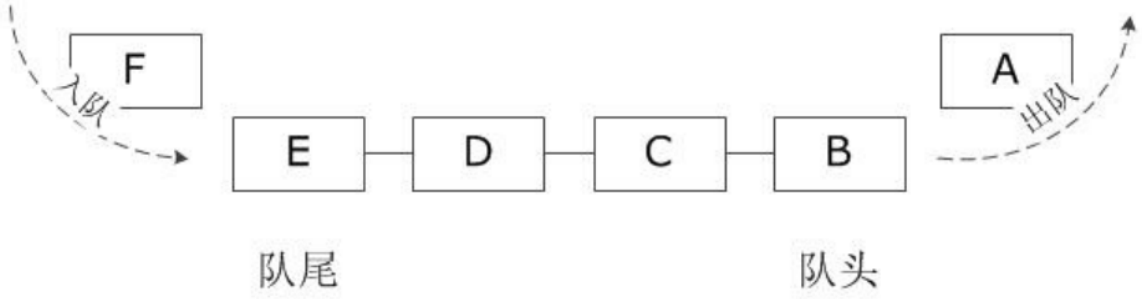


图 3-49 队列的逻辑

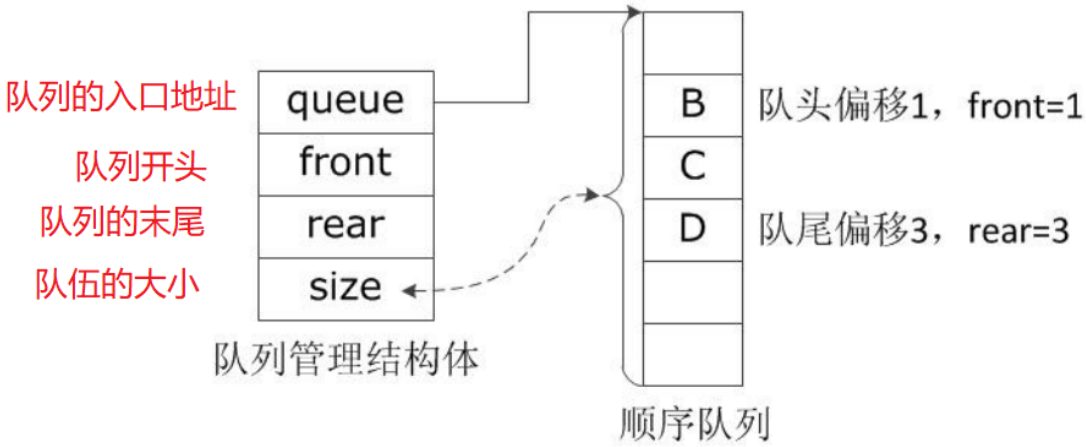
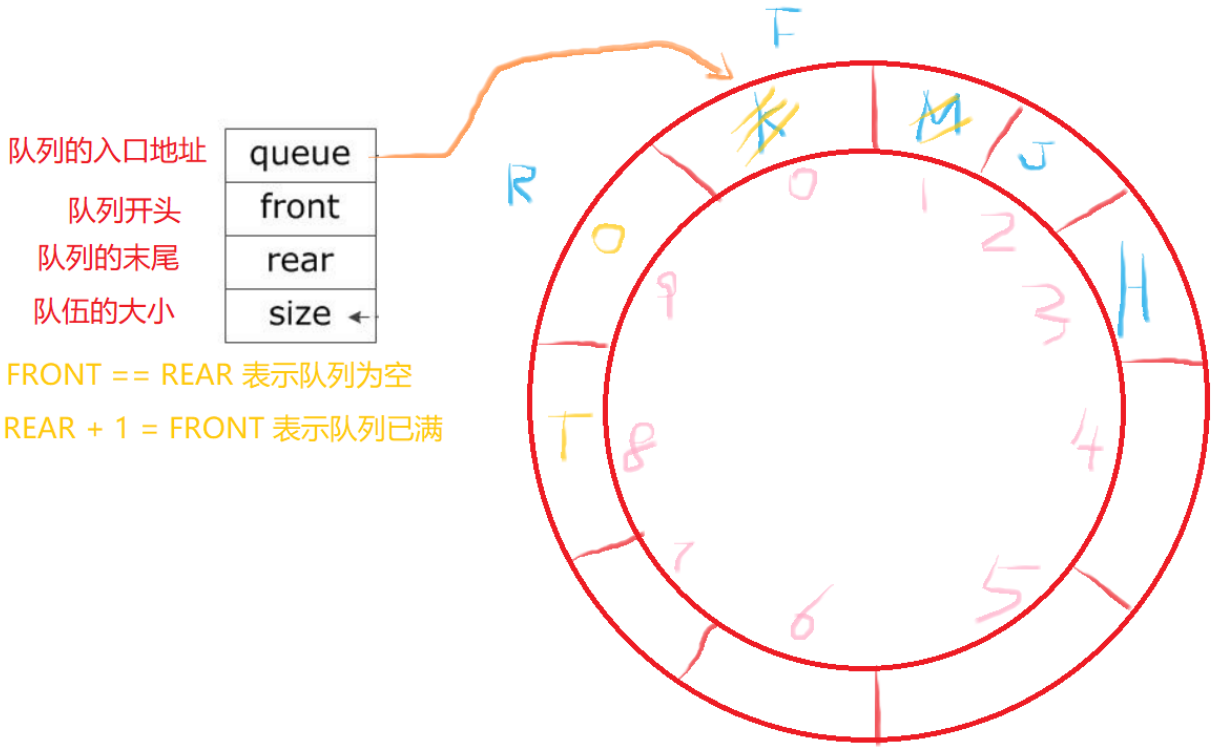


