# Homework3 作业3：栈和队列

## 姓名： 叶栩言

## 学号： 2023200033

## 题目 1：判断栈输出序列是否合法

**思路**
读取目标输出序列 从1开始到n依次入栈，每次入栈前，如果当前栈顶是输出序列的第i个（一开始i=1），那么该栈顶出栈，同时i++，然后继续看下一个栈顶是不是输出序列第i个，直到不匹配才入栈。 如果全部元素都已经入栈，此时栈顶与输出序列第i个不相同，且i不等于n，那么返回false，反之则true

```
#include <stdio.h>

int main() {
    int n;
    if (scanf("%d", &n) != 1) {
        return 0;
    }
    int *need = new int[n];
    int i = 0;
    while (i < n) {
        if (scanf("%d", &need[i]) != 1) {
            need[i] = 0;
        }
        i = i + 1;
    }
    int *stack = new int[n];
    int top = -1;
    int want = 0;
    int cur = 1;
    while (cur <= n) {
        while (top >= 0 && want < n && stack[top] == need[want]) {
            top = top - 1;
            want = want + 1;
        }
        top = top + 1;
        stack[top] = cur;
        cur = cur + 1;
    }
    while (top >= 0 && want < n && stack[top] == need[want]) {
        top = top - 1;
        want = want + 1;
    }
    if (want == n) {
        printf("Yes\n");
    } else {
        printf("No\n");
    }
    delete[] stack;
    delete[] need;
    return 0;
}
```

**复杂度**

- 时间复杂度 O(n)
- 空间复杂度 O(n)

## 题目 2：单链表中心对称判定

**思路**
用快慢指针找前半段，把前半段的数据顺手压进顺序栈；如果链表长度是奇数，就跳过正中间的那个结点；最后一边弹栈一边和后半段结点比较，只要有一次不一样就说明不是中心对称。

```
#include <stdio.h>

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

int main() {
    int n;
    if (scanf("%d", &n) != 1) {
        return 0;
    }
    Node *head = 0;
    Node *tail = 0;
    int i = 0;
    while (i < n) {
        int value;
        if (scanf("%d", &value) != 1) {
            value = 0;
        }
        Node *p = new Node;
        p->data = value;
        p->next = 0;
        if (head == 0) {
            head = p;
            tail = p;
        } else {
            tail->next = p;
            tail = p;
        }
        i = i + 1;
    }
    int half = n / 2;
    int *stack = new int[half];
    int top = -1;
    Node *slow = head;
    Node *fast = head;
    while (fast != 0 && fast->next != 0) {
        top = top + 1;
        stack[top] = slow->data;
        slow = slow->next;
        fast = fast->next->next;
    }
    if (fast != 0) {
        slow = slow->next;
    }
    int ok = 1;
    while (slow != 0) {
        if (top < 0 || stack[top] != slow->data) {
            ok = 0;
            break;
        }
        top = top - 1;
        slow = slow->next;
    }
    if (ok == 1 && top == -1) {
        printf("Symmetric\n");
    } else {
        printf("Not symmetric\n");
    }
    delete[] stack;
    Node *q = head;
    while (q != 0) {
        Node *next = q->next;
        delete q;
        q = next;
    }
    return 0;
}
```

**复杂度**

- 时间复杂度 O(n)
- 空间复杂度 O(n)

