

**数据结构课程实验报告**

**（2）**

**姓名：顾宬书**

**学号：2135061913**

**专业：通信**

**完成日期：2023.3.19**

**目录**

[目录 - 1 -](#_Toc434170429)

[1 设计要求 - 2 -](#_Toc434170430)

[2 程序功能框图 - 2 -](#_Toc434170431)

[3 数据结构说明 - 2 -](#_Toc434170432)

[4 重要算法核心代码 - 2 -](#_Toc434170433)

[5 测试运行界面 - 2 -](#_Toc434170434)

[6 完整源程序 - 3 -](#_Toc434170435)

**数据结构实验二（堆栈与队列）**

1. **堆栈应用(1)**

【问题描述】

输入任一表达式，“#”为表达式的结束符，试写一判断表达式中圆括号（“(”与“)”）是否配对的算法。用C语言实现。

【测试情况】

Input the expression string ended with '#' (length≤80):

(a+((a+b)\*c)-d/c)\*e#

Matched

Input the expression string ended with '#' (length≤80):

((a+((a+b)\*c)-d/c)\*e#

Unmatched

【完整源程序】

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

typedef struct Stack\_ {

char\* elements;

int top;

}Stack;

void init(Stack \*s) {

s->elements = (char\*)malloc(sizeof(char) \* 0xFFFFFFF);

s->top = -1;

}

void insert(Stack\* s, char e) {

s->top++;

s->elements[s->top] = e;

return;

}

void pop(Stack\* s) {

if (s->top != -1) {

//free((char\*) & (s->elements[s->top]));

s->top--;

}

else {

return;

}

return;

}

int main() {

Stack s;

init(&s);

char\* input = (char\*)malloc(sizeof(char) \* 0xFFFFF);

scanf("%s", input);

for (int i = 0; i < strlen(input); i++) {

if (input[i] == '(') {

insert(&s, input[i]);

}

else if (input[i] == ')') {

if (s.elements[s.top] == '(') {

pop(&s);

}

}

}

if (s.top == -1) {

printf("matched");

}

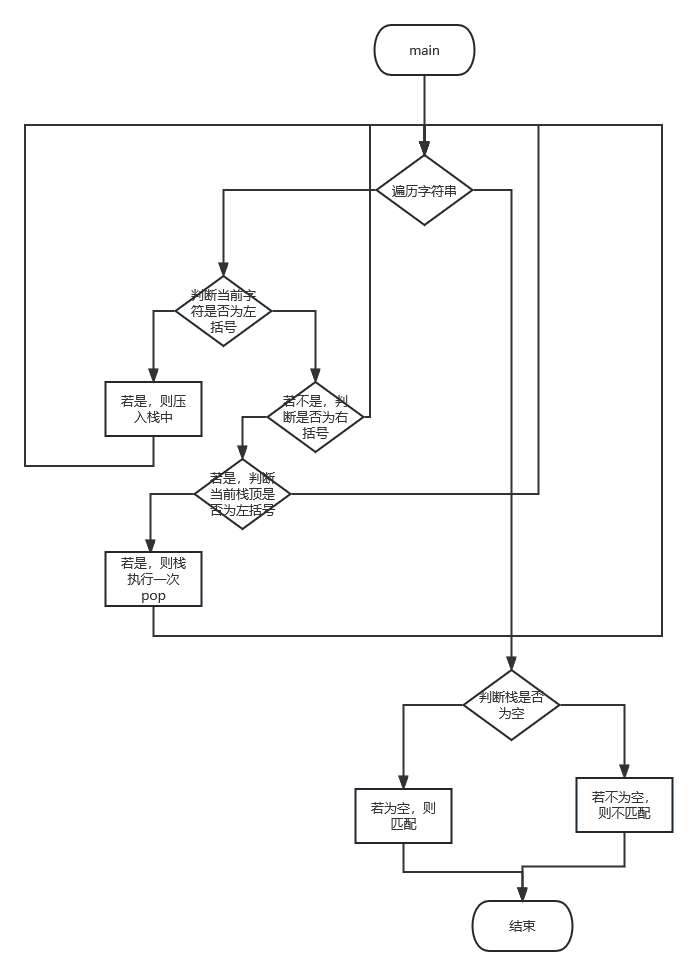
else {

printf("unmatched");

