**作业 HW2 实验报告**

姓名：何正潇 学号：1950095日期：2021年10月28日

1. **涉及数据结构和相关背景**

**主要涉及栈和队列的使用，其中包含基本的操作以及一些栈和队列在解决实际数学问题的应用。**

**2. 实验内容**

**2.1 栈的基本操作**

**2.1.1 问题描述**

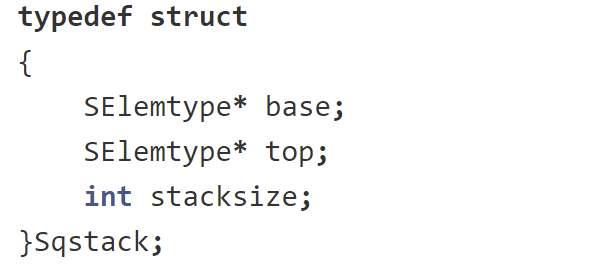
栈是限制仅在表的一端插入和删除的线性表。栈的操作简单，重点掌握栈具有后进先出（LIFO）的特性。顺序栈是栈的顺序存储结构的实现。链栈是栈的链式存储结构的实现。 本题练习顺序栈的基本操作，包括入栈、出栈、判栈空、判栈满、取栈顶元素、栈的遍历。**2.1.2 基本要求**

完成栈的建立，了解栈LIFO的特点。

进行栈的基本操作，例如出栈入栈等操作

进行栈的遍历操作

**2.1.3 数据结构设计**

**2.1.4功能说明（函数、类）**

1. /\*栈的初始化程序\*/
2. **int** InitStack**(**Sqstack**&** s**,int** n**)**
3. {
4. s**.**base **=** **(**SElemtype**\*)**malloc**(**n **\*** **sizeof(**SElemtype**));**
5. **if** **(!**s**.**base**)**
6. **return** **-**1**;**
7. s**.**top **=** s**.**base**;**
8. s**.**stacksize **=** 100**;**
9. **return** 0**;**
10. }
11. /\*出栈操作\*/
12. **int** Pop **(**Sqstack**&** s**)**
13. {
14. **if** **(**s**.**top **==** s**.**base**)**
15. {
16. cout **<<** "Stack is Empty" **<<** endl**;**
17. **return** **-**1**;**
18. }
19. **else**
20. {
21. cout **<<** **\*--**s**.**top **<<** endl**;**
22. **return** 0**;**
23. }
24. }
25. /\*入栈操作\*/
26. **int** push**(**Sqstack**&** s**,** SElemtype e**,int** n**)**
27. {
28. **if** **(**s**.**top **-** s**.**base **>=**n**)**
29. {
30. cout **<<** "Stack is Full" **<<** endl**;**
31. **return** **-**1**;**
32. }
33. **\***s**.**top**++** **=** e**;**
34. **return** 0**;**
35. }
36. /\*栈中全部元素压出操作\*/
37. **void** exit\_push\_all**(**Sqstack**&** s**)**
38. {
39. **if** **(**s**.**top **!=** s**.**base**)**
40. {
41. **while** **(**s**.**top **!=** s**.**base**)**
42. {
43. cout **<<** **\*--**s**.**top **<<** " "**;**
44. }
45. }
46. **else**
47. {
48. cout **<<** "Stack is Empty" **<<** endl**;**
49. }
50. }

**2.1.5 调试分析（遇到的问题和解决方法）**

此题整体来说比较简单。基本一遍就完成AC。

**2.1.6 总结和体会**

我在本题基本了解了栈的基本操作和栈的特点，实现了数据结构动手操作的目标。不同于很多同学直接使用stl容器，我在数据结构中基本都是代码实现数据结构的操作，因此对于数据结构有更深的理解。

**2.2 列车进站**

**2.2.1 问题描述**

每一时刻，列车可以从入口进车站或直接从入口进入出口，再或者从车站进入出口。即每一时刻可以有一辆车沿着箭头a或b或c的方向行驶。 现在有一些车在入口处等待，给出该序列，然后给你多组出站序列，请你判断是否能够通过上述的方式从出口出来。

