第1章《栈》实验报告

21051117 卢俊成

主要数据结构和变量

- 节点 `Node`
- 栈 `Stack` 及其操作函数 `S_*`
- 队列 `Queue` 及其操作函数 `Q_*`

程序主要流程

1. 初始化栈和队列
2. 读入数字，并将正数和负数分别压入栈和队列
3. 分别输出栈和队列内的数
4. 释放空间

程序主要函数功能

- 栈操作函数 `S_*` 和 队列操作函数 `Q_*`

已实现功能

全部题目要求功能

编译与运行信息

编译信息

```
/usr/bin/gcc -std=c17 -fdiagnostics-color=always -Wfatal-errors -Wall -Wextra -g
1.c -o 1 -lm
```

测试数据

输入

```
10709
15685
29913
2034
6935
10104
20386
12599
21628
14315
-11833
```

```
-28375
-13005
-29892
-4244
-13470
-5515
-3426
-17944
-31934
0
```

输出

```
14315 21628 12599 20386 10104 6935 2034 29913 15685 10709
-11833 -28375 -13005 -29892 -4244 -13470 -5515 -3426 -17944 -31934
```

源代码

```c
// Work on stdc17
#include <stdio.h>
#include <stdlib.h>

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

typedef struct queue {
    int length;
    Node* head;
    Node* tail;
} Queue;

Queue* Q_Create() {
    Queue* q = (Queue*)malloc(sizeof(Queue));
    if (q == NULL) {
        fprintf(stderr, "Error for malloc.\n");
        exit(1);
    }
    q->length = 0;
    q->head = NULL;
    q->tail = NULL;
    return q;
}

void Q_Destroy(Queue* q) {
    Node* current = q->head;
    Node* tmp;
    while (current != NULL) {
```

```
        tmp = current;
        current = current->next;
        free(tmp);
    }
    free(q);
}

int Q_IsEmpty(Queue* q) {
    return q->length == 0;
}

int Q_Length(Queue* q) {
    return q->length;
}

int Q_getTop(Queue* q) {
    if (Q_IsEmpty(q)) {
        fprintf(stderr, "Queue is empty.\n");
        exit(1);
    }
    return q->head->data;
}

void Q_pop(Queue* q) {
    if (Q_IsEmpty(q)) {
        fprintf(stderr, "Queue is empty.\n");
        exit(1);
    }
    Node* tmp = q->head;
    q->head = q->head->next;
    free(tmp);
    q->length--;
    if (q->length == 0) {
        q->tail = NULL;
    }
}

void Q_push(Queue* q, int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        fprintf(stderr, "Error for malloc.\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    if (Q_IsEmpty(q)) {
        q->head = newNode;
        q->tail = newNode;
    } else {
        q->tail->next = newNode;
        q->tail = newNode;
    }
    q->length++;
}
```

```c
typedef struct Stack {
    int length;
    Node* head;
} Stack;

Stack* S_Create() {
    Stack* s = (Stack*)malloc(sizeof(Stack));
    if (s == NULL) {
        fprintf(stderr, "Error for malloc.\n");
        exit(1);
    }
    s->length = 0;
    s->head = NULL;
    return s;
}

void S_Destroy(Stack* s) {
    Node* current = s->head;
    Node* tmp;
    while (current != NULL) {
        tmp = current;
        current = current->next;
        free(tmp);
    }
    free(s);
}

int S_IsEmpty(Stack* s) {
    return s->length == 0;
}

int S_getTop(Stack* s) {
    if (S_IsEmpty(s)) {
        fprintf(stderr, "Stack is empty.\n");
        exit(1);
    }
    return s->head->data;
}

void S_pop(Stack* s) {
    if (S_IsEmpty(s)) {
        fprintf(stderr, "Stack is empty.\n");
        exit(1);
    }
    Node* tmp = s->head;
    s->head = s->head->next;
    free(tmp);
    s->length--;
}

void S_push(Stack* s, int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        fprintf(stderr, "Error for malloc.\n");
        exit(1);
```

```
    }
    newNode->data = data;
    newNode->next = s->head;
    s->head = newNode;
    s->length++;
}

int main() {
#ifndef ONLINE_JUDGE
    freopen("1.in", "r", stdin);
    freopen("1.out", "w", stdout);
    freopen("1.err", "w", stderr);
#endif
    Queue* q = Q_Create();
    Stack* s = S_Create();
    int t;
    while (~scanf("%d", &t), t)
        if (t > 0)
            S_push(s, t);
        else
            Q_push(q, t);
    while (!S_IsEmpty(s)) {
        printf("%d ", S_getTop(s));
        S_pop(s);
    }
    putchar('\n');
    while (!Q_IsEmpty(q)) {
        printf("%d ", Q_getTop(q));
        Q_pop(q);
    }
    Q_Destroy(q);
    S_Destroy(s);
#ifndef ONLINE_JUDGE
    fclose(stdin);
    fclose(stdout);
    fclose(stderr);
#endif
    return 0;
}
```