}

}

【序功能框图】



【据结构说明】

名为Stack的结构体类型，包含了两个成员变量：

1. elements，类型为char\*，表示栈的元素存储区，用来存储栈中的各个元素。
2. top，类型为int，表示栈顶元素的下标，当栈为空时，top的值为-1。

该结构体类型的定义采用了typedef关键字进行类型定义，方便后续使用该类型。

【重要算法】

void insert(Stack\* s, char e) {

s->top++;

s->elements[s->top] = e;

return;

}

void pop(Stack\* s) {

if (s->top != -1) {

//free((char\*) & (s->elements[s->top]));

s->top--;

}

else {

return;

}

return;

}

【执行结果】





1. **堆栈应用(2)**

【问题描述】

算术表达式求值问题：约定表达式中只包含加减乘除4种算术运算（表达式中允许出现圆括号），所有运算对象均为简单变量，输入表达式的结束符为“#”，按回车键后即可得到运算结果。用C语言实现。

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define MAX 0xFFFF

typedef struct Stack\_ {

    char operators[MAX];

    int nums[MAX];

    int top;

}Stack;

void init(Stack\* s) {

    s->top = -1;

    return;

}

void insert\_num(Stack\* s, int num) {

    s->top++;

    s->nums[s->top] = num;

    return;

}

void insert\_operator(Stack\* s, char operator) {

    s->top++;

    s->operators[s->top] = operator;

    return;

}

int pop\_num(Stack\* s) {

    if (s->top != -1) {

        s->top--;

        return (int)(s->nums[s->top + 1]);

    }

    else {

        return -1;

    }

}

char pop\_operator(Stack\* s) {

    if (s->top != -1) {

        s->top--;

        return (char)(s->operators[s->top + 1]);

    }

    else {

        return -1;

    }

}

int peek\_num(Stack\* s) {

    if (s->top != -1) {

        return s->nums[s->top];

    }

    return -1;

}

char peek\_operator(Stack\* s) {

    if (s->top != -1) {

        return s->operators[s->top];

    }

    return -1;

}

int a2i(char c) {

    return c - '0';

}

int main() {

    char\* input = (char\*)malloc(sizeof(char) \* 0xFFFFF);

    scanf("%s", input);

    Stack stack\_nums;

    Stack stack\_operators;

    init(&stack\_nums);

    init(&stack\_operators);

    for (int i = 0; i < strlen(input); i++) {

        if (input[i] <= '9' && input[i] >= '0') {

            insert\_num(&stack\_nums, a2i(input[i]));

        }

        else {

            if (input[i] == '+' || input[i] == '-') {

                if (stack\_operators.top != -1) {

                    if (stack\_operators.operators[stack\_operators.top] == '\*') {

                        int A = pop\_num(&stack\_nums);

                        int B = pop\_num(&stack\_nums);

                        insert\_num(&stack\_nums, A \* B);

                        pop\_operator(&stack\_operators);

                        insert\_operator(&stack\_operators, input[i]);

                    }

                    else if (stack\_operators.operators[stack\_operators.top] == '/') {

                        int A = pop\_num(&stack\_nums);

                        int B = pop\_num(&stack\_nums);

                        insert\_num(&stack\_nums, A / B);

                        pop\_operator(&stack\_operators);

                        insert\_operator(&stack\_operators, input[i]);

                    }

                    else {

                        insert\_operator(&stack\_operators, input[i]);

                    }

                }

                else {

                    insert\_operator(&stack\_operators, input[i]);

                }

            }

            else {

                insert\_operator(&stack\_operators, input[i]);

