顺序栈

假设Q[1…10]是一个顺序队列，初始状态为front=rear=0，A…Z入队，并求出元素个数。

#include <stdio.h>

#include <malloc.h>

#define MAXLEN 100

#define ElemType char

typedef struct

{

ElemType data[MAXLEN];

int front;

int rear;

int flag; //每进行一次出队运算时，标示量flag设为0

}SeqQueue;

//循环队列置空

SeqQueue \* initQueue(SeqQueue \* q)

{

q->front = 0;

q->rear = 0;

return q;

}

//建一个空循环队列

SeqQueue \* setQueue()

{

SeqQueue \* Q = (SeqQueue \*)malloc(sizeof(SeqQueue));

Q->front = 0;

Q->rear = 0;

Q->flag = 0;

return Q;

}

//判队满

int QueueFull(SeqQueue \* q)

{

if ((q->front == (q->rear)%MAXLEN) && q->flag ==1)

return 1;

else

return 0;

}

//判队空

int QueueEmpty(SeqQueue \*q)

{

if ((q->front == (q->rear)%MAXLEN) && q->flag == 0)

return 1;

else

return 0;

}

//入队

void push(SeqQueue \* q, ElemType ch)

{

if (QueueFull(q))

printf("queue full");

else

{

q->flag = 1;

q->data[q->rear % MAXLEN] = ch;

q->rear = (q->rear + 1) % MAXLEN;

}

}

//出队

void pop(SeqQueue \* q)

{

if (QueueEmpty(q))

printf("queue empty");

else

{

q->front = (q->front + 1) % MAXLEN;

q->flag = 0;

}

}

int getQueueNum(SeqQueue \* q)

{

int num = q->rear - q->front;

return num;

}

//输出队中有效元素

void showQueue(SeqQueue \* q)

{

int i;

for (i = q->front; i < q->rear; ++i)

printf("%c ", q->data[i]);

printf("\n");

}

int main(void)

{

SeqQueue \* Q = setQueue();

int i = 0;

ElemType ch = 'A';

for (i = 0; i < 26; ++i) //队里放26个字母

{

push(Q, ch++);

}

//pop(Q);

//printf("%s\n", Q->data);

showQueue(Q);

printf("%d\n", getQueueNum(Q));

return 0;

}

链栈

用ABCDE入栈、DE出栈测试数据。

#include <stdio.h>

#include <malloc.h>

#define ElemType char

typedef struct node

{

ElemType data;

struct node \* next;

}LinkStack;

//链栈键空栈

LinkStack \* setStack()

{

LinkStack \* LS = NULL;

return LS;

}

//链栈判栈空

int StackEmpty(LinkStack \* LS)

{

if(LS == NULL)

return 1;

else

return 0;

}

//链栈取栈顶元素

ElemType getTop(LinkStack \* LS)

{

if(LS == NULL)

return 0;

else

return LS->data; //用头插法插入元素，所以头结点就是栈顶

}

//链栈入栈

LinkStack\* Push(LinkStack \* LS, ElemType ch)

{

LinkStack \* p;

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

p->data = ch;

p->next = LS;

LS = p;

return LS;

}

//链栈出栈

LinkStack \* Pop(LinkStack \* LS)

{

LinkStack \* p = LS;

LS = LS->next;

free(p);

return LS;

}

//链栈置空

LinkStack \* InitStrack(LinkStack \* LS)

{

while (LS != NULL)

{

LS = Pop(LS);

}

return LS;

}

//输出链栈所有元素

void showStack(LinkStack \* LS)

{

LinkStack \* p = LS;

while(p != NULL)

{

printf("%c\t", getTop(p));

p = p->next;

}

printf("\n");

}

int main(void)

{

LinkStack \* LS = setStack();

LS = Push(LS, 'A');

LS = Push(LS, 'B');

LS = Push(LS, 'C');

LS = Push(LS, 'D');

LS = Push(LS, 'E');

showStack(LS);

LS = Pop(LS); //E出栈

showStack(LS);

LS = Pop(LS); //D出栈

showStack(LS);

return 0;

}