队列

队列，又称为**伫列**（queue），是先进先出（FIFO, First-In-First-Out）的线性表。在具体应用中通常用链表或者数组来实现。队列只允许在后端（称为rear）进行插入操作，在前端（称为front）进行删除操作。
队列的操作方式和堆栈类似，唯一的区别在于队列只允许新数据在后端进行添加。

单链队列

单链队列使用链表作为基本数据结果，所以不存在伪溢出的问题，队列长度也没有限制。但插入和读取的时间代价较高

```
/* 单链队列——队列的链式存储结构 */
typedef struct QNode
{
  QElemType data;
  struct QNode *next;
}QNode,*QueuePtr;

typedef struct
{
  QueuePtr front,rear; /* 队头、队尾指针 */
}LinkQueue;

/* 链队列的基本操作(9个) */
void InitQueue(LinkQueue *Q)
{ /* 构造一个空队列Q */
  Q->front=Q->rear=malloc(sizeof(QNode));
  if(!Q->front)
    exit(OVERFLOW);
  Q->front->next=NULL;
}

void DestroyQueue(LinkQueue *Q)
{ /* 销毁队列Q(无论空否均可) */
  while(Q->front)
  {
```

```
    Q->rear=Q->front->next;
    free(Q->front);
    Q->front=Q->rear;
  }
}

void ClearQueue(LinkQueue *Q)
{ /* 将Q清为空队列 */
  QueuePtr p,q;
  Q->rear=Q->front;
  p=Q->front->next;
  Q->front->next=NULL;
  while(p)
  {
    q=p;
    p=p->next;
    free(q);
  }
}

Status QueueEmpty(LinkQueue Q)
{ /* 若Q为空队列，则返回TRUE，否则返回FALSE */
  if(Q.front->next==NULL)
    return TRUE;
  else
    return FALSE;
}

int QueueLength(LinkQueue Q)
{ /* 求队列的长度 */
  int i=0;
  QueuePtr p;
  p=Q.front;
  while(Q.rear!=p)
  {
    i++;
    p=p->next;
  }
  return i;
}
```

```
Status GetHead_Q(LinkQueue Q,QElemType *e)
{ /* 若队列不空，则用e返回Q的队头元素，并返回OK，否则返回ERROR */
  QueuePtr p;
  if(Q.front==Q.rear)
    return ERROR;
  p=Q.front->next;
  *e=p->data;
  return OK;
}

void EnQueue(LinkQueue *Q,QElemType e)
{ /* 插入元素e为Q的新的队尾元素 */
  QueuePtr p= (QueuePtr)malloc(sizeof(QNode));
  if(!p) /* 存储分配失败 */
    exit(OVERFLOW);
  p->data=e;
  p->next=NULL;
  Q->rear->next=p;
  Q->rear=p;
}

Status DeQueue(LinkQueue *Q,QElemType *e)
{ /* 若队列不空，删除Q的队头元素，用e返回其值，并返回OK，否则返回ERROR
*/
  QueuePtr p;
  if(Q->front==Q->rear)
    return ERROR;
  p=Q->front->next; /* 指向头结点 */
  *e=p->data;
  Q->front=p->next; /* 摘下头节点 */
  if(Q->rear==p)
    Q->rear=Q->front;
  free(p);
  return OK;
}

void QueueTraverse(LinkQueue Q,void(*vi)(QElemType))
{ /* 从队头到队尾依次对队列Q中每个元素调用函数vi() */
  QueuePtr p;
```

```
  p=Q.front->next;
  while(p)
  {
    vi(p->data);
    p=p->next;
  }
  printf("\n");
}
```

