哈尔滨工业大学计算学部

实验报告

课程名称：数据结构与算法

课程类型：专业基础（必修）

实验项目：栈结构及其应用

实验题目：算术表达式求值（算术计算器）

实验日期：2022年10月6日

班级：2103601

学号：2021112845

姓名：张智雄

|  |  |  |
| --- | --- | --- |
| 设计成绩 | 报告成绩 | 指导老师 |
|  |  | 李秀坤 |

**一、实验目的**

表达式求值是实现程序设计语言的基本问题之一，也是栈的应用的一个典型例子。一个算术表达式是由操作数（operand）、运算符（operator）和界限符（delimiter）组成的。假设操作数是正整数，运算符只含加减乘除等四种二元运算符，界限符有左右括号和表达式起始、结束符“#”，如：#（7+15）\*（23-28/4）#。引入表达式起始、结束符是为了方便。设计一个程序，演示算术表达式求值的过程。

**二、实验要求及实验环境**

实验要求：

1. 从文本文件输入任意一个语法正确的（中缀）表达式，显示并保存该表达式。
2. 利用栈结构，把（中缀）表达式转换成后缀表达式，并以适当的方式展示栈的状态变化过程和所得到的后缀表达式。
3. 利用栈结构，对后缀表达式进行求值，并以适当的方式展示栈的状态变化过程和最终结果。

实验环境：Windows11操作系统+VS Code编译器

**三、设计思想**

此实验主要分为**栈结构的实现**以及**中缀表达式的处理**两个部分

**1.栈结构的实现**

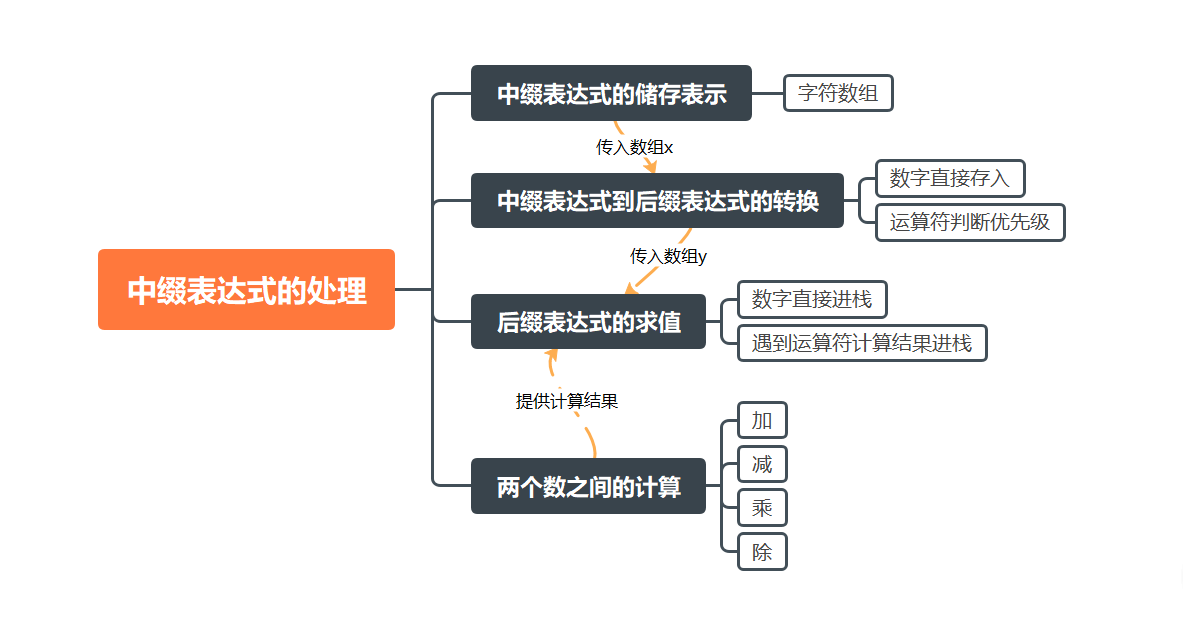
实验采用顺序栈的结构，利用template方式定义了一个栈的模板类，通过数组data[]储存栈中元素，整数型变量Top作为栈顶指针。

此部分主要实现了以下功能：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **函数名** | **函数功能** | **函数参数** | **函数返回值** |
| **1** | **Stack()** | **初始化栈** | **无** | **无** |
| **2** | **push(n)** | **压栈** | **进栈元素n** | **无** |
| **3** | **pop()** | **出栈** | **无** | **栈顶元素d** |
| **4** | **size()** | **查看栈长度** | **无** | **栈长度length** |
| **5** | **isEmpty()** | **判断是否栈空** | **无** | **bool变量x** |
| **6** | **top()** | **读取栈顶元素** | **无** | **栈顶元素d** |
| **7** | **show()** | **打印当前栈中元素** | **无** | **无** |