**2.2.2 基本要求**

**利用栈数据结构解决这个实际的问题。锻炼运用数据结构在实际问题的能力。**

**2.2.3 数据结构设计**

1. **typedef** **struct** S
2. {
3. SElemType**\*** base**;**
4. SElemType**\*** top**;**
5. **int** stacksize**;**
6. }Sqstack**;**

**2.2.4功能说明（函数、类）**

**由于这里的函数与上题相似，因此直接贴出了源代码**

1. **int** InitStack**(**Sqstack**&** s**)**
2. {
3. s**.**base **=** **(**SElemType**\*)**malloc**(**100 **\*** **sizeof(**SElemType**));**
4. **if** **(!**s**.**base**)**
5. **return** **-**1**;**
6. s**.**top **=** s**.**base**;**
7. s**.**stacksize **=** 100**;**
8. **return** 0**;**
9. }//初始化栈
10. **int** Pop**(**Sqstack**&** s**)**
11. {
12. **if** **(**s**.**top **==** s**.**base**)**
13. {
14. **return** **-**1**;**
15. }
16. **else**
17. {
18. **--**s**.**top**;**
19. **return** 0**;**
20. }
21. }//弹出栈的元素
22. **int** push**(**Sqstack**&** s**,** SElemType e**,** **int** n**)**
23. {
24. **if** **(**s**.**top **-** s**.**base **>=** n**)**
25. {
26. s**.**base **=** **(**SElemType**\*)**realloc**(**s**.**base**,** **(**s**.**stacksize **+** 10**)** **\*** **sizeof(**SElemType**));**
27. **if** **(!**s**.**base**)**
28. **return** **-**1**;**
29. **else**
30. {
31. s**.**stacksize **=** s**.**stacksize **+** 10**;**
32. }
33. }
34. **\***s**.**top**++** **=** e**;**
35. **return** 0**;**
36. }//压入栈元素
37. **void** exit\_push\_all**(**Sqstack**&** s**)**
38. {
39. **if** **(**s**.**top **!=** s**.**base**)**
40. {
41. **while** **(**s**.**top **!=** s**.**base**)**
42. {
43. **--**s**.**top**;**
44. }
45. }
46. }//将栈中元素全部压出
47. **int** main**()**
48. {
49. Sqstack in**,** station**;**
50. InitStack**(**in**),** InitStack**(**station**);**
51. **char** temp**[**30**];**
52. cin **>>** temp**;**
54. **bool** judge **=** 1**;**
55. **char** out**[**30**];**
56. **int** i **=** 0**;**
57. /\*倒序压入栈，如果In栈顶元素和out数组的目前元素相同，则将In中元素压出，如果station栈顶元素和当前out元素相同，则从station栈顶将当前元素压出，否则的话将In头元素压入station栈中，直到In栈空，如果station空栈，则输出yes，否则则将station进行出栈，如果出栈元素和out数组当前元素不同，输出no，如果能进行到station栈空，则输出yes\*/
58. **while** **(**1**)**
59. {
60. cin **>>** out**;**
61. **if** **(!**cin**.**eof**())**
62. {
63. **for** **(int** k **=** 0**;** k **<** **int(**strlen**(**temp**));** k**++)**
64. push**(**in**,** temp**[**strlen**(**temp**)** **-** 1 **-** k**],** strlen**(**temp**));**
65. **while** **(**in**.**base **!=** in**.**top**)**
66. {
68. **if** **(**in**.**top **!=** in**.**base **&&** **\*(**in**.**top **-** 1**)** **==** out**[**i**])**
69. {
70. Pop**(**in**);**
71. i**++;**
72. }
73. **else**
74. {
75. **if** **(\*(**station**.**top **-** 1**)** **==** out**[**i**])**
76. {
77. Pop**(**station**);**
78. i**++;**
79. }
80. **else**
81. {
82. push**(**station**,** **\*(**in**.**top **-** 1**),** 30**);**
83. Pop**(**in**);**
84. }

