#include<stdio.h>

#include <stdlib.h>

#pragma mark - 顺序栈

typedef struct

{

    char \*elem;

    int top;

    int size;

    int increment;

}SqStack;

//顺序栈的初始化

int InitStack\_Sq(SqStack \*S,int size,int inc)

{

    S->elem = (char \*)malloc(size \*sizeof(char));

    if (S->elem == NULL)

    {

        return 0;

    }

    S->top = 0;

    S->size = size;

    S->increment = inc;

    return 1;

}

//顺序栈入栈

int Push\_Sq(SqStack \*S, char e)

{

    char \*temp;

    if (S->top>S->size)

    {

        temp = (char \*)realloc(S->elem,(S->size+S->increment)\*sizeof(char));

        if (temp == NULL)

        {

            return 0;

        }

        S->elem = temp;

        S->size += S->increment;

    }

    S->elem[S->top++] = e;

    return 1;

}

//销毁顺序栈

int DestroyStack\_Sq(SqStack \*s)

{

    free(s->elem);

    s->elem = NULL;

    s->size = 0;

    s->top = 0;

    s->increment = 0;

    return 1;

}

//判断栈是否为空

int StackEmpty(SqStack \*s)

{

    if (s->top == 0)

    {

        return 1;

    }

    return 0;

}

//清空栈

int ClearStack\_Sq(SqStack \*s)

{

    s->top = 0;

    return 1;

}

//将栈顶元素出栈

char Pop\_Sq(SqStack \*s)

{

    if (s->top == 0)

    {

        return 0;

    }

    s->top = s->top-1;

    return s->elem[s->top];

}

//取出栈顶元素

char GetTop\_Sq(SqStack \*s)

{

    if (s->top == 0)

    {

        return 'n';

    }

    return s->elem[s->top-1];

}

#pragma mark - 循环队列

typedef struct

{

    char \*elem;

    int front;

    int rear;

    int maxSize;

}SqQueue;

//循环队列初始化

int InitQueue\_Sq(SqQueue \*Q,int size)

{

    Q->elem = (char \*)malloc(size \* sizeof(char));

    if (Q->elem == NULL)

    {

        return 0;

    }

    Q->maxSize = size;

    Q->front = Q->rear = 0;

    return 1;

}

//销毁队列

void DestroyQueue\_Sq(SqQueue \*Q)

{

    free(Q->elem);

    Q->elem = NULL;

    Q->front = Q->rear = 0;

    Q->maxSize = 0;

}

//将队列置为空

void ClearQueue\_Sq(SqQueue \*Q)

{

    Q->front = Q->rear = 0;

}

//判断队列是否为空队列

int QueueEmpty\_Sq(SqQueue \*Q)

{

    if (Q->front == Q->rear)

    {

        return 1;

    }

    return 0;

}

//返回队列元素个数

int QueueLength\_Sq(SqQueue \*Q)

{

    return (Q->rear - Q->front + Q->maxSize) % Q->maxSize;

}

//返回队列头元素

char GetHead\_Sq(SqQueue \*Q)

{

    if (Q->front == Q->rear)

    {

        return '0';

    }

    return Q->elem[Q->front];

}

//队尾插入元素

int EnQueue\_Sq(SqQueue \*Q,char a)

{

    if ((Q->rear+1)%Q->maxSize == Q->front)

    {

        return 0;

    }

    Q->elem[Q->rear] = a;

    Q->rear = (Q->rear+1)%Q->maxSize;

    return 1;

}

//删除队列的头元素

char DeQueue\_Sq(SqQueue \*Q)

{

    if (Q->front == Q->rear)

    {

        return '0';

    }

    int temp = Q->front;

    Q->front = (Q->front+1)%Q->maxSize;

    return Q->elem[temp];

}

#pragma mark - 链栈

typedef struct LSNode

{

    char data;

    struct LSNode \*next;

}LSNode,\*LStack;

//初始化链栈

LStack InitStack\_LS()

