

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

```
1 char* infixToRPN(const char* infix) {
2     // 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
3     // 使用CharStack.h中的栈
4     Stack* stack = createStack();
5     char *output = (char*)malloc(strlen(infix) + 1);
6     int j = 0;
7     for(int i = 0; infix[i] != '\0'; i++) {
8         if(int(infix[i]) >= int('a') && int(infix[i] <= int('z')) {
9             output[j++] = infix[i];
10        } else if(infix[i] == '(') {
11            push(stack, infix[i]);
12        } else if(infix[i] == ')') {
13            while(!isEmpty(stack) && peek(stack) != '(') {
14                output[j++] = peek(stack);
15                pop(stack);
16            }
17            pop(stack); // 弹出左括号
18        } else { // 如果是运算符
19            if(infix[i] == '+' || infix[i] == '-') {
20                while(!isEmpty(stack) && peek(stack) != '(') {
21                    output[j++] = peek(stack);
22                    pop(stack);
23                }
24                push(stack, infix[i]);
25            } else if(infix[i] == '*' || infix[i] == '/') {
26                while(!isEmpty(stack) && peek(stack) != '(' && peek(stack) != '+' &&
27                peek(stack) != '-') {
28                    output[j++] = peek(stack);
29                    pop(stack);
30                }
31                push(stack, infix[i]);
32            }
33        }
34        while(!isEmpty(stack)) {
35            output[j++] = peek(stack);
36            pop(stack);
37        }
38        output[j] = '\0';
39        return output;
40    }
```

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]...)

```
1  int calculateRPN(char* rpn, float* values) {
2      int j = 0;
3      float *result = (float*)malloc(strlen(rpn) * sizeof(float)); // 用数组模拟栈
4      char *p = rpn; // p指向rpn的开头
5      while(*p != '\0') {
6          if(int(*p) >= int('a') && int(*p) <= int('z')) {
7              result[j++] = values[int(*p) - int('a')]; // 如果是变量就入栈
8          } else {
9              float op2 = result[--j];
10             float op1 = result[--j];
11             switch(*p) {
12                 case '+':
13                     result[j++] = op1 + op2;
14                     break;
15                 case '-':
16                     result[j++] = op1 - op2;
17                     break;
18                 case '*':
19                     result[j++] = op1 * op2;
20                     break;
21                 case '/':
22                     result[j++] = op1 / op2;
23                     break;
24             }
25         }
26         p++;
27     }
28     return result[0];
29 }
```

3.21&3.22测试程序

```
1  #include "RPN.h"
2
3  int main() {
4      const char* infix = "a+b*c-d/e";
5      char* rpn = infixToRPN(infix);
6      float values[5] = {5, 3, 4, 10, 2}; // a=5, b=3, c=4, d=10, e=2
7
8      printf("%s\n", infix);
9      printf("%s\n", rpn);
10     printf("%.2f\n", calculateRPN(rpn, values)); // 12.00
11
12     return 0;
13 }
```

```
问题 输出 调试控制台 终端 端口 PLAYWRIGHT 串行监视器

z1.dua' '--stdout=Microsoft-MIEngine-Out-ytpkovjd.a33' '--
reter=mi'
a+b*c-d/e
abc*+de/-
12.00
PS F:\Huobin\Materials\Data Structure> 
(Data Structure)  Arm Tools: 0
```