87. }
88. }
89. **if** **(**station**.**top **==** station**.**base**)**
90. judge **=** 1**;**
91. **while** **(**station**.**top **!=** station**.**base**)**
92. {
93. **if** **(**station**.**top **!=** station**.**base **&&** **\*(**station**.**top -1**)** **==** out**[**i**])**
94. {
95. Pop**(**station**);**
96. i**++;**
97. }
98. **else**
99. {
100. judge **=** 0**;**
101. **break;**
102. }
103. judge **=** 1**;**
104. }
105. exit\_push\_all**(**station**);**
106. **if** **(**judge **==** 1**)**
107. cout **<<** "yes" **<<** endl**;**
108. **else**
109. cout **<<** "no" **<<** endl**;**
110. judge **=** 0**;**
111. i **=** 0**;**
112. }
113. **else**
114. {
115. **break;**
116. }
117. }
119. **return** 0**;**
121. }

**2.2.5 调试分析（遇到的问题和解决方法）**

**这题我认为一开始的逻辑分析是一个比较复杂的地方。需要大概了解需要使用几个栈模拟过程，同时需要分析栈的进出栈实际情况。**

**2.2.6 总结和体会**

**此题我基本掌握了栈在实际问题的情况，和具体情景情况的分析能力。总体来说，对于我们的逻辑分析能力和对于数据结构的熟悉程度很有帮助。**

**2.3 布尔表达式**

**2.3.1问题描述**

计算如下布尔表达式 ( V | V ) & F & ( F | V ) 其中V表示True，F表示False，|表示or，&表示and，！表示not（运算符优先级not> and > or）、

**2.3.2基本要求**

**用栈进行逻辑运算，掌握栈实际上的一个重要应用**

**2.3.3 数据结构设计**

1. typedef char SElemType;
2. typedef struct {
3. SElemType\* base; //栈底指针
4. SElemType\* top; //栈顶指针
5. int stacksize; //当前已分配的存储空间，以元素为单位
6. }SqStack;
8. //构造一个空栈S
9. 基本也就是一个正常的栈结构

**2.3.4功能说明（函数、类）**

**//构造一个空栈S**

**void InitStack(SqStack& S)**

**{**

**S.base = new(nothrow) SElemType[STACK\_INIT\_SIZE];**

**if (S.base == NULL) //存储分配失败**

**return;**

**S.top = S.base;**

**S.stacksize = STACK\_INIT\_SIZE;**

**}**

**//删除S栈顶元素,返回给e**

**void Pop(SqStack& S, SElemType& e)**

**{**

**if (S.top == S.base)**

**return;**

**else**

**e = \*(--S.top);**

**}**

**//插入元素e作为栈顶的新元素**

**void Push(SqStack& S, SElemType e)**

**{**

**if (S.top - S.base >= S.stacksize)**

**return;**

**else**

**\*S.top++ = e;**

**}**

**//比较栈顶元素和新传入的运算符的优先级**

**char Precede(char first, char next)**

**{**

**if (first == '(' && next == ')') //双边括号情况**

**return '=';**

**else if ((first == '|' && (next == '&' || next == '!')) || (first == '&' && next == '!') || first == '#' || first == '(' || next == '('**

