数据结构作业 第四周

霍斌 PB24111627

1 3.21

中缀表示式转换为逆波兰表示式

```c
char* infixToRPN(const char* infix) {
    // 将单字母变量和双目四则运算符的正常表示式转换为逆波兰表示式
    // 使用CharStack.h中的栈
    Stack* stack = createStack();
    char *output = (char*)malloc(strlen(infix) + 1);
    int j = 0;
    for(int i = 0; infix[i] != '\0'; i++) {
        if(int(infix[i]) >= int('a') && int(infix[i] <= int('z'))) {
            output[j++] = infix[i];
        } else if(infix[i] == '(') {
            push(stack, infix[i]);
        } else if(infix[i] == ')') {
            while(!isEmpty(stack) && peek(stack) != '(') {
                output[j++] = peek(stack);
                pop(stack);
            }
            pop(stack); // 弹出左括号
        } else { // 如果是运算符
            if(infix[i] == '+' || infix[i] == '-') {
                while(!isEmpty(stack) && peek(stack) != '(') {
                    output[j++] = peek(stack);
                    pop(stack);
                }
                push(stack, infix[i]);
            } else if(infix[i] == '*' || infix[i] == '/') {
                while(!isEmpty(stack) && peek(stack) != '(' && peek(stack) != '+' && peek(stack) != '-') {
                    output[j++] = peek(stack);
                    pop(stack);
                }
                push(stack, infix[i]);
            }
        }
    }
    while(!isEmpty(stack)) {
        output[j++] = peek(stack);
        pop(stack);
    }
    output[j] = '\0';
    return output;
}
```

2 3.22

计算逆波兰表达式(通过float *values 指定字母变量值, a = values[0], b = values[1]...)

```
int calculateRPN(char* rpn, float* values) {
    int j = 0;
    float *result = (float*)malloc(strlen(rpn) * sizeof(float)); // 用数组模拟栈
    char *p = rpn; // p指向rpn的开头
    while(*p != '\0') {
        if(int(*p) >= int('a') && int(*p) <= int('z')) {
            result[j++] = values[int(*p) - int('a')]; // 如果是变量就入栈
        } else {
            float op2 = result[--j];
            float op1 = result[--j];
            switch(*p) {
                case '+':
                    result[j++] = op1 + op2;
                    break;
                case '-':
                    result[j++] = op1 - op2;
                    break;
                case '*':
                    result[j++] = op1 * op2;
                    break;
                case '/':
                    result[j++] = op1 / op2;
                    break;
            }
        }
        p++;
    }
    return result[0];
}
```

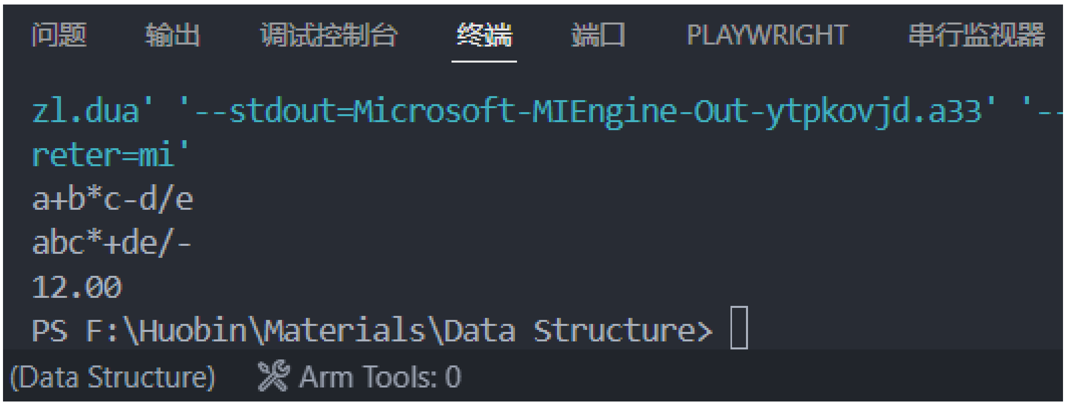
3.21&3.22测试程序

```
#include "RPN.h"

int main() {
    const char* infix = "a+b*c-d/e";
    char* rpn = infixToRPN(infix);
    float values[5] = {5, 3, 4, 10, 2}; // a=5, b=3, c=4, d=10, e=2

    printf("%s\n", infix);
    printf("%s\n", rpn);
    printf("%.2f\n", calculateRPN(rpn, values)); // 12.00

    return 0;
}
```



```
zl.dua' '--stdout=Microsoft-MIEngine-Out-ytpkovjd.a33' '--
reter=mi'
a+b*c-d/e
abc*+de/-
12.00
PS F:\Huobin\Materials\Data Structure> 
```

