1. 栈
   1. 栈的相关定义
   2. 顺序栈
      1. 进制转换Bin2Oct.c Bin2Dec.c
      2. 逆波兰计算器 RPN.c
      3. 中缀表达式转后缀表达式 z2h.c
   3. 链式栈
2. 队列
   1. 链式队列的应用queue.c

## 【栈的相关定义】

1栈（stack）: 先进后出( FILO）的线性表。

或后进先出（ LIFO）的线性表。

或仅在表尾进行插入和删除操作的线性表。

2.栈顶（top）: 线性表的表尾端，即可操作端。

3.栈底（bottom）: 线性表的表头。

## 【顺序栈】

#define STACK\_INIT\_SIZE 100

#define STACKINCREMENT 10

#define OK 1

#define OVERFLOW -1

#define ERROR 0

typedef struct{

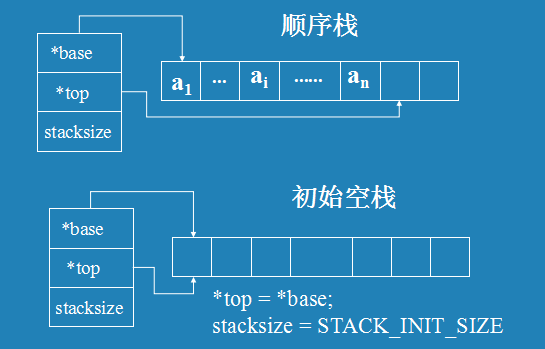
SElemType \*base;

/\* 在栈构造之前和销毁之后，base的值为NULL\*/

SElemType \*top; /\* 栈顶指针 \*/

int stacksize; /\* 当前已分配的存储空间，以元素为单位\*/

}SqStack;



顺序栈的示意图

#define STACK\_INIT\_SIZE 20

#define STACKINCREMENT 10

#define OK 1

#define ERROR 0

#define SOVERFLOW -1

typedef int Status;

typedef char ElemType;

typedef struct

{

ElemType \*base;

ElemType \*top;

int stacksize;

}sqStack;

Status InitStack(sqStack \*s)

{

s->base = (ElemType \*)malloc(STACK\_INIT\_SIZE \* sizeof(ElemType));

if(!s->base)return ERROR;

s->top = s->base;

s->stacksize = STACK\_INIT\_SIZE;

return OK;

}

Status Push(sqStack \*s,ElemType e)

{

if(s->top - s->base >= s->stacksize)

{

s->base = (ElemType \*)realloc(s->base,(s->stacksize + STACKINCREMENT)\*sizeof(ElemType));

if(!s->base)return SOVERFLOW;

s->top = s->base + s->stacksize;

s->stacksize = s->stacksize + STACKINCREMENT;

}

\*(s->top++)=e;

return OK;

}

Status Pop(sqStack \*s,ElemType \*e)

{

if(s->top == s->base)return ERROR;

\*e = \*(--s->top);

return OK;

}

int StackLen(sqStack s)

{

return(s.top - s.base);

}

## 【链式栈】

typedef struct StackNode

{

ElemType data;

struct StackNode \*next

}StackNode,\*LinkStackPtr;

typedef struct LinkStack

{

LinkStackPrt top;

int count;

}

Status Push(LinkStack \*s,ElemType e)

{

LinkStackPtr p = (LinkStackPtr) malloc(sizeof(StackNode));

p->data = e;

p->next = s->top;

s->top = p;

s->count++;

return OK;

}

Status Pop(LinkStack \*s,ElemType \*e)

{

LinkStackPtr p;

if(StackEmpty(\*s))return ERROR;

\*e = s->top->data;

p = s->top;

s-top = s->top->next;

free(p);

s->count--;

return OK;

}

int StackLen(sqStack s)

{

return ()

}

【队列的定义】

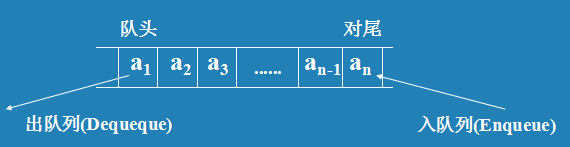
队列（Queue）: 先进先出(First In First Out)

(缩写为FIFO）的线性表。

仅在队尾进行插入和队头进行删除操作的线性表。

队头（front）: 线性表的表头端，即可删除端。

队尾（rear）: 线性表的表尾端，即可插入端。



typedef char ElemType;

typedef int Status;

typedef struct QNode

{

ElemType data;

struct QNode \*next;

}QNode,\*QueuePtr;

typedef struct

{

QueuePtr front,rear;

}LinkQueue;

Status InitQueue(LinkQueue \*q)

{

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

if(!q->front)return ERROR;

q->front->next=NULL;

return OK;

}

Status InsertQueue(LinkQueue \*q,ElemType e)

{

QueuePtr p;

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

if(p==NULL)return ERROR;

p->data=e;

p->next=NULL;

q->rear->next=p;

q->rear = p;

return OK;

}

Status DeleteQueue(LinkQueue \*q,ElemType \*e)

{

QueuePtr p;

if(q->front == q->rear)return ;

p=q->front->next;

\*e = p->data;

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

if(q->rear==p) q->rear = q->front;

free(p);

return OK;

}

Status DestroyQueue(LinkQueue \*q)

{

while(q->front)

{

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

free(q->front);

q->front = q->rear;

}

return OK;

}