山东大学 计算机科学与技术 学院

数据结构与算法 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：202000130198 | 姓名： 隋春雨 | | 班级： 20.4 |
| 实验题目：栈 | | | |
| 实验学时：2 | | 实验日期： 2021-11-4 | |
| 实验目的：  1、 掌握栈结构的定义与实现；  2、 掌握栈结构的使用。 | | | |
| 软件开发环境：  CLION2020 | | | |
| 1. 实验内容   题目描述：  创建栈类，采用数组描述；计算数学表达式的值。 输入数学表达式，输出表达式的计算结果。数学表达式由单个数字和运算符“+”、“-”、“\*”、“/”、“(”、“) ”构成，例如 2+3\*(4+5)–6/4。  输入输出格式：  输入：  第一行一个整数n(1<=n<=100)，代表表达式的个数。  接下来n行，每行一个表达式，保证表达式内的数字为单个整数，表达式内各运算符和数字间没有空格，且表达式的长度不超过2000。  输出：  每行表达式输出一个浮点数，要求保留两位小数，保证输入表达式合法。   1. 数据结构与算法描述 （整体思路描述，所需要的数据结构与算法） 2. 算法：首先我们知道，对于计算机来说，中缀表达式是一个很复杂的结构，它读取到一个符号，不知道该不该进行运算，因为它后面的数据我们不知道，比如说3\*（6\*（6/7\*（4+2））），进行判断及其的麻烦，所以我们采取后缀表达式（逆波兰式），中缀表达式是相对人类的思维结构来说比较简单的，对计算机而言中序表达式是非常复杂的结构。相对的，逆波兰式在计算机看来却是比较简单易懂的结构。因为计算机普遍采用的内存结构是栈式结构，它执行先进后出的顺序。首先用一个string来存储这个表达式，如果读取到数字，那么直接压入数字stack。如果读取到了符号，若是(，那么直接压入，因为我们知道(在读取到的时候，它的优先级是最高的。     如果读取到了)，那么stack一直pop，直到pop到了（。如果都不是，那么比较优先级。优先处理优先级高的符号，如果优先级相同（比如\* /）,那么执行从左到右的顺序。值得注意的是，我们需要用一个函数来表示优先级，否则使用起来太麻烦了 。    当结束的时候，因为符号stack可能还有符号，那么我们直接一个一个pop，并且计算即可     1. 数据结构：我们在1中的算法中提及到了后进先出，那么我们很自然的想到了栈的这种存储结构        1. 测试结果（测试输入，测试输出）   输入：    输出：    提交OJ最后的结果：     1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径） 2. 对于循环，我们一定一定要防止RE，比如下面这组代码，我们可能最开始想的是要弹走栈顶，但是我们需要知道的是，弹走栈顶一定要保证非空，否则最后会导致数组下标越界， 3. 对于特殊情况一定要特殊讨论，比如我们计算一个括号中的表达式，应该考虑一下，栈顶是否为(，以及栈顶为(的时候，它的优先级是最高还是最低？经过我们的思考，我们得出结论，此时的优先级为最低，同时我们需要考虑先计算优先级高的，如果优先级相同，那么从左到右计算      1. 对于边界条件的处理也一定要注意，这个string最后一定stack非空，那么我们需要边界处理一下，计算表达式      1. 值得注意的是，一定一定要记得更新私有成员，因为public接口的函数是没有记忆性的，只有靠着私有成员才能判断自身的状态。并且不要硬编码，如果硬编码的话，写的时间会很长，并且写代码出bug的几率也会更大，所以我们一定要先想好思路再开始，否则会很麻烦，比如下图     如果是暴力解决也可以硬枚举出来，但是用函数可以封装的更好更优雅，如果出了Bug改起来更快，也更容易。所以更推荐用函数解决。   1. 计算表达式的时候，一定要分清谁去计算谁，比如下图，也反映了我们计算的时候思路清不清晰，要先有思路，再有代码      1. 自己写的时候测的样例都是对的，交到oj平台上就RE了，怎么办？   解决：RE常见情况的是数组下标越界，但是经过自己debug发现，实际情况是switch case条件没有break语句，才RE，在平时，能用switch case尽量用switch case而不是If else ，因为switch case执行的次数少。   1. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释） 2. #include<iostream> 3. #include <sstream> 4. #include<string> 5. #include<iomanip> 6. #include<stack> 7. **using** **namespace** std; 8. **template**<**class** T> 9. **class** arrayStack 10. { 11. **public**: 12. arrayStack(**int** initialCapacity = 10);//初始化 13. ~arrayStack() { **delete**[] stack; }//析构 14. **bool** empty() **const** { **return** stackTop == -1; } 15. **int** size() **const**//size函数 16. { 17. **return** stackTop + 1; 18. } 19. **bool** empty()//判断是否为空 20. { 21. **return** stackTop == -1; 22. } 23. **void** checkEmpty() 24. { 25. **if** (empty()) 26. { 27. **throw** "the stack is empty"; 28. } 29. } 30. T& top()//返回栈顶 31. { 32. checkEmpty(); 33. **return** stack[stackTop]; 34. } 35. **void** pop()//弹出栈顶 36. { 37. checkEmpty(); 38. stackTop--; // destructor for T 39. } 40. **void** push(**const** T& theElement); 42. **private**: 43. **int** stackTop;         // current top of stack 44. **int** arrayLength;      // stack capacity 45. T\* stack;           // element array 47. }; 49. **template**<**class** T> 50. arrayStack<T>::arrayStack(**int** initialCapacity) 51. { 52. arrayLength = initialCapacity; 53. stack = **new** T[arrayLength]; 54. stackTop = -1;//表示栈顶的位置，并且-1也表示空 55. } 57. **template**<**class** T> 58. **void** arrayStack<T>::push(**const** T& theElement) 59. { 60. **if** (stackTop == arrayLength - 1)//如果满了，那么扩容 61. { 62. T\* temp = **new** T[2 \* arrayLength]; 63. copy(stack, stack + arrayLength, temp); 64. **delete**[]stack; 65. stack = temp; 66. arrayLength \*= 2;//更新私有变量 67. } 68. stack[++stackTop] = theElement; 69. }  72. **int** level(**const** **char**& c)//用来计算等级的，也就是优先级 73. { 74. **switch** (c) 75. { 76. **case** '+': 77. **case**'-': 78. **return** 0; 79. **case**'\*': 80. **case**'/': 81. **return** 1; 82. **default**: 83. **return** 0; 84. } 85. } 87. **double** caculate(**double**& a, **double**& b, **char**& c)//计算表达式 88. { 89. **switch** (c) 90. { 91. **case**'+': 92. **return** a + b; 93. **case**'-': 94. **return** a - b; 95. **case**'\*': 96. **return** a \* b; 97. **case**'/': 98. **return** a / b;//一定一定要注意除法的使用，因为a/b!=b/a 99. **default**: 100. **throw** "Error"; 101. } 102. }  105. **int** main() 106. { 108. **int** n; 109. cin >> n; 110. **for** (**int** j = 0; j < n; j++) 111. { 112. string str; 113. cin >> str; 114. arrayStack<**double**>num;//创建两个stack 115. arrayStack<**char**>ope;//符号stack 116. **for** (**int** i = 0; i < str.size(); i++) 117. { 118. **if** (str[i] >= '0' && str[i] <= '9')//数字 119. { 120. num.push(**double**(str[i] - '0'));//数字直接push 121. } 122. **else** 123. { 124. **if** (str[i] == '(') 125. { 126. ope.push(str[i]);//(直接push 127. } 128. **else** **if** (str[i] == ')') 129. { 130. **while** (!ope.empty() && ope.top() != '(')//探出并且计算 131. { 132. **char** c = ope.top(); 133. ope.pop(); 134. **double** num2 = num.top(); 135. num.pop(); 136. **double** num1 = num.top(); 137. num.pop(); 138. num.push(caculate(num1, num2, c)); 139. } 140. ope.pop();//弹掉( 141. } 142. **else** 143. {//+ - \* / 144. **if** (ope.empty() || ope.top() == '(')//如果是空，或者栈顶是(那么直接压入 145. { 146. ope.push(str[i]); 147. } 148. **else** 149. { 150. **while** (!ope.empty() && ope.top() != '(' && level(ope.top()) >= level(str[i])) 151. {//找到比它优先级低的，或者最后是空 152. **char** c = ope.top(); 153. ope.pop(); 154. **double** num2 = num.top(); 155. num.pop(); 156. **double** num1 = num.top(); 157. num.pop(); 158. num.push(caculate(num1, num2, c)); 159. } 160. ope.push(str[i]); 161. } 163. } 164. } 165. } 166. **while** (!ope.empty())//最后的处理 167. { 168. **char** c = ope.top(); 169. ope.pop(); 170. **double** num2 = num.top(); 171. num.pop(); 172. **double** num1 = num.top(); 173. num.pop(); 174. num.push(caculate(num1, num2, c)); 175. } 176. cout << fixed << setprecision(2) << num.top() << endl;//输出两位 178. }   182. **return** 0; 183. } | | | |