**2.中缀表达式的处理**

**此部分函数调用已经逻辑关系如下：**



1. **首先是对中缀表达式的储存表示。**

此过程通过一个字符数组x[]即可实现，

1. **其次是中缀表达式到后缀表达式的转换**

这里主要是对运算符“(”“+”“-”“\*”“/”“)”的优先级判断。基本算法思想是：对上述字符数组从头至尾扫描，

1. 遇见数字字符或小数点“.”直接存进后缀表达式的字符数组y[]中；
2. 如遇到运算符，则判断其与栈顶元素的优先级后，执行对应操作后再进栈。从栈中弹出的运算符依序存进后缀表达式的字符数组y[]。

注意，在此步骤中，由于字符一次扫描一位，为方便可视化以及多位数的计算，在相邻两个字符间以空格间隔，表示多位数的连续数字字符之间则无空格隔开。

**优先级如下表所示：**

|  |  |
| --- | --- |
| **运算符** | **优先级及相关操作** |
| **“(”** | **无序比较，直接进入栈** |
| **“\*”或“/”** | **当栈顶为“\*”或“/”时，弹出当前栈顶直至不为“\*”或“/”后进栈** |
| **“+”或“-”** | **当栈顶不为“#”或“(”时，弹出当前栈顶直至为“#”或“(”后进栈** |
| **“)”** | **无需进栈，逐个将栈顶弹出直至“(”恰好弹出栈** |
| **“#”** | **第一次直接进栈，第二次直接将栈中元素全部弹出。** |

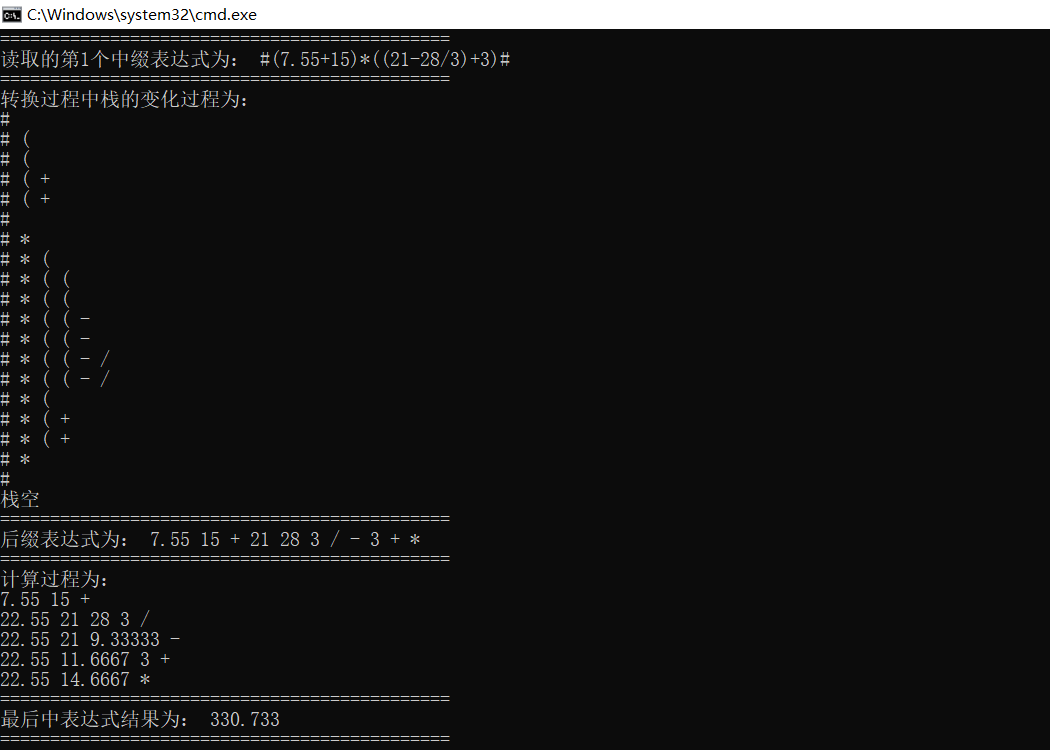
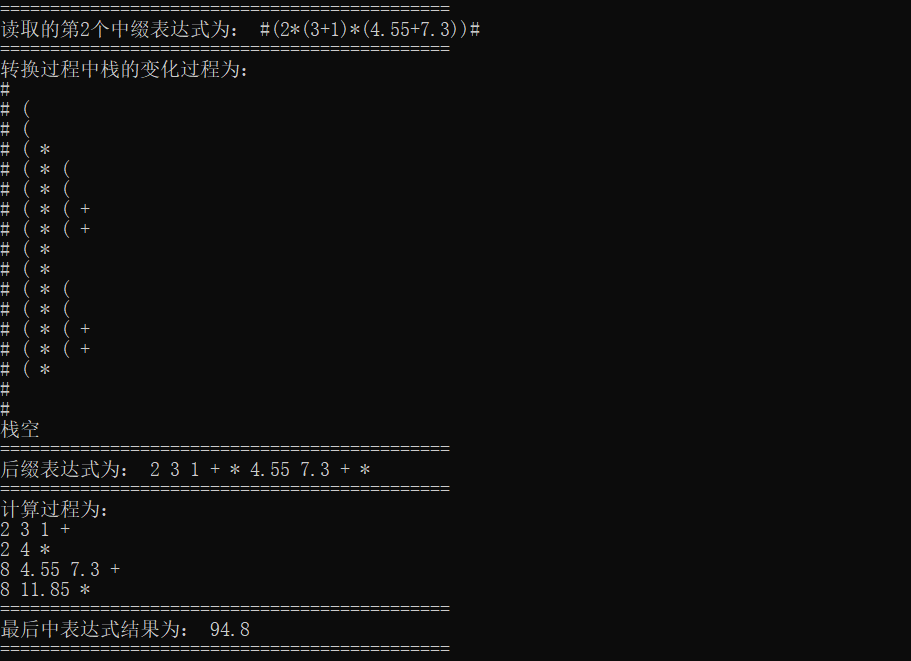
1. **最后是对后缀表达式的求值**

