1. 铁道车厢调度问题：
   (1) 如果进站的车厢序列为123，则可能得到的出站车厢序列是132。这是因为铁道车厢调度问题中，车厢的出站顺序通常是在栈操作的过程中确定的，而这里只有一种可能的出站顺序。

   (2) 如果进站的车厢序列为123456，则不能得到435612和135426的出站序列。
   - 435612无法得到，因为在进站序列中，5和6之间没有出站的顺序，所以无法满足要求。
   - 135426无法得到，因为3和4之间没有出站的顺序，所以也无法满足要求。

2.程序的输出结果是："stack"

3.下面是表达式 "A-B×C/D+E^F" 的操作数栈和运算符栈的变化过程：

步骤	操作数栈	运算符栈	备注
1	A		入栈操作数 A
2	A	-	入栈运算符 -
3	A	- B	入栈操作数 B
4	A	- B ×	入栈运算符 ×
5	A	- B × C	入栈操作数 C
6	A	- B × C	出栈运算符 ×
7	A - B × C		运算符 × 出栈
8	A - B × C	/	入栈运算符 /
9	A - B × C	/ D	入栈操作数 D
10	A - B × C	/ D	出栈运算符 /
11	A - B × C / D		运算符 / 出栈
12	A - B × C / D +		入栈运算符 +
13	A - B × C / D + E		入栈操作数 E
14	A - B × C / D + E		出栈运算符 +
15	A - B × C / D + E ^		入栈运算符 ^
16	A - B × C / D + E ^ F		入栈操作数 F
17	A - B × C / D + E ^ F		出栈运算符 ^
18	A - B × C / D + E ^ F		出栈操作数 F
19	A - B × C / D + E ^ F		出栈运算符 +
20	A - B × C / D + E ^ F		出栈运算符 -

   求值结果为 A-B×C/D+E^F。

4.

① 对于n = 1的情况，只需要执行1次移动操作，将一个盘子从起始柱子移动到目标柱子。

② 假设对于n = k的情况，至少需要执行2^k - 1次移动操作。

③ 考虑n = k + 1的情况。我们可以将问题分为三个步骤：
   a. 将前k个盘子从起始柱子移动到中间柱子，需要2^k - 1次移动。
   b. 将第k + 1个盘子从起始柱子移动到目标柱子，需要1次移动。
   c. 将前k个盘子从中间柱子移动到目标柱子，需要2^k - 1次移动。

   总共需要的移动次数为 (2^k - 1) + 1 + (2^k - 1) = 2^(k + 1) - 1。

根据数学归纳法，我们得出结论：对于n阶汉诺塔问题，至少需要执行2^n - 1次移动操作。

5. 实现双向栈操作的算法：

```
#define MAXSIZE 100  // 假定栈的最大容量

typedef struct {
    SElemType data[MAXSIZE];
    int top[2];  // 两个栈的栈顶指针
} tws;

// 初始化双向栈
Status InitStack(tws* t) {
    t->top[0] = -1;  // 初始化第一个栈的栈顶
    t->top[1] = MAXSIZE;  // 初始化第二个栈的栈顶
    return OK;
}

// 入栈操作
Status Push(tws* t, int i, SElemType x) {
    if (i == 0) {
        if (t->top[0] + 1 == t->top[1]) {
            return ERROR;  // 栈满，无法入栈
        }
        t->data[++t->top[0]] = x;  // 第一个栈入栈
    } else if (i == 1) {
        if (t->top[1] - 1 == t->top[0]) {
            return ERROR;  // 栈满，无法入栈
        }
        t->data[--t->top[1]] = x;  // 第二个栈入栈
    }
    return OK;
}

// 出栈操作
Status Pop(tws* t, int i, SElemType* x) {
    if (i == 0) {
        if (t->top[0] == -1) {
            return ERROR;  // 第一个栈为空，无法出栈
        }
        *x = t->data[t->top[0]--];
    } else if (i == 1) {
        if (t->top[1] == MAXSIZE) {
            return ERROR;  // 第二个栈为空，无法出栈
        }
        *x = t->data[t->top[1]++];
    }
    return OK;
}
```

6. 删除单链表中多余的数据元素：

```
typedef struct Node {
    int data;
    struct Node* next;
} Node;

void removeDuplicates(Node* head) {
```

```
    if (head == NULL) {
        return;
    }

    Node* current = head;
    Node* prev = NULL;
    int seen[1000] = {0};  // 假定数据的范围

    while (current != NULL) {
        int value = current->data;

        if (seen[value] == 1) {
            // 如果该值已经出现过，删除当前节点
            prev->next = current->next;
            free(current);
            current = prev->next;
        } else {
            // 如果该值没有出现过，标记为已见
            seen[value] = 1;
            prev = current;
            current = current->next;
        }
    }
}
```