武汉纺织大学外经贸学院

工程学院

数据结构课程设计报告

2022 ~2023学年第1学期

|  |  |
| --- | --- |
|  |  |
| |  |  | | --- | --- | | **题目** |  | | **班级** |  |  |  |  |  | | --- | --- | --- | |  | 学号 | 姓名 | | **组长** |  |  | | **成员1** |  |  | | **成员2** |  |  | | **成员3** |  |  | | |

目录

[1. 题目与要求 4](#_Toc24554759)

[1.1. 问题描述 4](#_Toc24554760)

[1.2. 本系统涉及的知识点 4](#_Toc24554761)

[1.3. 功能要求 4](#_Toc24554762)

[2. 功能设计 4](#_Toc24554763)

[2.1. 数据结构定义 4](#_Toc24554764)

[2.2. 模块图 4](#_Toc24554765)

[3. 功能代码 4](#_Toc24554766)

[4. 调试与测试 5](#_Toc24554767)

[4.1. 调试分析 5](#_Toc24554768)

[4.2. 用户手册 5](#_Toc24554769)

[4.3. 测试过程 5](#_Toc24554770)

[5. 总结 5](#_Toc24554771)

[6. 参考文献 5](#_Toc24554772)

[7. 附录 6](#_Toc24554773)

[8. 项目分工 7](#_Toc24554774)

# 题目与要求（左对齐，宋体粗四号）

**栈的应用表达式求值**

## 问题描述（左对齐，宋体粗小四号）

输入表达式后，由程序使用逆波兰表达式自动计算，并将结果保留，方便继续计算或让用户提取。

## 本系统涉及的知识点

结构体：将不同的数据类型存储在一起，并分为数据域和指针域，以此形成栈的结构

栈：本质上是操作功能受到限制的单链表，通过出栈入栈的规则限定

## 功能要求

创建栈：初始化栈并赋初值

显示栈中所有信息：从栈顶遍历到栈底打印所有数据域中的信息

判断栈是否为空：检查栈是否为空栈，如果是则不能进行特殊的操作

入栈：将数据压入栈顶

出栈：将栈顶的数据输出

计算：计算表达式

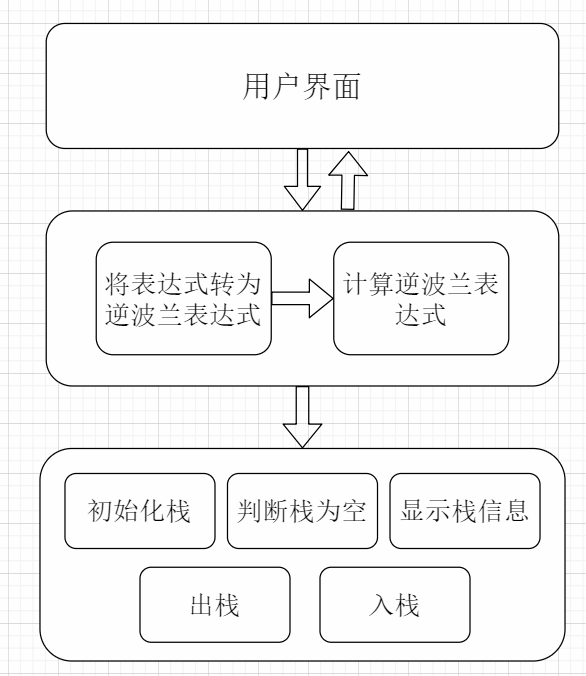
输入表达式：为用户提供界面，使功能明了

# 功能设计

## 数据结构定义

1. **typedef** **int** DataType;   // 定义数据域的数据类型
2. **typedef** **struct** LinkedStack {
3. DataType data;  // 数据域
4. LinkedStack \*next;   // 指针域
5. }LinkedStack;   // 重定义栈结点数据类型

