```
#include<stdio.h>
#include<stdlib.h>

#define ElemType int


//---------------------------------------------顺序栈的定义-----------------------------
#define STACK_MAXSIZE 100
typedef struct{
    ElemType *base;
    ElemType *top;
    int stackSize;
}SqStack;


//---------------------------------------------顺序栈的操作-----------------------------
bool InitStack_Sq(SqStack &S);      //初始化顺序栈
bool StackEmpty_Sq(SqStack S);      //判空, 空则返回true, 非空则返回false
int StackLength_Sq(SqStack S);      //求栈中的元素个数
bool ClearStack_Sq(SqStack &S);     //清空顺序栈
bool DestroyStack_Sq(SqStack &S);   //销毁顺序栈
bool Push_Sq(SqStack &S, ElemType e);  //顺序栈入栈
bool Pop_Sq(SqStack &S, ElemType &e);  //顺序栈出栈
bool GetTop_Sq(SqStack S, ElemType &e);//读取栈顶元素


//---------------------------------------------顺序栈的函数实现--------------------------
//--------------------------------"""top指针始终指向栈顶元素的后移个位置"""------------
//初始化顺序栈
bool InitStack_Sq(SqStack &S){
    S.base = new ElemType[STACK_MAXSIZE];
    if(!S.base) return false;
    S.top = S.base;
    S.stackSize = STACK_MAXSIZE;
    return true;
}
//判空, 空则返回true, 非空则返回false
bool StackEmpty_Sq(SqStack S){
    if(S.base == S.top)
        return true;
    else
        return false;
}
//求栈中的元素个数
int StackLength_Sq(SqStack S){
    return S.top - S.base;
}
//清空顺序栈
bool ClearStack_Sq(SqStack &S){
    if(S.base) S.top = S.base;
    return true;
}
//销毁顺序栈
bool DestroyStack_Sq(SqStack &S){
    if(S.base){
        delete S.base;
        S.base = S.top = NULL;
        S.stackSize = 0;
    }
    return true;
}
//顺序栈入栈
bool Push_Sq(SqStack &S, ElemType e){
    if(S.top-S.base == S.stackSize)
        return false;   //栈满出错
    *S.top++ = e;       //等价于: *S.top = e; S.top++;
    return true;
}
//顺序栈出栈
bool Pop_Sq(SqStack &S, ElemType &e){
    if(S.base == S.top)
        return false;   //栈空, 出栈错误
    e = *--S.top;       //等价于: S.top--; e = *S.top;
    return true;
}
//读取栈顶元素
bool GetTop_Sq(SqStack S, ElemType &e){
    if(S.base == S.top)
        return false;   //栈空, 读栈顶出错
    e = *(S.top - 1);
    return true;
}
```

```
//---------------------------------------------顺序循环队列(少用一个存储空间)的定义-------------------------------
#define QUEUE_MAXSIZE 100        //队列中只能容纳 M-1 个元素
typedef struct{
    ElemType *base;          //初始化的动态存储分配空间
    int front;               //头指针
    int rear;                //尾指针
}SqQueue;

//---------------------------------------------顺序循环队列的操作-----------------------------------------------
bool InitQueue_Sq(SqQueue &Q);       //初始化顺序循环队列
int QueueLength_Sq(SqQueue Q);       //求队列长度
bool EnQueue_Sq(SqQueue &Q, ElemType e);           //入队
bool DeQueue_Sq(SqQueue &Q, ElemType &e);          //出队
bool DestroyQueue_Sq(SqQueue&Q);                   //销毁
bool ClearQueue_Sq(SqQueue &Q);                    //清空
bool QueueEmpty_Sq(SqQueue Q);                     //判空 空则返回true, 非空则返回false
bool GetHead_Sq(SqQueue Q, ElemType &e);           //读队头

//---------------------------------------------顺序循环队列的函数定义-------------------------------------------
//初始化顺序循环队列
bool InitQueue_Sq(SqQueue &Q){
    Q.base = new ElemType[QUEUE_MAXSIZE];
    if(!Q.base) return false;
    Q.front = Q.rear = 0;
    return true;
}
//求队列长度
int QueueLength_Sq(SqQueue Q){
    return (Q.rear-Q.front+QUEUE_MAXSIZE) % QUEUE_MAXSIZE;
}
//入队
bool EnQueue_Sq(SqQueue &Q, ElemType e){
    if( (Q.rear+1)%QUEUE_MAXSIZE == Q.front )
        return false;
    Q.base[Q.rear] = e;
    Q.rear = (Q.rear+1) % QUEUE_MAXSIZE;
    return true;
}
//出队
bool DeQueue_Sq(SqQueue &Q, ElemType &e){
    if(Q.front == Q.rear)
        return false;
    e = Q.base[Q.front];
    Q.front = (Q.front+1) % QUEUE_MAXSIZE;
    return true;
}
//销毁
bool DestroyQueue_Sq(SqQueue&Q){
    delete Q.base;
    Q.front = Q.rear = 0;
    return true;
}
//清空
bool ClearQueue_Sq(SqQueue &Q){
    if(!Q.base) return false;
    Q.front = Q.rear = 0;
    return true;
}
//判空 空则返回true, 非空则返回false
bool QueueEmpty_Sq(SqQueue Q){
    if(Q.rear == Q.front)
        return true;
    else
        return false;
}
//读队头
bool GetHead_Sq(SqQueue Q, ElemType &e){
    if(Q.front == Q.rear)
        return false;
    e = Q.base[Q.front];
    return true;
}
```