基本思想是从头至尾扫描后缀表达式y[],

1. 遇到数字字符时通过ASCII码将其转换为float数据类型后压入栈中，
2. 遇到运算符“+”“-”“\*”“/”时，依次弹出栈顶的两个数字进行运算，运算结果压入栈中，继续扫描。

**四、测试结果**

测试用例：#(7.55+15)\*((21-28/3)+3)#，#(2\*(3+1)\*(4.55+7.3))#

测试结果如下：

**五、经验体会与不足**

通过此实验，对栈的使用以及特点有了比较深的了解，同时对中缀表达式和后缀表达式有了一定了解。栈的数据结构相较于数组和链表操作更为受限，但是对数据的处理更为可控，在一些问题的处理上具有较为高效的表现。

不足：

1. 缺少对负实数的处理，如-8在程序中需要将输入0-8才能实现正常的数值处理；
2. 运算方式较为有限，仅能实现“+”“-”“\*”“/”的简单四则运算，后续可以加入幂，取整，对数等运算符；
3. 操作仅在实数域内，后续可以引入虚数的简单计算。

**六、附录：源代码（带注释）**

**1. seqstack.h（手动实现顺序栈结构）**

1. #include<iostream>
2. #define SIZE 50
3. using namespace std;
4. template<class T>class Stack
5. {
6. private:
7. T data[SIZE]; *//顺序储存数据*
8. int Top; *//栈顶下标*
9. public:
10. void stack() *//栈的初始化*
11. {
12. Top = -1;
13. }
14. void push(T n) *//压栈*
15. {
16. data[Top + 1]  = n;
17. Top++;
18. }
19. T pop() *//出栈*
20. {
21. if(Top <= -1)
22. abort();
23. T d = data[Top];
24. Top--;
25. return d;
26. }
27. int size() *//查看栈长度*
28. {
29. return Top + 1;
30. }
31. T top() *//读取栈顶元素*
32. {
33. return data[Top];
34. }
35. bool isEmpty() *//判断是否栈空*
36. {
37. if(Top == -1)
38. return true;
39. else
40. return false;
41. }
42. void show() *//打印当前栈中元素*
43. {
44. for(int i = 0; i<=Top; i++)
45. cout<<data[i]<<" ";
46. if(Top < 0)
47. cout<<"栈空";
48. }
49. };