            }

        }

    }

    while (stack\_nums.top != 0){

        if (peek\_operator(&stack\_operators) == '+') {

            int A = pop\_num(&stack\_nums);

            int B = pop\_num(&stack\_nums);

            insert\_num(&stack\_nums, A + B);

            pop\_operator(&stack\_operators);

        }

        if (peek\_operator(&stack\_operators) == '-') {

            int A = pop\_num(&stack\_nums);

            int B = pop\_num(&stack\_nums);

            insert\_num(&stack\_nums, A - B);

            pop\_operator(&stack\_operators);

        }

        if (peek\_operator(&stack\_operators) == '\*') {

            int A = pop\_num(&stack\_nums);

            int B = pop\_num(&stack\_nums);

            insert\_num(&stack\_nums, A \* B);

            pop\_operator(&stack\_operators);

        }if (peek\_operator(&stack\_operators) == '/') {

            int A = pop\_num(&stack\_nums);

            int B = pop\_num(&stack\_nums);

            insert\_num(&stack\_nums, A / B);

            pop\_operator(&stack\_operators);

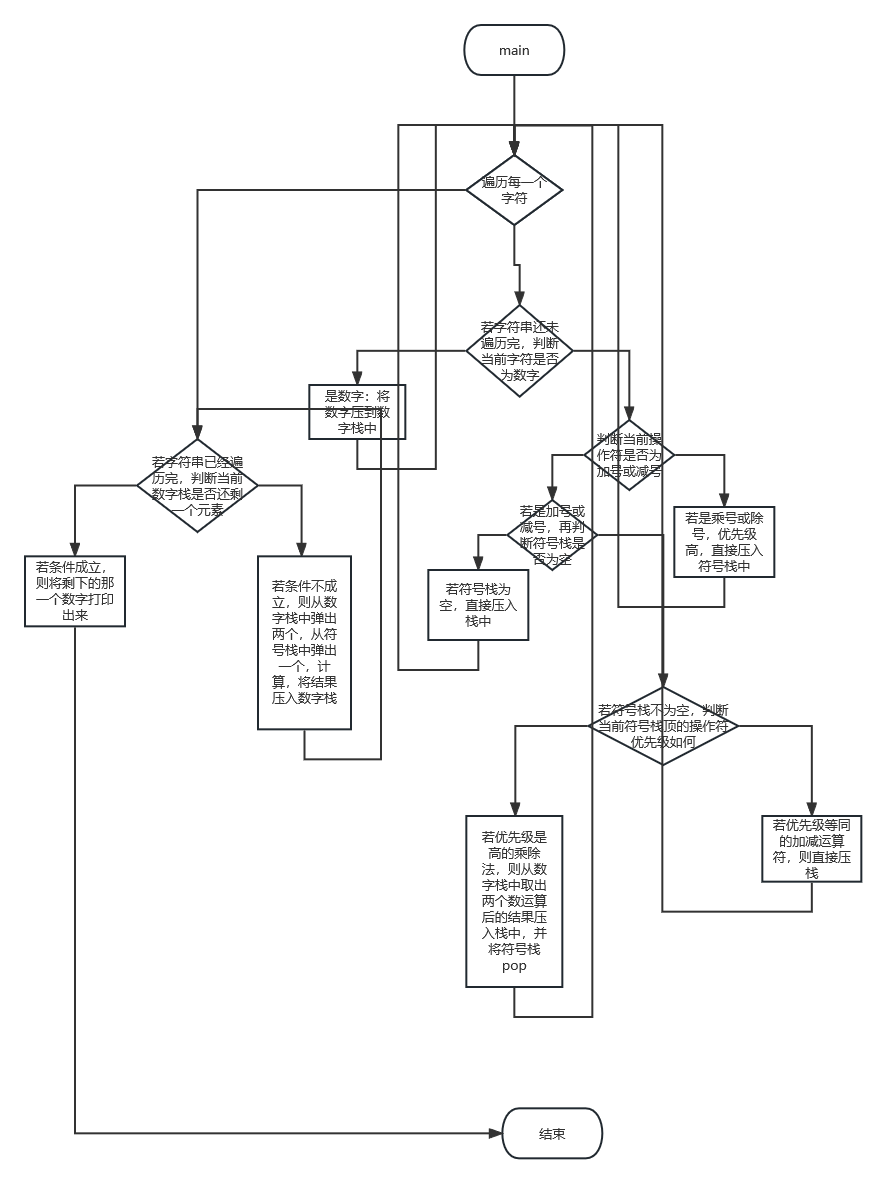
        }

    }

    printf("%d", pop\_num(&stack\_nums));

}

【流程框图】



【数据结构说明】

Stack中有两个线性表，一个用于存放数字，另一个用于存放操作符。若当前的stack是数字栈，则存放操作符的线性表弃用；反之，则存放数字的线性表弃用。

【重要算法】

for (int i = 0; i < strlen(input); i++) {

        if (input[i] <= '9' && input[i] >= '0') {

            insert\_num(&stack\_nums, a2i(input[i]));

        }

        else {

            if (input[i] == '+' || input[i] == '-') {

                if (stack\_operators.top != -1) {

                    if (stack\_operators.operators[stack\_operators.top] == '\*') {

                        int A = pop\_num(&stack\_nums);

                        int B = pop\_num(&stack\_nums);

                        insert\_num(&stack\_nums, A \* B);

                        pop\_operator(&stack\_operators);

                        insert\_operator(&stack\_operators, input[i]);

