

```

1  #include<stdio.h>
2  #include<stdlib.h>
3
4  #define ElemType int
5
6
7  //-----顺序栈的定义-----
8  #define STACK_MAXSIZE 100
9  typedef struct{
10     ElemType *base;
11     ElemType *top;
12     int stackSize;
13 }SqStack;
14
15
16 //-----顺序栈的操作-----
17 bool InitStack_Sq(SqStack &S);    //初始化顺序栈
18 bool StackEmpty_Sq(SqStack S);    //判空，空则返回true，非空则返回false
19 int StackLength_Sq(SqStack S);    //求栈中的元素个数
20 bool ClearStack_Sq(SqStack &S);    //清空顺序栈
21 bool DestroyStack_Sq(SqStack &S);    //销毁顺序栈
22 bool Push_Sq(SqStack &S, ElemType e);    //顺序栈入栈
23 bool Pop_Sq(SqStack &S, ElemType &e);    //顺序栈出栈
24 bool GetTop_Sq(SqStack S, ElemType &e);    //读取栈顶元素
25
26
27 //-----顺序栈的函数实现-----
28 //-----"top指针始终指向栈顶元素的后移个位置"-----
29 //初始化顺序栈
30 bool InitStack_Sq(SqStack &S){
31     S.base = new ElemType[STACK_MAXSIZE];
32     if(!S.base) return false;
33     S.top = S.base;
34     S.stackSize = STACK_MAXSIZE;
35     return true;
36 }
37 //判空，空则返回true，非空则返回false
38 bool StackEmpty_Sq(SqStack S){
39     if(S.base == S.top)
40         return true;
41     else
42         return false;
43 }
44 //求栈中的元素个数
45 int StackLength_Sq(SqStack S){
46     return S.top - S.base;
47 }
48 //清空顺序栈
49 bool ClearStack_Sq(SqStack &S){
50     if(S.base) S.top = S.base;
51     return true;
52 }
53 //销毁顺序栈
54 bool DestroyStack_Sq(SqStack &S){
55     if(S.base){
56         delete S.base;
57         S.base = S.top = NULL;
58         S.stackSize = 0;
59     }
60     return true;
61 }
62 //顺序栈入栈
63 bool Push_Sq(SqStack &S, ElemType e){
64     if(S.top-S.base == S.stackSize)
65         return false;    //栈满出错
66     *S.top++ = e;    //等价于: *S.top = e; S.top++;
67     return true;
68 }
69 //顺序栈出栈
70 bool Pop_Sq(SqStack &S, ElemType &e){
71     if(S.base == S.top)
72         return false;    //栈空，出栈错误
73     e = *--S.top;    //等价于: S.top--; e = *S.top;
74     return true;
75 }
76 //读取栈顶元素
77 bool GetTop_Sq(SqStack S, ElemType &e){
78     if(S.base == S.top)
79         return false;    //栈空，读栈顶出错
80     e = *(S.top - 1);
81     return true;
82 }
83
84

```

```

85
86 //-----顺序循环队列(少用一个存储空间)的定义-----
87 #define QUEUE_MAXSIZE 100 //队列中只能容纳 M-1 个元素
88 typedef struct{
89     ElemType *base; //初始化的动态存储分配空间
90     int front; //头指针
91     int rear; //尾指针
92 }SqQueue;
93
94 //-----顺序循环队列的操作-----
95
96 bool InitQueue_Sq(SqQueue &Q); //初始化顺序循环队列
97 int QueueLength_Sq(SqQueue Q); //求队列长度
98 bool EnQueue_Sq(SqQueue &Q, ElemType e); //入队
99 bool DeQueue_Sq(SqQueue &Q, ElemType &e); //出队
100 bool DestroyQueue_Sq(SqQueue&Q); //销毁
101 bool ClearQueue_Sq(SqQueue &Q); //清空
102 bool QueueEmpty_Sq(SqQueue Q); //判空 空则返回true, 非空则返回false
103 bool GetHead_Sq(SqQueue Q, ElemType &e); //读队头
104
105 //-----顺序循环队列的函数定义-----
106
107 //初始化顺序循环队列
108 bool InitQueue_Sq(SqQueue &Q){
109     Q.base = new ElemType[QUEUE_MAXSIZE];
110     if(!Q.base) return false;
111     Q.front = Q.rear = 0;
112     return true;
113 }
114 //求队列长度
115 int QueueLength_Sq(SqQueue Q){
116     return (Q.rear-Q.front+QUEUE_MAXSIZE) % QUEUE_MAXSIZE;
117 }
118 //入队
119 bool EnQueue_Sq(SqQueue &Q, ElemType e){
120     if( (Q.rear+1)%QUEUE_MAXSIZE == Q.front )
121         return false;
122     Q.base[Q.rear] = e;
123     Q.rear = (Q.rear+1) % QUEUE_MAXSIZE;
124     return true;
125 }
126 //出队
127 bool DeQueue_Sq(SqQueue &Q, ElemType &e){
128     if(Q.front == Q.rear)
129         return false;
130     e = Q.base[Q.front];
131     Q.front = (Q.front+1) % QUEUE_MAXSIZE;
132     return true;
133 }
134 //销毁
135 bool DestroyQueue_Sq(SqQueue&Q){
136     delete Q.base;
137     Q.front = Q.rear = 0;
138     return true;
139 }
140 //清空
141 bool ClearQueue_Sq(SqQueue &Q){
142     if(!Q.base) return false;
143     Q.front = Q.rear = 0;
144     return true;
145 }
146 //判空 空则返回true, 非空则返回false
147 bool QueueEmpty_Sq(SqQueue Q){
148     if(Q.rear == Q.front)
149         return true;
150     else
151         return false;
152 }
153 //读队头
154 bool GetHead_Sq(SqQueue Q, ElemType &e){
155     if(Q.front == Q.rear)
156         return false;
157     e = Q.base[Q.front];
158     return true;
159 }

```