双向栈的实现

双向栈是一种特殊的栈结构，它在一个一维数组中同时存储两个栈，每个栈的栈底分别位于数组的两端。我们将这两个栈分别称为栈0和栈1。

数据结构定义

使用一个一维数组 `data` 来存储两个栈的数据，栈0的栈底在数组的左端，栈1的栈底在数组的右端。定义两个指针 `top0` 和 `top1` 分别指向栈0和栈1的栈顶位置。

#define MAX\_SIZE 100 // 假设数组的最大容量为100

typedef struct {

int data[MAX\_SIZE];

int top0; // 栈0的栈顶指针

int top1; // 栈1的栈顶指针

} TwoWayStack;

TwoWayStack tws;

初始化双向栈

初始化双向栈时，将栈0的栈顶指针设为-1，将栈1的栈顶指针设为数组的最大索引。

void initStack(TwoWayStack\* tws) {

tws->top0 = -1;

tws->top1 = MAX\_SIZE;

}

入栈操作

int push(TwoWayStack\* tws, int i, int x) {

if (tws->top0 + 1 == tws->top1) { // 栈已满

return 0;

}

if (i == 0) {

tws->data[++tws->top0] = x;

}

else if (i == 1) {

tws->data[--tws->top1] = x;

}

else {

return 0;

}

return 1;

}

出栈操作

int pop(TwoWayStack\* tws, int i, int\* result) {

if (i == 0) {

if (tws->top0 == -1) { // 栈0为空

return 0;

}

\*result = tws->data[tws->top0--];

}

else if (i == 1) {

if (tws->top1 == MAX\_SIZE) { // 栈1为空

return 0;

}

\*result = tws->data[tws->top1++];

}

else {

return 0;

}

return 1;

}

讨论：过程 vs 函数设计

过程设计

优点：

1. 灵活性高：过程设计可以更灵活地处理错误状态，可以通过参数传递错误信息。

2. 状态管理：可以在过程中直接修改传入的参数，方便管理状态变量。

缺点：

1. 可读性差：过程设计的代码可能不如函数设计的代码直观和易读。

2. 调用复杂：调用过程中需要传递更多的参数，增加了调用的复杂性。

函数设计

优点：

1. 可读性强：函数设计的代码通常更直观和易读，符合函数式编程的习惯。

2. 调用简单：调用函数时通常只需要传递必要的参数，减少了调用的复杂性。

缺点：

1. 状态管理：函数设计中返回值只能传递单一结果，需要额外的变量来管理状态。

2. 错误处理：错误处理可能需要额外的逻辑来处理返回值和状态变量。

示例代码C

#include <stdio.h>

#define MAX\_SIZE 100

typedef struct {

int data[MAX\_SIZE];

int top0;

int top1;

} TwoWayStack;

void initStack(TwoWayStack\* tws) {

tws->top0 = -1;

tws->top1 = MAX\_SIZE;

}

int push(TwoWayStack\* tws, int i, int x) {

if (tws->top0 + 1 == tws->top1) {

return 0;

}

if (i == 0) {

tws->data[++tws->top0] = x;

}

else if (i == 1) {

tws->data[--tws->top1] = x;

}

else {

return 0;

}

return 1;

}

int pop(TwoWayStack\* tws, int i, int\* result) {

if (i == 0) {

if (tws->top0 == -1) {

return 0;

}

\*result = tws->data[tws->top0--];

}

else if (i == 1) {

if (tws->top1 == MAX\_SIZE) {

return 0;

}

\*result = tws->data[tws->top1++];

}

else {

return 0;

}

return 1;

}

int main() {

TwoWayStack tws;

initStack(&tws);

int result;

if (push(&tws, 0, 10)) {

printf("Pushed 10 to stack 0\n");

}

else {

printf("Failed to push 10 to stack 0\n");

}

if (push(&tws, 1, 20)) {

printf("Pushed 20 to stack 1\n");

}

else {

printf("Failed to push 20 to stack 1\n");

}

if (pop(&tws, 0, &result)) {

printf("Popped %d from stack 0\n", result);

}

else {

printf("Failed to pop from stack 0\n");

}

if (pop(&tws, 1, &result)) {

printf("Popped %d from stack 1\n", result);

}

else {

printf("Failed to pop from stack 1\n");

}

return 0;

}