

大家好，我是蓝蓝，这是我们一期数据结构应用题专题的第一天。day03/15

蓝蓝B站首页: [蓝蓝希望你上岸呀B站首页](#)

蓝蓝公众号: [应用题训练营专题](#)

01、写出顺序栈的基本操作包括(进栈、出栈、读栈顶元素等基本操作代码)。

栈与队列

1. 栈的基本操作 (LIFO)

① 顺序栈存储类型

```
#define MaxSize 50  
typedef struct {  
    int data[MaxSize]; // 栈中元素  
    int top; // 栈顶指针  
} SqStack;
```

② 初始化

```
Void InitStack (SqStack &S) {  
    S.top = -1;  
}
```

③ 判空

```
bool StackEmpty (SqStack S) {  
    if (S.top == -1)  
        return true; // 栈空  
    else  
        return false; // 不空  
}
```

④ 入栈

```
bool Push (SqStack &S, int x) {  
    if (S.top == MaxSize - 1) // 栈满  
        return false;  
    S.data[++S.top] = x; // 指针先加1, 再入栈  
    return true;  
}
```

⑤ 出栈

```
bool Pop (SqStack &S, int x) {  
    if (S.top == -1) // 栈空  
        return false;  
    x = S.data[S.top--]; // 先出栈, 后减1  
    return true;  
}
```

① 除栈顶

```

bool GetTop(SqStack S, int x){
    if (S.top == -1) // 栈空
        return false;
    x = S.data[S.top]; // 记录栈顶元素
    return true;
}

```

注意: 为不同情况下
下的栈顶指针指向
会随出入栈代码变化

1) top 指向栈顶元素

```

{
    S.data[++S.top] = x; // 入栈
    x = S.data[S.top--]; // 出栈
}

```

2) S.top = 0 即 top 指向栈
顶端的下位置

```

{
    S.data[S.top++] = x; // 入栈
    x = S.data[--S.top]; // 出栈
}

```

02、写出链栈的基本操作包括(链栈的存储结构、进栈、出栈等基本操作代码)。

2. 栈的链式操作

① 链栈存储结构

```

typedef struct Linknode {
    int data;
    struct Linknode *next; // 指向栈顶
} * LiStack;

```

栈顶结点

栈底结点

只在开口指向时省略

Date / /

② 链栈元素进栈

```

bool push (LinkStack S, int x) {
    StackNode *top = S;
    StackNode *q = (StackNode*) malloc (sizeof StackNode);
    if (q == NULL)
        return false;
    q->data = x; // 头指针
    q->next = top;
    S = q;
    return true;
}

```

栈操作 = 链表操作

③ 链栈元素出栈

```

bool pop (LinkStack S, int x) {
    if (StackEmpty(S)) return false;
    StackNode *top = S;
    int x = top->data; // 头指针
    S = top->next; // 后移动指针
    free(top);
    return true;
}

```

栈删除 = 删除第1个元素

03、写出队列的顺序存储结构，以及循环队列的判空、入队、出队的基本操作代码。

Date / /

front 队头
rear 队尾

3、队列的基本操作 (FIFO) 验证实验

① 队列顺序存储

```

#define MaxSize 50
typedef struct {
    int data[MaxSize]; // 队头指针
    int front, rear; // 队尾指针
} SqQueue;

```

② 队空条件 $Q.front == Q.rear == 0$

③ 循环队列初始化 (数组-1单元的情况)

```

void InitQueue (SqQueue Q) {
    Q.rear = Q.front;
}

```

④ 循环队列判空

```

bool IsEmpty (SqQueue Q) {
    if (Q.rear == Q.front)
        return true;
    else
        return false;
}

```

⑤ 循环队列入队

```

bool EnQueue (SqQueue Q, int x) {
    if ((Q.rear + 1) % MaxSize == Q.front)
        return false;
    Q.data[Q.rear] = x; // 队尾指针加1取模
    Q.rear = (Q.rear + 1) % MaxSize;
    return true;
}

```

⑥ 循环队列出队

```

bool DeQueue (SqQueue Q, int x) {
    if (Q.rear == Q.front)
        return false;
    x = Q.data[Q.front];
    Q.front = (Q.front + 1) % MaxSize;
    return true; // 队头指针加1取模
}

```

04、写出队列的链式存储结构与初始化，以及链队的判空、入队、出队的基本操作代码。

链队列——同时
有队头与队尾指针单
链表

当front==null
且rear==null时
链式队列为空

4. 队列的链式结构

① 链队列存储类型

```
typedef struct LinkNode { // 链式队列结点
    int data;
    struct LinkNode *next;
} LinkNode;
typedef struct {
    LinkNode *front, *rear;
} LinkQueue;
```

② 链队列初始化

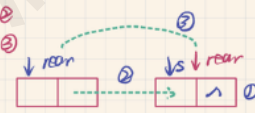
```
void InitQueue (LinkQueue Q) {
    Q.front = Q.rear = (LinkNode *) malloc(sizeof (LinkNode)); // 建立结点
    Q.front->next = NULL;
}
```

③ 链队判空

```
bool IsEmpty (LinkQueue Q) {
    if (Q.front == Q.rear) return true;
    else return false;
}
```

④ 链队入队

```
void EnQueue (LinkQueue Q, int x) {
    LinkNode *s = (LinkNode *) malloc(sizeof (LinkNode));
    s->data = x;
    s->next = NULL; // ①
    Q.rear->next = s; // ②
    Q.rear = s; // ③
}
```



⑤ 链队出队

```
bool DeQueue (LinkQueue Q, int x) {
    if (Q.front == Q.rear) return false;
    LinkNode *p = Q.front->next;
    x = p->data;
    Q.front->next = p->next;
    if (Q.rear == p) // 队尾到队头时，队尾指针也要移动
        Q.rear = Q.front;
    free(p);
    return true;
}
```

