**数据结构实验报告**

|  |  |  |  |
| --- | --- | --- | --- |
| 学号-姓名 | 桑龙龙-20030540015 | 实验时间 | 2020年 10月 17日 |
| 诚信声明 | 本实验及实验报告所写内容为本人所作，没有抄袭。 | | |
| 实验题目 | 实验二 栈和队列的实现与应用题目  题目一 数制转换  题目二 括号匹配问题  题目三 停车场管理  题目四 迷宫问题 | | |
| 实验过程中遇到的主要问题 | 在实现链栈的过程中，考虑如何使链栈的实现更为实用，最后通过将链栈封装到类中。 | | |
| 实验小结 | 本次实验进行了队列和栈的应用，通过这次实验，对递归理解更为深刻，我们通过递归可以使代码看起来更为简洁，这是通过系统栈帮助我们实现的，当我自己将以前以递归方法写的代码改成非递归方法时，需要自己去注意很多细节，如何标记一个方法的运行过程，如何判断进栈出栈条件，学习到很多。 | | |
| 数据结构  （自定义数据类型） | 1. **struct** node{ 2. **int** x, y , i; 3. //x,y表示坐标 4. //i表示要递归哪个方向，初始i为1 5. //当i为5的时候表示4个方向已经递归过 6. node(**int** x, **int** y, **int** i){ 7. **this**->x = x; 8. **this**->y = y; 9. **this**->i = i; 10. } 11. }; 12. **class** linkstack{ 13. //链栈 14. **struct** mynode{ 15. **int** x, y , i; 16. mynode\* next; 17. mynode(**int** x, **int** y, **int** i){ 18. **this**->x = x; 19. **this**->y = y; 20. **this**->i = i; 21. **this**->next = NULL; 22. } 23. mynode(**const** node& a){ 24. mynode(a.x, a.y, a.i); 25. } 26. }; 27. **private**: 28. **int** mysize; 29. mynode\* head; 30. **public**: 31. linkstack(){ 32. mysize = 0; 33. head = NULL; 34. } 35. **void** push(node& a){ 36. mynode\* temp = **new** mynode(a.x, a.y, a.i); 37. mysize++; 38. **if**(head == NULL){ 39. head = temp; 40. }**else**{ 41. temp->next = head; 42. head = temp; 43. } 44. } 45. node top(){ 46. node ret(head->x, head->y, head->i); 47. **return** ret; 48. } 49. **void** pop(){ 50. **if**(!head) **return**; 51. mynode\* temp = head->next; 52. **delete** head; 53. head = temp; 54. mysize--; 55. } 56. **int** size(){ 57. **return** mysize; 58. } 59. **bool** empty(){ 60. **return** mysize == 0; 61. } 62. }; | | |
| 主要算法  （或算法说明） | 1. //1、数制转换 2. #include<stdlib.h> 3. #include <stdio.h> 4. **int** stack[30]; 5. //栈 6. **void** d2b(**long** **long** ori); 7. //十进制转二进制 9. **int** main(){ 10. **long** **long** ori; 11. scanf("%lld\n",&ori); 12. d2b(ori); 13. **return** 0; 14. } 15. **void** d2b(**long** **long** ori){ 16. //十进制转二进制 17. **if**(ori == 0) 18. {//如果ori为0 20. putchar('0'); 21. **return**; 22. }**else** **if**(ori < 0) 23. {//如果ori为负数 25. putchar('-'); 26. ori = -ori; 27. } 28. **int** top=0; 29. **while**(ori){ 30. stack[top++] = ori % 2; 31. ori /= 2; 32. } 33. **while**(top){ 34. putchar('0' + stack[(top--) - 1]); 35. } 36. } 37. //1、数制转换 结束 38. //4、迷宫问题 39. #include<stdlib.h> 40. #include <stdio.h> 41. #define N 256 42. /\* 43. 使用语言:C++ 44. 输入格式 45. 第一行两个空格分隔的正整数r，c 46. 随后r行，每行c个由01组成的字符 47. 例： 48. 5 5 49. 01000 50. 00100 51. 10000 52. 11110 53. 00000 54. 注释：0表示可以走，1表示不可以走 55. \*/ 56. **int** dir[5][2]={0,0,1,0,0,1,-1,0,0,-1}; 57. //方向数组dir[1 2 3 4]表示向下右上左四个方向 58. **int** r,c; 59. //表示图的行数和列数 60. **bool** grid[N][N]; 61. //表示一个r\*c的10地图 62. **bool** vis[N][N]; 63. //访问标记 64. **char** path[N][N]; 65. //路径记录 66. **class** linkstack; 67. //链栈 68. **struct** node; 69. **void** pri(**int** i);   73. **int** main(){ 74. //输入部分 75. scanf("%d %d\n", &r, &c); 76. **char** a; 77. **for**(**int** i = 0; i < r ; i++){ 78. **for**(**int** j = 0; j < c; j++){ 79. scanf("%c", &a); 80. grid[i][j] = a == '1' ? **false** : **true**; 81. } 82. getchar(); 83. } 84. //主程序部分 