{

    LStack s = (LStack)malloc(sizeof(LSNode));

    if (s==NULL)

    {

        printf("初始化失败\n");

    }else

    {

        s->next = NULL;

        printf("初始化成功\n");

        return s;

    }

    return s;

}

//清空栈

void ClearStack\_LS(LStack s)

{

    LStack t = s->next;

    while (t)

    {

        s->next = t->next;

        free(t);

        t->next = NULL;

        t = s->next;

    }

}

//判断空栈

int StackEmpty\_LS(LStack s)

{

    if (s->next == NULL)

    {

        return 1;

    }

    return 0;

}

//销毁链栈

void DestroyStack\_LS(LStack s)

{

    if (StackEmpty\_LS(s) == 1)

    {

         free(s);

         s->next = NULL;

    }else

    {

        ClearStack\_LS(s);

        free(s);

        s->next = NULL;

    }

}

//元素入栈

void Puah\_LS(LStack s,char e)

{

    if (s != NULL)

    {

        LStack t = (LStack)malloc(sizeof(LSNode));

        t->data = e;

        t->next = s->next;

        s->next = t;

    }

}

//栈顶元素出栈

char Pop\_LS(LStack s)

{

    char e;

    if (s->next)

    {

        LStack l = s->next;

        e = l->data;

        s->next = l->next;

    }

    return e;

}

//取栈顶元素

char GetTop\_LS(LStack s)

{

    if (s->next)

    {

        return s->next->data;

    }

    return '0';

}

#pragma mark - 链队列

typedef struct LQNode

{

    char data;

    struct  LQNode \*next;

}LQNode, \*QueuePtr;

typedef struct

{

    QueuePtr front;

    QueuePtr rear;

}LQueue;

//初始化空队列

void InitQueue\_LQ(LQueue \*q)

{

    q->front = q->rear = (QueuePtr)malloc(sizeof(LQNode));

    q->front = NULL;

}

//队列判空

int QueueEmpty\_LQ(LQueue \*q)

{

    if (q->front == NULL)

    {

        return 1;

    }else

    {

        return 0;

    }

}

//销毁队列

void DestroyQueue\_LQ(LQueue \*q)

{

    if (QueueEmpty\_LQ(q) == 0)

    {

        if (q->front == q->rear)

        {

            free(q);

            q->front = q->rear = NULL;

        }else

        {

            while (q->front != NULL)

            {

                free(q->front);

                q->front = q->front->next;

            }

        }

    }

}

//队列长度

int QueueLength\_LQ(LQueue \*q)

{

    int k=0;

    QueuePtr p;

    p = (QueuePtr)malloc(sizeof(LQNode));

    p = q->front;

    if (p == NULL)

    {

        return 0;

    }

    while (p != NULL)

    {

        k++;

        p = p->next;

    }

    return k;

}

//返队列头元素

char GetHead\_LQ(LQueue \*q)

{

    if (q->front != NULL)

    {

        return q->front->data;

    }

    return '0';

}

//队尾插入元素

void EnQueue\_LQ(LQueue \*q,char e)

{

    LQNode \*p;

    p =(LQNode \*)malloc(sizeof(LQNode));

    if (p != NULL)

    {

        p->data = e;

        p->next = NULL;

        if (q->front == NULL)

        {

            q->front = p;

        }else

        {

            q->rear->next = p;

        }

        q->rear = p;

    }

}

//删除队头元素

void DeQueue\_LQ(LQueue \*q)

{

    LQNode \*p;

    char e;

    if (q->front != NULL)

    {

        p = q->front;

        e = p->data;

        q->front = p->next;

        if (q->rear == p)

        {

            q->rear = NULL;

        }

        free(p);

    }

}

int main() {

// #pragma 顺序栈

//     SqStack \*s;

//     //初始化栈

//     InitStack\_Sq(s,10,6);

//     //元素压入栈

//     Push\_Sq(s,'a');

//     Push\_Sq(s,'c');

//     printf("%c压入栈\n", s->elem[0]);