## 2.1模块图



# 算法设计

初始化栈：new一个栈的结点，为其分配空间，并将其中的属性赋初值。

判断栈为空：查看栈下一个结点是否为空为依据，若为空则为空栈。

显示栈信息：使用while循环直到栈为空为止，一次遍历打印数据域的信息，但要保证信息不丢失，即不能以出栈的方式打印信息。

出栈：使用引用传递，将栈顶数据取出，并调整栈顶位置

入栈：使用值传递，向栈顶添加一个数据

表达式转换：现将输入的字符串分成数组，并遍历转换。分为两个栈，一个栈用于存放顺序逆序的表达式，一个栈用于判断运算符的优先级，合并后再调整栈的顺序。

计算逆波兰表达式：依次取出栈顶元素，如果是运算符，则取出两个数进行运算后再返回到栈顶，直到栈为空

# 调试与测试

## 4.1.调试分析

使用两种测试方法，黑盒测试和白盒测试。

白盒测试：先推导出一个具体的表达式的逆波兰形式，通过程序计算后对比是否一致，发生错误时还可以细致到具体每个阶段栈内的信息。

黑盒测试：随意输入一个表达式给程序，查看程序计算的结果是否准确。

## 4.2.用户手册

首先要创建链栈，调用CreateStack()函数。

其次将表达式以字符串的形式传入exchange()函数中，并将返回值直接传入Calculate()函数中。

查看结果可选择使用Pop()函数出栈，也可以使用ShowStack()函数查看。

## 4.3.测试过程





# 总结

功能实现：实现了利用逆波兰表达式计算四则运算，并通过理论分析和实际测试都通过了测试。

功能缺陷：其中只实现了四则运算，逻辑运算和单目运算符没有实现，但运算其本质和四则运算基本一致。

考虑和改进：实现的过程中一开始是使用了对数组赋值的方式进行运算，但这样对用户并不友好，而且繁琐；之后通过读数组自动入栈，但依然需要一些门槛；最后引入了string数据类型，通过读每一位的数据来分割表达式，实现了目前的最优解决。

算法分析：其中最复杂的转换和计算算法都只是遍历一次栈，通过两个栈的配合完成算法，所以总体的时间复杂度为O(n)。

# 参考文献

C++ Primer Plus （第6版）中文版

zhuanlan.zhihu.com/p/346164833

www.bilibili.com/video/BV1xp4y1r7rc/?spm\_id\_from=333.788.recommend\_more\_video.0&vd\_source=8566cec36593b0e28ee03f3c724b87d0