85. //如果入口grid[0][0]处就是1，表示被堵死 86. **if**(!grid[0][0]){ 87. printf("end\n"); 88. **return** 0; 89. } 90. linkstack S; 91. //声明链栈 92. node cur(0, 0, 1); 93. S.push(cur); 94. **bool** find\_path = **false**; 95. **while**(!S.empty()){ 96. //模拟递归 97. cur = S.top(); 98. S.pop(); 99. **if**(cur.i == 5){ 100. //(cur.x,cur.y)坐标递归完成 101. //并将其访问标记重新标记为未访问 102. vis[cur.x][cur.y] = **false**; 103. **continue**; 104. }**else**{ 105. //标记(cur.x,cur.y)被访问标记 106. vis[cur.x][cur.y] = **true**; 107. } 108. **if**(cur.x == r - 1 && cur.y == c -1 ){ 109. //到达终点（右下角），停止循环 110. find\_path = **true**; 111. **break**; 112. } 113. cur.i++; 114. S.push(cur);//重新入站 115. cur.i--; 116. //next\_x和next\_y下一个要访问的点 117. **int** next\_x = dir[cur.i][0] + cur.x; 118. **int** next\_y = dir[cur.i][1] + cur.y; 119. **if**(next\_x >= r || next\_x < 0 || next\_y >= c || next\_y < 0) **continue**; 120. **if**(vis[next\_x][next\_y] || !grid[next\_x][next\_y]) **continue**; 121. //如果next\_x,next\_y出了grid的范围 122. //或者本身是一堵墙（即grid[x][y]==false）或者被访问过 123. path[next\_x][next\_y] = cur.i; 124. //记录路径 125. node next(next\_x, next\_y, 1); 126. S.push(next); 127. } 128. **while**(!S.empty()) S.pop(); 129. cur = {r-1, c-1, path[r-1][c-1]}; 130. **while**(cur.x != 0 || cur.y != 0){ 131. S.push(cur); 132. **int** x = cur.x, y = cur.y; 133. cur.x = x - dir[cur.i][0]; 134. cur.y = y - dir[cur.i][1]; 135. cur.i = path[cur.x][cur.y]; 136. } 137. S.push(cur); 138. **while**(!S.empty()){ 139. cur = S.top(); 140. S.pop(); 141. **if**(!S.empty()){ 142. printf("%d %d ", cur.x, cur.y); 143. pri(S.top().i); 144. } 145. **else** printf("%d %d end\n", cur.x, cur.y); 146. } 147. **return** 0; 148. } 149. **void** pri(**int** i){ 150. **if**(i == 1) printf("down\n"); 151. **else** **if**(i == 2) printf("right\n"); 152. **else** **if**(i == 3) printf("up\n"); 153. **else** printf("left\n"); 154. } 155. **struct** node{ 156. **int** x, y , i; 157. //x,y表示坐标 158. //i表示要递归哪个方向，初始i为1 159. //当i为5的时候表示4个方向已经递归过 160. node(**int** x, **int** y, **int** i){ 161. **this**->x = x; 162. **this**->y = y; 163. **this**->i = i; 164. } 165. }; 166. **class** linkstack{ 167. **struct** mynode{ 168. **int** x, y , i; 169. mynode\* next; 170. mynode(**int** x, **int** y, **int** i){ 171. **this**->x = x; 172. **this**->y = y; 173. **this**->i = i; 174. **this**->next = NULL; 175. } 176. mynode(**const** node& a){ 177. mynode(a.x, a.y, a.i); 178. } 179. }; 180. **private**: 181. **int** mysize; 182. mynode\* head; 183. **public**: 184. linkstack(){ 185. mysize = 0; 186. head = NULL; 187. } 188. **void** push(node& a){ 189. mynode\* temp = **new** mynode(a.x, a.y, a.i); 190. mysize++; 191. **if**(head == NULL){ 192. head = temp; 193. }**else**{ 194. temp->next = head; 195. head = temp; 196. } 197. } 198. node top(){ 199. node ret(head->x, head->y, head->i); 200. **return** ret; 201. } 202. **void** pop(){ 203. **if**(!head) **return**; 204. mynode\* temp = head->next; 205. **delete** head; 206. head = temp; 207. mysize--; 208. } 209. **int** size(){ 210. **return** mysize; 211. } 212. **bool** empty(){ 213. **return** mysize == 0; 214. } 215. }; 216. //4、迷宫问题 结束 | | |