循环队列

循环队列可以更简单防止伪溢出的发生，但队列大小是固定的。

```
/* 队列的顺序存储结构(循环队列) */
#define MAX_QSIZE 5 /* 最大队列长度+1 */
typedef struct
{
  QElemType *base; /* 初始化的动态分配存储空间 */
  int front; /* 头指针，若队列不空，指向队列头元素 */
  int rear; /* 尾指针，若队列不空，指向队列尾元素的下一个位置 */
}SqQueue;

/* 循环队列的基本操作(9个) */
void InitQueue(SqQueue *Q)
{ /* 构造一个空队列Q */
  Q->base=malloc(MAX_QSIZE*sizeof(QElemType));
  if(!Q->base) /* 存储分配失败 */
    exit(OVERFLOW);
  Q->front=Q->rear=0;
}

void DestroyQueue(SqQueue *Q)
{ /* 销毁队列Q，Q不再存在 */
  if(Q->base)
    free(Q->base);
  Q->base=NULL;
  Q->front=Q->rear=0;
}

void ClearQueue(SqQueue *Q)
```

```
{ /* 将Q清为空队列 */
  Q->front=Q->rear=0;
}

Status QueueEmpty(SqQueue Q)
{ /* 若队列Q为空队列，则返回TRUE；否则返回FALSE */
  if(Q.front==Q.rear) /* 队列空的标志 */
    return TRUE;
  else
    return FALSE;
}

int QueueLength(SqQueue Q)
{ /* 返回Q的元素个数，即队列的长度 */
  return(Q.rear-Q.front+MAX_QSIZE)%MAX_QSIZE;
}

Status GetHead(SqQueue Q,QElemType *e)
{ /* 若队列不空，则用e返回Q的队头元素，并返回OK；否则返回ERROR */
  if(Q.front==Q.rear) /* 队列空 */
    return ERROR;
  *e=Q.base[Q.front];
  return OK;
}

Status EnQueue(SqQueue *Q,QElemType e)
{ /* 插入元素e为Q的新的队尾元素 */
  if((Q->rear+1)%MAX_QSIZE==Q->front) /* 队列满 */
    return ERROR;
  Q->base[Q->rear]=e;
  Q->rear=(Q->rear+1)%MAX_QSIZE;
  return OK;
}

Status DeQueue(SqQueue *Q,QElemType *e)
{ /* 若队列不空，则删除Q的队头元素，用e返回其值，并返回OK；否则返回
ERROR */
  if(Q->front==Q->rear) /* 队列空 */
    return ERROR;
  *e=Q->base[Q->front];
```

```
  Q->front=(Q->front+1)%MAX_QSIZE;
  return OK;
}

void QueueTraverse(SqQueue Q,void(*vi)(QElemType))
{ /* 从队头到队尾依次对队列Q中每个元素调用函数vi() */
  int i;
  i=Q.front;
  while(i!=Q.rear)
  {
    vi(Q.base[i]);
    i=(i+1)%MAX_QSIZE;
  }
  printf("\n");
}
```

阵列队列

```
#include<stdio.h>
#include<stdlib.h>
/*佇列資料結構*/
struct Queue
{
  int Array[10];//陣列空間大小
  int head;//前端(front)
  int tail;//後端(rear)
  int length;//佇列長度 //將其視為當前資料大小，並且使用這個成員判斷滿或空
};
/*資料加入佇列*/
void EnQueue(struct Queue *Queue1,int x)
{
  Queue1->Array[Queue1->tail]=x;
  if(Queue1->tail+1==10)//Queue1->length改為空間大小10
  {
    Queue1->tail=0;//1改為0
  }
  else
  {
    Queue1->tail=Queue1->tail+1;
```

```
    //Queue1->length=Queue1->length+1;//這行邏輯上有問題   //Modify By pcjackal.tw //這行放於外面
  }
  Queue1->length=Queue1->length+1;//當前個數增1
}
/*資料移出佇列*/
int DeQueue(struct Queue *Queue1)
{
  int x=Queue1->Array[Queue1->head];
  if(Queue1->head+1==10)//空間大小10
  {
    Queue1->head=0;
  }
  else
  {
    Queue1->head=Queue1->head+1;
  }
  Queue1->length=Queue1->length-1;//移出后減少1
  return x;
}
/*佇列操作*/
int main()
{
  struct Queue *Queue1=malloc(sizeof(struct Queue));//建立資料結構
  Queue1->length=0;//新增長度//10改為0，初始狀態
  Queue1->head=0;//必須要先初始化
  Queue1->tail=0;//必須要先初始化
  EnQueue(Queue1,5);//將5放入佇列
  EnQueue(Queue1,8);//將8放入佇列
  EnQueue(Queue1,3);//將3放入佇列
  EnQueue(Queue1,2);//將2放入佇列
  printf("%d ",DeQueue(Queue1));//輸出佇列(5)
  printf("%d ",DeQueue(Queue1));//輸出佇列(8)
  printf("%d ",DeQueue(Queue1));//輸出佇列(3)
  system("pause");
}
```