                    }

                    else if (stack\_operators.operators[stack\_operators.top] == '/') {

                        int A = pop\_num(&stack\_nums);

                        int B = pop\_num(&stack\_nums);

                        insert\_num(&stack\_nums, A / B);

                        pop\_operator(&stack\_operators);

                        insert\_operator(&stack\_operators, input[i]);

                    }

                    else {

                        insert\_operator(&stack\_operators, input[i]);

                    }

                }

                else {

                    insert\_operator(&stack\_operators, input[i]);

                }

            }

            else {

                insert\_operator(&stack\_operators, input[i]);

            }

        }

    }

    while (stack\_nums.top != 0){

        if (peek\_operator(&stack\_operators) == '+') {

            int A = pop\_num(&stack\_nums);

            int B = pop\_num(&stack\_nums);

            insert\_num(&stack\_nums, A + B);

            pop\_operator(&stack\_operators);

        }

        if (peek\_operator(&stack\_operators) == '-') {

            int A = pop\_num(&stack\_nums);

            int B = pop\_num(&stack\_nums);

            insert\_num(&stack\_nums, A - B);

            pop\_operator(&stack\_operators);

        }

        if (peek\_operator(&stack\_operators) == '\*') {

            int A = pop\_num(&stack\_nums);

            int B = pop\_num(&stack\_nums);

            insert\_num(&stack\_nums, A \* B);

            pop\_operator(&stack\_operators);

        }if (peek\_operator(&stack\_operators) == '/') {

            int A = pop\_num(&stack\_nums);

            int B = pop\_num(&stack\_nums);

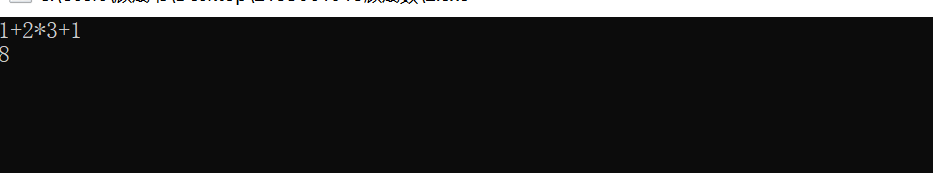
            insert\_num(&stack\_nums, A / B);

            pop\_operator(&stack\_operators);