附录

1 CharStack(字符栈辅助头文件)

```c
#ifndef CHARSTACK_H
#define CHARSTACK_H

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define MAXSIZE 100
typedef struct Stack {
    int top;
    char data[MAXSIZE];
} Stack;

Stack* createStack();
bool isFull(Stack *stack);
bool isEmpty(Stack *stack);
void push(Stack *stack, char value);
void pop(Stack *stack);
char peek(Stack *stack);

Stack* createStack() {
    Stack *stack = (Stack*)malloc(sizeof(Stack));
    stack->top = -1;
    return stack;
}

bool isFull(Stack *stack) {
    return stack->top == MAXSIZE - 1;
}

bool isEmpty(Stack *stack) {
    return stack->top == -1;
}

void push(Stack *stack, char value) {
    if(!isFull(stack)) {
        stack->data[stack->top + 1] = value;
        stack->top++;
    } else {
        perror("StackOverflow");
    }
}

void pop(Stack *stack) {
    if(!isEmpty(stack)) {
        stack->top--;
    } else {
        perror("Stack is Empty");
    }
}

char peek(Stack *stack) {
    if(!isEmpty(stack)) {
        return stack->data[stack->top];
    } else {
        perror("Stack is Empty");
        return '\0';
    }
}

#endif // CHARSTACK_H
```

2 RPN(逆波兰表示式获取与计算)

```
#ifndef RPN_H
#define RPN_H

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "CharStack.h"

// 获取字符串长度
int strlen(const char* str) {
    int len = 0;
    while(str[len] != '\0') {
        len++;
    }
    return len;
}

char* infixToRPN(const char* infix) {
    // 将单字母变量和双目四则运算符的正常表示式转换为逆波兰表示式
    // 使用CharStack.h中的栈
    Stack* stack = createStack();
    char *output = (char*)malloc(strlen(infix) + 1);
    int j = 0;
    for(int i = 0; infix[i] != '\0'; i++) {
        if(int(infix[i]) >= int('a') && int(infix[i] <= int('z'))) {
            output[j++] = infix[i];
        } else if(infix[i] == '(') {
            push(stack, infix[i]);
        } else if(infix[i] == ')') {
            while(!isEmpty(stack) && peek(stack) != '(') {
                output[j++] = peek(stack);
```

```c
                pop(stack);
            }
            pop(stack); // 弹出左括号
        } else { // 如果是运算符
            if(infix[i] == '+' || infix[i] == '-') {
                while(!isEmpty(stack) && peek(stack) != '(') {
                    output[j++] = peek(stack);
                    pop(stack);
                }
                push(stack, infix[i]);
            } else if(infix[i] == '*' || infix[i] == '/') {
                while(!isEmpty(stack) && peek(stack) != '(' && peek(stack) != '+' && peek(stack) != '-') {
                    output[j++] = peek(stack);
                    pop(stack);
                }
                push(stack, infix[i]);
            }
        }
    }
    while(!isEmpty(stack)) {
        output[j++] = peek(stack);
        pop(stack);
    }
    output[j] = '\0';
    return output;
}

// 计算逆波兰表示式，values数组存储变量a-z的值
float calculateRPN(char* rpn, float* values) {
    int j = 0;
    float *result = (float*)malloc(strlen(rpn) * sizeof(float)); // 用数组模拟栈
    char *p = rpn; // p指向rpn的开头
    while(*p != '\0') {
        if(int(*p) >= int('a') && int(*p) <= int('z')) {
            result[j++] = values[int(*p) - int('a')]; // 如果是变量就入栈
        } else {
            float op2 = result[--j];
            float op1 = result[--j];
            switch(*p) {
                case '+':
                    result[j++] = op1 + op2;
                    break;
                case '-':
                    result[j++] = op1 - op2;
                    break;
                case '*':
                    result[j++] = op1 * op2;
                    break;
                case '/':
                    result[j++] = op1 / op2;
                    break;
            }
        }
        p++;
    }
    return result[0];
}

#endif // RPN_H
```