**2.** **main.cpp（具体功能实现）**

1. #include <iostream>
2. #include <fstream>
3. #include "seqstack.h"
4. using namespace std;
5. *//中缀转后缀*
6. void translate(char x[], char y[]);
7. *//后缀的计算*
8. float calculate(char y[]);
9. *//四则运算*
10. float operate(float x, float y, char c)
11. {
12. switch (c)
13. {
14. case '\*': return (x \* y); break;
15. case '/': return (y / x); break;
16. case '+': return (x + y); break;
17. case '-': return (y - x); break;
18. default:  break;
19. }
20. }
21. int main()
22. {
23. char x[SIZE],y[SIZE];
24. ifstream infile;
25. int i = 0;
26. *//文件读取*
27. infile.open("data.txt");
28. while(!infile.eof()) *//判断文件是否读取完毕*
29. {
30. i++;
31. infile.getline(x,SIZE);
32. cout<<"============================================="<<endl;
33. cout<<"读取的第"<<i<<"个中缀表达式为： "<<x<<endl;
34. cout<<"============================================="<<endl;
35. cout<<"转换过程中栈的变化过程为："<<endl;
36. translate(x,y);
37. cout<<"============================================="<<endl;
38. cout<<"后缀表达式为： "<<y<<endl;
39. cout<<"============================================="<<endl;
40. cout<<"计算过程为： "<<endl;
41. float m = calculate(y);
42. cout<<"============================================="<<endl;
43. cout<<"最后中表达式结果为： "<<m<<endl;
44. cout<<"============================================="<<endl;
45. }
46. infile.close();
47. }
48. void translate(char x[], char y[])
49. {
50. Stack<char> s; *//定义运算符栈*
51. s.stack();
52. int i = 0, j = 0,number = 0;
53. *//初始化字符数组*
54. for(int k = 0; k < SIZE; k++)
55. y[k] = ' ';
56. *//逐个扫描*
57. while (x[i] != '\0')
58. {
59. *//数字字符*
60. if (x[i] >= '0' && x[i] <= '9')
61. {
62. while((x[i + 1] >= '0' && x[i + 1] <= '9' ) || (x[i + 1] == '.'))
63. {
64. y[j] = x[i];
65. i++;j++;
66. }
67. y[j] = x[i]; y[j + 1] = ' ';
68. j += 2;
69. }
70. *//运算符处理*
71. if(x[i] == '#')
72. {
73. number++;
74. if(number == 1)
75. s.push(x[i]);
76. *//当第二次遇到#时结束转换*
77. if(number == 2)
78. {
79. while(s.top() != '#')
80. {
81. char x = s.pop();
82. y[j] = x;
83. y[j + 1] = ' ';
84. j += 2;
85. }
86. s.show();cout<<'\n';
87. char x = s.pop();
88. s.show();cout<<'\n';
89. return;
90. }
91. }
92. if (x[i] == '(' )
93. s.push(x[i]);
94. if (x[i] == '\*' || x[i] == '/')
95. {
96. while(s.top() == '\*' || s.top() == '/')
97. {
98. char x = s.pop();
99. y[j] = x;
100. y[j + 1] = ' ';
101. j += 2;
102. }
103. s.push(x[i]);
104. }
105. if (x[i] == '+' || x[i] == '-')
106. {
107. while(s.top() != '#' && s.top() != '(')
108. {
109. char x = s.pop();
110. y[j] = x;
111. y[j + 1] = ' ';
112. j += 2;
113. }
114. s.push(x[i]);
115. }
116. if(x[i] == ')')
117. {
118. while(s.top() != '(')
119. {
120. char x = s.pop();
121. y[j] = x;
122. y[j + 1] = ' ';
123. j += 2;
124. }
125. char q = s.pop();
126. }
127. i++;
128. *//展示当前栈状态*
129. s.show();
130. cout<<'\n';
131. }
132. *// 加入字符串的结尾*
133. y[j+1] = '\0';
134. }
135. float calculate(char y[])
136. {
137. Stack<float> numbers;
138. numbers.stack();
139. int i = 0;
140. while(y[i] != '\0')
141. {
142. *//数字字符处理*
143. if(y[i] >= '0' && y[i] <= '9')
144. {
145. float m = 0;
146. *//整数部分*
147. while(y[i + 1] >= '0' && y[i + 1] <= '9')
148. {
149. m += (y[i] - '0');
150. m \*= 10;
151. i++;
152. }
153. m += (y[i] - '0');
154. *//小数部分*
155. if(y[i + 1] == '.')
156. {
157. i = i + 2;
158. float j = 1;
159. while(y[i + 1] >= '0' && y[i + 1] <= '9')
160. {
161. m += ((float)(y[i] - '0')/(10\*j));
162. j \*= 10;
163. i++;
164. }
165. m += ((y[i] - '0')/(10\*j));
166. }
167. numbers.push(m);
168. }
169. *//运算符处理*
170. if(y[i] == '\*' || y[i] == '/' || y[i] == '+' || y[i] == '-' )
171. {
172. numbers.show();
173. cout<<y[i]<<endl;
174. float x1,x2;
175. x1 = numbers.pop();
176. x2 = numbers.pop();
177. float x = operate(x1,x2,y[i]);
178. numbers.push(x);
179. }
180. i++;
181. }
182. return numbers.pop();
183. }