        }

    }

【执行结果】



1. **队列应用**

【问题描述】

对顺序循环队列，将自然数按序入队、出队。具体的操作是：队列未满时，入队、入队、出队（即做连续两次入队操作之后，做一次出队操作），输出出队元素的值；队列满时，执行连续的出队操作，输出出队元素的值（应与队列未满时所输出的有不同标识），直至队列为空。编写算法实现以上操作。用C语言实现。

【测试情况】

假设顺序循环队列使用的数组大小为20

运行程序得到的实际输出如下：

1\*2\*3\*4\*5\*6\*7\*8\*9\*10\*11\*12\*13\*14\*15\*16\*17\*18\*19#20#21#22#23#24#25#26#27#28#29#30#31#32#33#34#35#36#37#

【完整源程序】

#include<stdio.h>

#include<stdlib.h>

#define MAX 20

typedef struct queue {

    int data[MAX];

    int head;

    int tail;

    int isFull;

}Queue;

void init(Queue\* q) {

    q->head = 0;

    q->tail = 0;

    q->isFull = 0;

}

int isEmpty(Queue\* q) {

    if (!q->isFull && q->head == q->tail) {

        return 1;

    }

    else {

        return 0;

    }

}

int pop(Queue\* q) {

    if (!q->isFull && q->tail == q->head) {

        printf("Empty\n");

        return -1;

    }

    else {

        int res = q->data[q->head];

        q->data[q->head] = 0;

        q->head = (q->head + 1) % MAX;

        if (q->isFull) {

            q->isFull = 0;

        }

        return res;

    }

}

int insert(Queue\* q, int n) {

    if (!q->isFull) {

        q->data[q->tail] = n;

        q->tail = (q->tail + 1) % MAX;

        if (q->tail == q->head) {

            q->isFull = 1;

        }

        return 1;

    }

    else {

        return -1;

    }

}

int main() {

    Queue q;

    init(&q);

    int i = 1;

    while (1) {

        if (q.isFull)

            break;

        insert(&q, i);

        i++;

        if (q.isFull)

            break;

        insert(&q, i);

        printf("%d\*", pop(&q));