# 附录

1. #include<iostream>
2. #include<string>
3. **using** **namespace** std;
5. **typedef** **int** DataType;
6. **typedef** **struct** LinkedStack {
7. DataType data;
8. LinkedStack \*next;
9. }LinkedStack;
11. LinkedStack\* CreateStack();
12. **void** ShowStack(LinkedStack\* stack);
13. **int** EmptyStack(LinkedStack\* stack);
14. LinkedStack\* Push(LinkedStack\* stack, DataType data);
15. LinkedStack\* Pop(LinkedStack\* stack, DataType\* data);
16. LinkedStack\* Calculate(LinkedStack\* stack);
17. LinkedStack\* exchange(string str);
18. **int** isSymob(**int** content);
20. **int** main() {
21. LinkedStack\* stack = CreateStack();
22. string str;
23. **int** result;
24. // 示例："9+(3-1)\*3+10/2"
25. cout << "请输入计算表达式；";
26. cin >> str;
27. stack = Calculate(exchange(str));
28. stack = Pop(stack, &result);
29. cout << "结果为：" << result << endl;
30. // ShowStack(stack);
32. **return** 0;
33. }
35. **int** isSymob(**int** content) {
36. **switch** (content) {
37. **case** (**int**)'+':
38. **case** (**int**)'-':
39. **return** 1;
40. **break**;
41. **case** (**int**)'\*':
42. **case** (**int**)'/':
43. **case** (**int**)'(':
44. **case** (**int**)')':
45. **return** 2;
46. **break**;
47. **default**:
48. **return** 0;
49. **break**;
50. }
51. }
53. LinkedStack\* exchange(string str) {
54. **int** data[30];
55. **int** i, j = 0;
56. **int** result = 0;
57. **bool** lock = **false**;
58. LinkedStack\* stack = CreateStack();
59. LinkedStack\* temp = CreateStack();
60. **for** (i = 0; i <= str.size(); i++) {
61. **if** (str[i] >= '0' && str[i] <= '9') {
62. result = result \* 10 + (**int**)str[i] - 48;
63. lock = **true**;
64. } **else** {
65. **if** (lock) {
66. data[j] = result;
67. j++;
68. lock = **false**;
69. }
70. data[j] = str[i];
71. j++;
72. result = 0;
73. }
74. }
75. **for** (i = 0; i < j-1; i++) {
76. **if** (!isSymob(data[i])) {
77. temp = Push(temp, data[i]);
78. } **else** {
79. **switch** (data[i]) {
80. **case** (**int**)'+':
81. **case** (**int**)'-':
82. **if** (stack->data == '\*' || stack->data == '/') {
83. **while** (!EmptyStack(stack)) {
84. stack = Pop(stack, &result);
85. temp = Push(temp, result);
86. }
87. }
88. stack = Push(stack, data[i]);
89. **break**;
90. **case** (**int**)'\*':
91. **case** (**int**)'/':
92. **case** (**int**)'(':
93. stack = Push(stack, data[i]);
94. **break**;
95. **case** (**int**)')':
96. **while** (stack->data != (**int**)'(') {
97. stack = Pop(stack, &result);
98. temp = Push(temp, result);
99. }
100. stack = Pop(stack, &result);
101. **break**;
102. **default**:
103. **break**;
104. }
105. }
106. }
107. **while** (!EmptyStack(stack)) {
108. stack = Pop(stack, &result);
109. temp = Push(temp, result);
110. }
111. **while** (!EmptyStack(temp)) {
112. temp = Pop(temp, &result);
113. stack = Push(stack, result);
114. }
115. **return** stack;
116. }
118. LinkedStack\* Calculate(LinkedStack\* stack) {
119. **int** front, end;
120. **int** result;
121. **int** temp;
122. LinkedStack\* resultStack = CreateStack();
123. **while** (!EmptyStack(stack)) {
124. stack = Pop(stack, &temp);
125. **if** (isSymob(temp)) {
126. resultStack = Pop(resultStack, &end);
127. resultStack = Pop(resultStack, &front);
128. **switch** (temp) {
129. **case** (**int**)'+':
130. result = front + end;
131. **break**;
132. **case** (**int**)'-':
133. result = front - end;
134. **break**;
135. **case** (**int**)'\*':
136. result = front \* end;
137. **break**;
138. **case** (**int**)'/':
139. result = front / end;
140. **break**;
141. **default**:
142. **break**;
143. }
144. resultStack = Push(resultStack, result);
145. } **else** {
146. resultStack = Push(resultStack, temp);
147. }
148. }
149. **return** resultStack;
150. }
152. LinkedStack\* CreateStack() {
153. LinkedStack\* head = **new** LinkedStack;
154. head->data = -1;
155. head->next = NULL;
156. **return** head;
157. }
159. **void** ShowStack(LinkedStack\* stack) {
160. **if** (EmptyStack(stack)) {
161. cout << "栈为空" << endl;
162. } **else** {
163. LinkedStack\* work;
164. work = stack;
165. **while** (work->next != NULL) {
166. cout << work->data << endl;
167. work = work->next;
168. }
169. }
170. }
172. **int** EmptyStack(LinkedStack\* stack) {
173. **if** (stack->next == NULL) {
174. **return** 1;
175. } **else** {
176. **return** 0;
177. }
178. }
180. LinkedStack\* Push(LinkedStack\* stack, DataType data) {
181. LinkedStack\* newTop = **new** LinkedStack;
182. newTop->data = data;
183. newTop->next = stack;
184. **return** newTop;
185. }
187. LinkedStack\* Pop(LinkedStack\* stack, DataType\* data) {
188. **if** (EmptyStack(stack)) {
189. cout << "栈为空" << endl;
190. **return** stack;
191. } **else** {
192. \*data = stack->data;
193. LinkedStack\* newTop = stack->next;
194. stack->next = NULL;
195. **return** newTop;
196. }
197. }

# 8.项目分工（每组不超过4人）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **成员姓名** | **承担任务** | **完成情况** | **贡献比例（%）** |
| **1** |  |  |  |  |
| **2** |  |  |  |  |
| **3** |  |  |  |  |
| **4** |  |  |  |  |

小组成员签名：

日 期：