## 题目 3：在一个数组中实现两个栈

```
#include <stdio.h>

struct DoubleStack {
    int *data;
    int leftTop;
    int rightTop;
    int size;
};
```

```
void init(DoubleStack *s, int n) {
    s->data = new int[n];
    s->leftTop = -1;
    s->rightTop = n;
    s->size = n;
}

int isEmpty(DoubleStack *s, int which) {
    if (which == 1) {
        return s->leftTop < 0;
    } else {
        return s->rightTop >= s->size;
    }
}

int isFull(DoubleStack *s) {
    if (s->leftTop + 1 == s->rightTop) {
        return 1;
    }
    return 0;
}

int push(DoubleStack *s, int which, int value) {
    if (isFull(s)) {
        return 0;
    }
    if (which == 1) {
        s->leftTop = s->leftTop + 1;
        s->data[s->leftTop] = value;
    } else {
        s->rightTop = s->rightTop - 1;
        s->data[s->rightTop] = value;
    }
    return 1;
}

int pop(DoubleStack *s, int which, int *value) {
    if (which == 1) {
        if (s->leftTop < 0) {
            return 0;
        }
        *value = s->data[s->leftTop];
        s->leftTop = s->leftTop - 1;
        return 1;
    } else {
        if (s->rightTop >= s->size) {
            return 0;
        }
        *value = s->data[s->rightTop];
        s->rightTop = s->rightTop + 1;
        return 1;
    }
}
```

```
int main() {
    DoubleStack s;
    init(&s, 12);
    push(&s, 1, 3);
    push(&s, 1, 6);
    push(&s, 2, 9);
    push(&s, 2, 12);
    int x;
    if (pop(&s, 1, &x)) {
        printf("pop left %d\n", x);
    }
    if (pop(&s, 2, &x)) {
        printf("pop right %d\n", x);
    }
    delete[] s.data;
    return 0;
}
```

**复杂度**

- 每次 `push`/`pop` 时间复杂度 O(1)
- 额外空间复杂度 O(n)（数组保存两个栈）

## 题目 4：输出受限双端循环队列

**思路**
使用顺序循环数组保存作业，可以在队头或队尾入队，只允许队头出队。新作业时间若小于队头和队尾作业时间的平均值则插入队头，否则插入队尾。

```
#include <stdio.h>

struct JobDeque {
    int *data;
    int maxSize;
    int front;
    int rear;
    int count;
};

void init(JobDeque *q, int n) {
    q->data = new int[n];
    q->maxSize = n;
    q->front = 0;
    q->rear = 0;
    q->count = 0;
}

int isEmpty(JobDeque *q) {
    return q->count == 0;
}
```

```c
int isFull(JobDeque *q) {
    return q->count == q->maxSize;
}

int goPrev(JobDeque *q, int index) {
    index = index - 1;
    if (index < 0) {
        index = q->maxSize - 1;
    }
    return index;
}

int goNext(JobDeque *q, int index) {
    index = index + 1;
    if (index >= q->maxSize) {
        index = 0;
    }
    return index;
}

int insertFront(JobDeque *q, int value) {
    q->front = goPrev(q, q->front);
    q->data[q->front] = value;
    q->count = q->count + 1;
    return 1;
}

int insertRear(JobDeque *q, int value) {
    q->data[q->rear] = value;
    q->rear = goNext(q, q->rear);
    q->count = q->count + 1;
    return 1;
}

int enqueue(JobDeque *q, int value) {
    if (isFull(q)) {
        return 0;
    }
    if (isEmpty(q)) {
        insertFront(q, value);
        return 1;
    }
    int tailIndex = q->rear - 1;
    if (tailIndex < 0) {
        tailIndex = q->maxSize - 1;
    }
    double average = (q->data[q->front] + q->data[tailIndex]) / 2.0;
    if (value < average) {
        insertFront(q, value);
    } else {
        insertRear(q, value);
    }
    return 1;
}
```

```
int dequeue(JobDeque *q, int *value) {
    if (isEmpty(q)) {
        return 0;
    }
    *value = q->data[q->front];
    q->front = goNext(q, q->front);
    q->count = q->count - 1;
    return 1;
}

int main() {
    int capacity;
    int jobCount;
    if (scanf("%d", &capacity) != 1) {
        return 0;
    }
    if (scanf("%d", &jobCount) != 1) {
        jobCount = 0;
    }
    JobDeque q;
    init(&q, capacity);
    int i = 0;
    while (i < jobCount) {
        int time;
        if (scanf("%d", &time) != 1) {
            time = 0;
        }
        enqueue(&q, time);
        i = i + 1;
    }
    int job;
    while (!isEmpty(&q)) {
        if (dequeue(&q, &job)) {
            printf("%d ", job);
        }
    }
    printf("\n");
    delete[] q.data;
    return 0;
}
```