    }

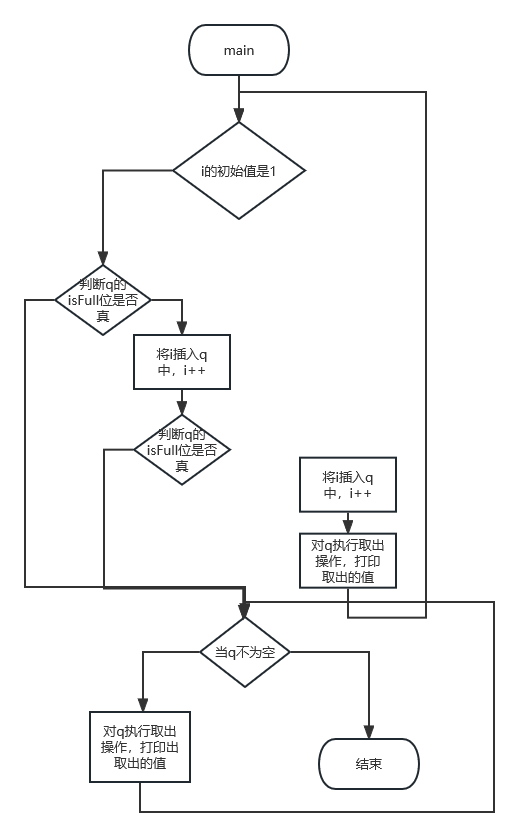
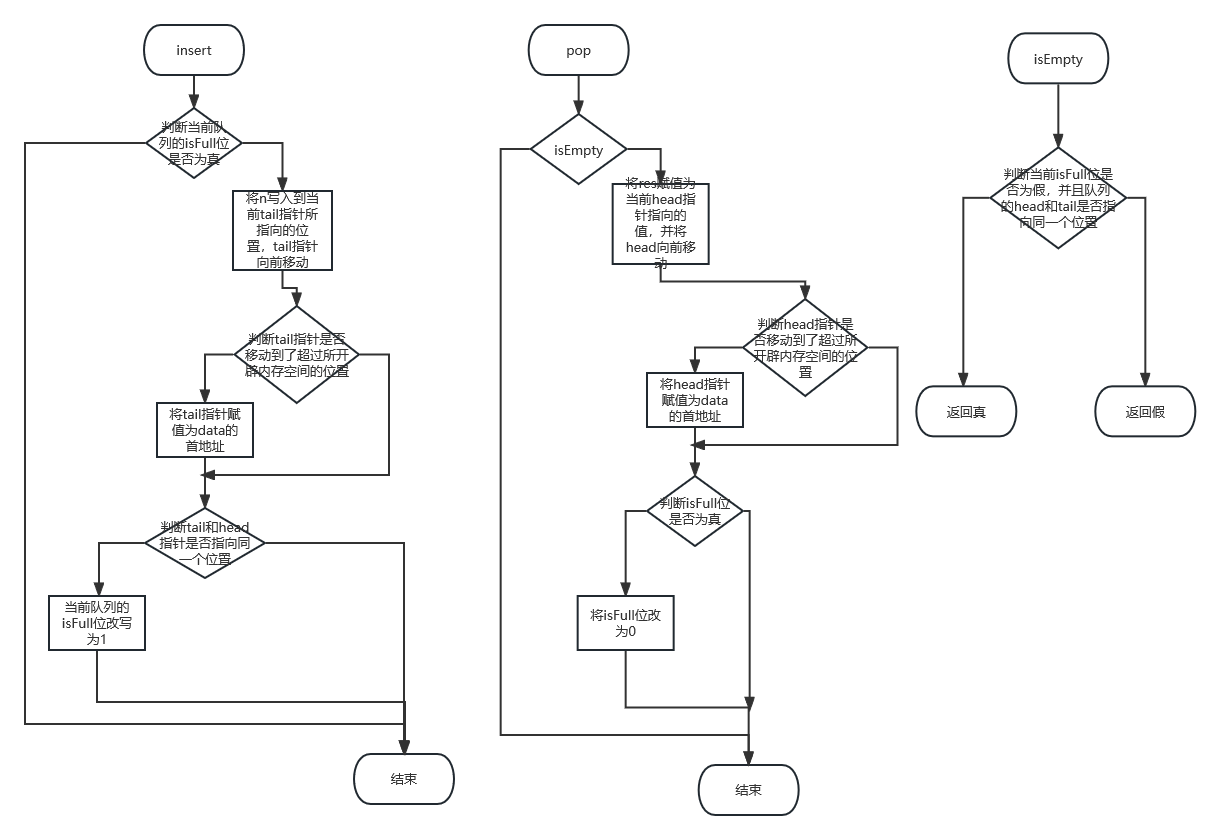
    while (!isEmpty(&q)) {

        printf("%d#", pop(&q));

    }

}

【程序功能框图】



【数据结构说明】

名为Queue的结构体，它包含以下成员变量：

* int data[MAX]：一个数组，用于存储队列中的元素，MAX是一个常量，代表数组的最大长度。
* int head：一个整型变量，代表队列头的位置。
* int tail：一个整型变量，代表队列尾的位置。
* int isFull：一个整型变量，代表队列是否已满。

这个结构体表示了一个队列的基本属性，可以用来实现队列相关的操作。

【重要算法】

int pop(Queue\* q) {

    if (!q->isFull && q->tail == q->head) {

        printf("Empty\n");

        return -1;

    }

    else {

        int res = q->data[q->head];

        q->data[q->head] = 0;

        q->head = (q->head + 1) % MAX;

        if (q->isFull) {

            q->isFull = 0;

        }

        return res;

    }

}

int insert(Queue\* q, int n) {

    if (!q->isFull) {

        q->data[q->tail] = n;

        q->tail = (q->tail + 1) % MAX;

        if (q->tail == q->head) {

            q->isFull = 1;

        }

        return 1;

    }

    else {

        return -1;

    }

}

【执行结果】