//     printf("%c压入栈\n", s->elem[1]);

//     printf("栈顶位标%d\n",s->top);

//     //判断栈空

//     if (StackEmpty(s) == 1)

//     {

//         printf("栈为空\n");

//     }else {

//         printf("栈不为空\n");

//     }

//     //栈顶元素出栈

//     printf("栈顶元素%c出栈\n",Pop\_Sq(s));

//     //取出栈顶元素

//      printf("取出栈顶元素%c\n",GetTop\_Sq(s));

//     //清空栈

//     if (ClearStack\_Sq(s) == 1)

//     {

//         printf("清空栈，当前栈顶位标为%d\n",s->top);

//     }

//      //销毁栈

//     if (DestroyStack\_Sq(s) == 1)

//     {

//         printf("销毁栈\n");

//     }

//     //判断栈空

//     if (StackEmpty(s) == 1)

//     {

//         printf("栈为空\n");

//     }else {

//         printf("栈不为空\n");

//     }

// #pragma 循环队列

//     //初始化循环队列

//     SqQueue \*Q;

//     InitQueue\_Sq(Q,10);

//     //队尾插入元素

//     EnQueue\_Sq(Q,'a');

//     EnQueue\_Sq(Q,'c');

//     printf("队尾插入元素%c\n",Q->elem[0]);

//     printf("队尾插入元素%c\n",Q->elem[1]);

//     //返回队列元素个数

//     printf("队列元素为%d个\n",QueueLength\_Sq(Q));

//     //返回队列头元素

//     printf("队头元素为%c\n",GetHead\_Sq(Q));

//     //判断队列是否为空

//     if (QueueEmpty\_Sq(Q) == 1)

//     {

//         printf("队列为空\n");

//     }else

//     {

//         printf("队列不为空\n");

//     }

//     //删除队列的头元素

//     printf("删除队头元素%c\n",DeQueue\_Sq(Q));

//     printf("删除队头元素%c\n",GetHead\_Sq(Q));

//     //将队列置为空

//     ClearQueue\_Sq(Q);

//     if (QueueEmpty\_Sq(Q) == 1)

//     {

//         printf("队列为空\n");

//     }else

//     {

//         printf("队列不为空\n");

//     }

//     //销毁队列

//      DestroyQueue\_Sq(Q);

// #pragma 链栈

//     //初始化链栈

//     LStack l = InitStack\_LS();

//     //链栈判空

//     if (StackEmpty\_LS(l) == 1)

//     {

//         printf("栈为空\n");

//     }else

//     {

//         printf("栈不空\n");

//     }

//     //元素入栈

//     Puah\_LS(l,'a');

//     Puah\_LS(l,'c');

//     printf("元素ac入栈\n");

//     //取栈顶元素

//     printf("取栈顶元素%c\n",GetTop\_LS(l));

//     //栈顶元素出栈

//      printf("栈顶元素%c出栈\n",Pop\_LS(l));

//      //取栈顶元素

//     printf("再取栈顶元素%c\n",GetTop\_LS(l));

//     //清空栈

//     ClearStack\_LS(l);

//     //销毁栈

//     DestroyStack\_LS(l);

#pragma 链队列

    //初始化链队列

    LQueue \*q;

    InitQueue\_LQ(q);

    //队尾插入元素

    EnQueue\_LQ(q,'a');

    EnQueue\_LQ(q,'c');

    printf("插入元素ac\n");

    //返队列头元素

    printf("队头元素%c\n",GetHead\_LQ(q));

    //队列判空

    if (QueueEmpty\_LQ(q) == 1)

    {

        printf("队列为空\n");

    }else

    {

        printf("队列不为空\n");

    }

    //删除队头元素

    DeQueue\_LQ(q);

    printf("删除队头元素%c\n",GetHead\_LQ(q));

    //队列长度

    printf("队长%d\n",QueueLength\_LQ(q));

    //销毁队列

    DestroyQueue\_LQ(q);

    return 0;

}