**复杂度**

- 每次入队、出队时间复杂度 O(1)
- 额外空间复杂度 O(n)

## 题目 5：铁道转轨调度（输出受限双端队列）

**思路**

维护一个输出受限双端队列，记录操作序列：`E` 表示从队头进队，`A` 表示从队尾进队，`D` 表示从队头出队。处理

输入序列时，硬座车厢 (P) 直接在队头入队并立刻出队；软卧 (S) 在队头入队等待；硬卧 (H) 在队尾入队等待。最终先依次出队所有软卧，再出队所有硬卧，即可得到 P-S-H 的最终排列。

```
#include <stdio.h>

struct CarDeque {
    char *data;
    int maxSize;
    int front;
    int rear;
    int count;
};

void initCarDeque(CarDeque *dq, int n) {
    dq->data = new char[n];
    dq->maxSize = n;
    dq->front = 0;
    dq->rear = 0;
    dq->count = 0;
}

int goPrev(CarDeque *dq, int index) {
    index = index - 1;
    if (index < 0) {
        index = dq->maxSize - 1;
    }
    return index;
}

int goNext(CarDeque *dq, int index) {
    index = index + 1;
    if (index >= dq->maxSize) {
        index = 0;
    }
    return index;
}

int isEmpty(CarDeque *dq) {
    return dq->count == 0;
}

int isFull(CarDeque *dq) {
    return dq->count == dq->maxSize;
}

int pushHead(CarDeque *dq, char value) {
    if (isFull(dq)) {
        return 0;
    }
    dq->front = goPrev(dq, dq->front);
    dq->data[dq->front] = value;
    dq->count = dq->count + 1;
    return 1;
}

int pushTail(CarDeque *dq, char value) {
    if (isFull(dq)) {
        return 0;
    }
    dq->data[dq->rear] = value;
    dq->rear = goNext(dq, dq->rear);
    dq->count = dq->count + 1;
    return 1;
}

int popHead(CarDeque *dq, char *value) {
    if (isEmpty(dq)) {
        return 0;
    }
    *value = dq->data[dq->front];
    dq->front = goNext(dq, dq->front);
    dq->count = dq->count - 1;
    return 1;
}

int main() {
    int n;
    if (scanf("%d", &n) != 1) {
        return 0;
    }
    char *cars = new char[n];
    int i = 0;
    while (i < n) {
        scanf(" %c", &cars[i]);
        i = i + 1;
    }
    CarDeque dq;
    initCarDeque(&dq, n);
    char *ops = new char[4 * n];
    char *out = new char[n];
    int opCount = 0;
    int outCount = 0;
    int softLeft = 0;
    int hardLeft = 0;

    i = 0;
    while (i < n) {
        char kind = cars[i];
        if (kind == 'P') {
            pushHead(&dq, kind);
            ops[opCount++] = 'E';
            char served;
            popHead(&dq, &served);
            ops[opCount++] = 'D';
            out[outCount++] = served;
        } else if (kind == 'S') {
            pushHead(&dq, kind);
            ops[opCount++] = 'E';
            softLeft = softLeft + 1;
        } else {
            pushTail(&dq, kind);
            ops[opCount++] = 'A';
            hardLeft = hardLeft + 1;
        }
        i = i + 1;
    }

    i = 0;
    while (i < softLeft) {
        char served;
        popHead(&dq, &served);
        ops[opCount++] = 'D';
        out[outCount++] = served;
        i = i + 1;
    }

    i = 0;
    while (i < hardLeft) {
        char served;
        popHead(&dq, &served);
        ops[opCount++] = 'D';
        out[outCount++] = served;
        i = i + 1;
    }

    i = 0;
    while (i < opCount) {
        printf("%c", ops[i]);
        i = i + 1;
    }
    printf("\n");
    i = 0;
    while (i < outCount) {
        printf("%c", out[i]);
        i = i + 1;
    }
    printf("\n");

    delete[] ops;
    delete[] out;
    delete[] cars;
    delete[] dq.data;
    return 0;
}
```

**复杂度**

- 时间复杂度 O(n)

- 空间复杂度 O(n)