图 3-50 顺序队列



循环队列满、队列空状态如图 3-17 所示，其中阴影部分为已占用的空间。

队列满条件为：

((CQ.rear+1)% maxsize==CQ.front) 成立

队列空条件为：

(CQ.rear==CQ.front) 成立

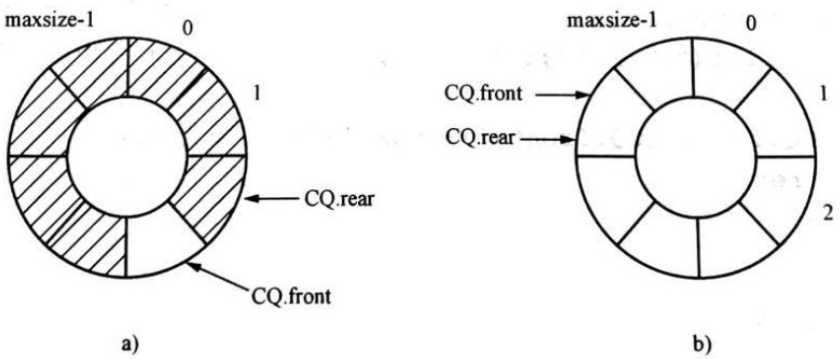


图 3-17　循环队列的队列满和队列空情况示意图

a) 队列满情况　b) 队列空情况

顺序队列

设计管理结构体：

```
typedef     int     Data_type;
```

```
2
3  //设计管理结构体
4  struct queue
5  {
6      Data_type * Enter;  // 队列入口地址
7      int size ;          // 队列总大小
8      int front ;         // 队列的头部（当前的队头 偏移量 / 数组的下标）
9      int rear ;          // 队列的尾部（当前的队尾 偏移量 / 数组的下标）
10 };
```

初始化：

```
1
2  P_Node queue_init(int queue_size)
3  {
4      // 申请管理结构体
5      P_Node queue = calloc(1, sizeof(Node));
6
7      // 申请一个队列空间并把入口地址赋值给 管理结构体中的入口成员
8      queue->Enter = calloc( queue_size , sizeof(Data_type) );
9
10     // 设置管理结构体中队列的大小
11     queue->size = queue_size ;
12
13     // 设置队头等于队尾 ，  表示空队
14     queue->front = queue->rear = 0 ;
15
16     return queue ;
17 }
```

入队：

```
1
2  bool add_2_queue( P_Node queue , Data_type data )
3  {
4      if(queue == NULL || queue->Enter == NULL ||
5              queue->front == (queue->rear+1)%queue->size )
6      {
```

```
        printf("队列异常，可能是满了！！！\n");
        return false ;
    }

    // 把数据存入到当前 队尾标记的下标位置
    * (queue->Enter + queue->rear ) = data ;

    // 队尾往后挪一个（下标）
    queue->rear ++ ;

    return true ;
}
```

出队：

```

Data_type exit_queue( P_Node queue )
{
    if( queue->front == queue->rear )
    {
        printf("当前队列为空， 没有任务！！！\n");
        return -1 ;
    }

    // 把队头数据赋值给临时变量
    Data_type tmp = *(queue->Enter+queue->front) ;

    // 把头加一， 往队尾移动
    queue->front ++ ;

    return tmp ;
}
```

主函数：

```

int main(int argc, char const *argv[])
{
```

```
    // 初始化
    P_Node queue = queue_init( 10 ) ;

    // 入队
    for (int i = 0; i < 11 ; i++)
    {
        add_2_queue( queue , i );
    }

    // 出队
    for (size_t i = 0; i < 11 ; i++)
    {
        Data_type tmp =  exit_queue(  queue );
        printf("data:%d\n" , tmp );
    }


    free(queue->Enter);
    free(queue);

    return 0;
}
```

链式队列

节点设计：

```

typedef     int     Data_Type ;
//节点设计
typedef struct list_queue
{
    Data_Type data ;
    struct list_queue * prev , * next ;
}Node , *P_Node;


```

初始化：

```
P_Node new_node_init( Data_Type data )
{
    P_Node new = calloc( 1, sizeof(Node)) ;
    new->data = data ;
    new->next = new->prev = new ;
    return new ;
}
```

入队：
使用尾插入队

```
int add_2_queue( P_Node head , Data_Type data )
{
    if( head == NULL )
    {
        printf("队列头异常！！\n");
        return -1 ;
    }
    P_Node new = new_node_init(  data );
    new->next = head ;
    new->prev = head->prev ;
    head->prev->next = new ;
    head->prev = new;
    return 0 ;
}
```

出队：

使用头出，进行出队操作

```
int exit_queue( P_Node head )
{
    if( head->next == head )
    {
        printf("当前队列为空！！\n");
        return -1 ;
    }

    P_Node tmp = head->next ;
    printf("队头数据为：%d\n", tmp->data) ;


    // 把第一个节点删除
    head->next = tmp->next ;
    tmp->next->prev = head ;

    tmp->prev = tmp->next = NULL ;
    free(tmp);// 释放该节点的堆空间内存
    tmp = NULL ;

    return 0 ;
}
```

作业：

1. 搞懂队列的操作（链式队列/顺序队列）
2. 使用双向循环链表实现类似菜单列表功能
    a. 用户点击上一个/下一个 可以显示上一个/下一个节点的内容

预习：

内核链表