附录

1 CharStack(字符栈辅助头文件)

```
1  #ifndef CHARSTACK_H
2  #define CHARSTACK_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <stdbool.h>
7
8  #define MAXSIZE 100
9  typedef struct Stack {
10     int top;
11     char data[MAXSIZE];
12 } Stack;
13
14 Stack* createStack();
15 bool isFull(Stack *stack);
16 bool isEmpty(Stack *stack);
17 void push(Stack *stack, char value);
18 void pop(Stack *stack);
19 char peek(Stack *stack);
20
21 Stack* createStack() {
22     Stack *stack = (Stack*)malloc(sizeof(Stack));
23     stack->top = -1;
24     return stack;
25 }
26
27 bool isFull(Stack *stack) {
28     return stack->top == MAXSIZE - 1;
29 }
30
31 bool isEmpty(Stack *stack) {
32     return stack->top == -1;
33 }
34
35 void push(Stack *stack, char value) {
36     if(!isFull(stack)) {
37         stack->data[stack->top + 1] = value;
38         stack->top++;
39     } else {
```

```

40     perror("StackOverflow");
41 }
42 }
43
44 void pop(Stack *stack) {
45     if(!isEmpty(stack)) {
46         stack->top--;
47     } else {
48         perror("Stack is Empty");
49     }
50 }
51
52 char peek(Stack *stack) {
53     if(!isEmpty(stack)) {
54         return stack->data[stack->top];
55     } else {
56         perror("Stack is Empty");
57         return '\0';
58     }
59 }
60
61 #endif // CHARSTACK_H

```

2 RPN(逆波兰表示式获取与计算)

```

1  #ifndef RPN_H
2  #define RPN_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <stdbool.h>
7  #include "CharStack.h"
8
9  // 获取字符串长度
10 int strlen(const char* str) {
11     int len = 0;
12     while(str[len] != '\0') {
13         len++;
14     }
15     return len;
16 }
17
18 char* infixToRPN(const char* infix) {
19     // 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
20     // 使用CharStack.h中的栈
21     Stack* stack = createStack();
22     char *output = (char*)malloc(strlen(infix) + 1);
23     int j = 0;
24     for(int i = 0; infix[i] != '\0'; i++) {
25         if(int(infix[i]) >= int('a') && int(infix[i] <= int('z')) {
26             output[j++] = infix[i];
27         } else if(infix[i] == '(') {
28             push(stack, infix[i]);
29         } else if(infix[i] == ')') {
30             while(!isEmpty(stack) && peek(stack) != '(') {
31                 output[j++] = peek(stack);

```

```

32         pop(stack);
33     }
34     pop(stack); // 弹出左括号
35 } else { // 如果是运算符
36     if(infix[i] == '+' || infix[i] == '-') {
37         while(!isEmpty(stack) && peek(stack) != '(') {
38             output[j++] = peek(stack);
39             pop(stack);
40         }
41         push(stack, infix[i]);
42     } else if(infix[i] == '*' || infix[i] == '/') {
43         while(!isEmpty(stack) && peek(stack) != '(' && peek(stack) != '+' &&
peek(stack) != '-') {
44             output[j++] = peek(stack);
45             pop(stack);
46         }
47         push(stack, infix[i]);
48     }
49 }
50 }
51 while(!isEmpty(stack)) {
52     output[j++] = peek(stack);
53     pop(stack);
54 }
55 output[j] = '\0';
56 return output;
57 }
58
59 // 计算逆波兰表示式, values数组存储变量a-z的值
60 float calculateRPN(char* rpn, float* values) {
61     int j = 0;
62     float *result = (float*)malloc(strlen(rpn) * sizeof(float)); // 用数组模拟栈
63     char *p = rpn; // p指向rpn的开头
64     while(*p != '\0') {
65         if(int(*p) >= int('a') && int(*p) <= int('z')) {
66             result[j++] = values[int(*p) - int('a')]; // 如果是变量就入栈
67         } else {
68             float op2 = result[--j];
69             float op1 = result[--j];
70             switch(*p) {
71                 case '+':
72                     result[j++] = op1 + op2;
73                     break;
74                 case '-':
75                     result[j++] = op1 - op2;
76                     break;
77                 case '*':
78                     result[j++] = op1 * op2;
79                     break;
80                 case '/':
81                     result[j++] = op1 / op2;
82                     break;
83             }
84         }
85         p++;
86     }

```

```
87     return result[0];
88 }
89
90 #endif // RPN_H
```