**|| (first == '!' && next == '!')) //注意**

**return '<';**

**else**

**return '>';**

**}**

**/\*operate运算函数 这里我考虑不能加入！，难以兼容\*/**

**char Operate(char a, char optr, char b)**

**{**

**if (optr == '|')**

**return (a - '0' || b - '0') + '0';**

**else**

**return (a - '0' && b - '0') + '0';**

**}**

**int main()**

**{**

**SqStack OPTR, OPND;**

**InitStack(OPTR); //寄存运算符**

**Push(OPTR, '#'); //运算栈底的标志**

**InitStack(OPND); //寄存操作数或运算结果**

**int i = 1;**

**while (1) {**

**char signal = getchar();**

**if (signal== EOF)//文件读取结束**

**break;**

**/\*只要运算符还没空以及字符还没输完执行循环\*/**

**while (\*(OPTR.top - 1) != '#' || signal != '\r') {**

**if (signal == ' ') {**

**signal = getchar();**

**continue;**

**}//适应文件中可怕的随机空格**

**else if (signal == 'V' || signal == 'F') {**

**if (signal == 'V')**

**Push(OPND, '1');**

**else**

**Push(OPND, '0');**

**signal = getchar();**

**}**

**else {**

**char pop\_out;**

**char x, y;**

**char optr; //运算符**

**switch (Precede(\*(OPTR.top - 1), signal)) {**

**case '<': //栈顶元素优先级底**

**Push(OPTR, signal);//压入**

**signal = getchar();**

**break;**

**case '=': //脱去括号**

**Pop(OPTR, pop\_out);**

**signal = getchar();**

**break;**

**case '>': //退栈并将运算结果入栈**

**Pop(OPTR, optr);**

**Pop(OPND, x);**

**if (optr != '!') {**

**Pop(OPND, y);**

**Push(OPND, Operate(x, optr, y));**

**}**

**else**

**Push(OPND, x == '1' ? '0' : '1');**

**break;**

**}**

**}**

**}**

**getchar();**

**char result;**

**Pop(OPND, result);**

**cout << "Expression " << i++ << ": " << (result == '1' ? 'V' : 'F') << endl;**

**}**

**return 0;**

**}**

**2.3.5 调试分析（遇到的问题和解决方法）**

**这题总体来说比较复杂的地方在于对于！的单独分析，其他的地方不太容易错误。基本就是模拟逻辑运算的一道习题。**

**2.3.6 总结和体会**

**我在这题中掌握了栈在逻辑运算中的使用案例，对栈有了更深层次的理解，基本了解了栈的应用场景。**

**2.4 队列的应用**

**2.4.1 问题描述**

输入一个n\*m的0 1矩阵，1表示该位置有东西，0表示该位置没有东西。所有四邻域联通的1算作一个区域，仅在矩阵边缘联通的不算作区域。求区域数。此算法在细胞计数上会经常用到。

对于所有数据，0<=n,m<=1000

**2.4.2题目要求**

**这道题基本考察的是一个队列的基本操作，广度搜索，使用广度搜索以后这道题可以较为简单的完成。**

**2.4.3数据结构**

**typedef int QElemtype;**

**typedef int Status;**

**typedef struct Node {**

**QElemtype x;**

**QElemtype y;**

**struct Node\* next;**

**}QNode;**

**typedef struct Queue {**

**QNode\* front; //队首指针**

**QNode\* rear; //队尾指针**

**}LinkQueue;**

**2.4.4功能说明（函数、类）**

**/\*建立队列\*/**

**Status InitQueue(LinkQueue& Q)**

**{**

**Q.front = Q.rear = (QNode\*)malloc(sizeof(QNode));**

**if (Q.front == NULL)**

**return -1;**

**Q.front->next = NULL;**

**return 0;**

**}**

**/\*摧毁队列\*/**

**Status DestroyQuene(LinkQueue& Q)**

**{**

**while(Q.front)**

**{**

**Q.rear = Q.front->next;**

**free(Q.front);**

**Q.front = Q.rear;**

**}**

**return 0;**

**}**

**/\*实现入队操作\*/**

**Status Enquene(LinkQueue& Q, QElemtype x,QElemtype y)**

**{**

**QNode\*p = (QNode\*)malloc(sizeof(QNode));**

**if (p == NULL)**

**return -1;**

**p->x = x;**

**p->y = y;**

**p->next = NULL;**

**Q.rear->next = p;**

**Q.rear = p;**

**return 0;**

**}**

**/\*实现出队操作\*/**

**Status Dequene(LinkQueue& Q,QElemtype& x,QElemtype& y)**

**{**

**if (Q.front == Q.rear)**

**return -1;**

**QNode\* p = Q.front->next;**

**Q.front->next = p->next;**

**x = p->x, y = p->y;**

**if (Q.rear == p)**

**Q.rear = Q.front;**

**free(p);**

**return 0;**

**}**

**/\*广度搜索代码\*/**

**void BFS(LinkQueue& Q, int \*\*data, int\*\* visited,int m, int n)**

**{**

**int num = 0;**

**QElemtype x, y;**

**for (int i=1;i<m-1;i++)**

**for (int j = 1; j < n-1; j++)**

**{**

**if (data[i][j] == 1 && visited[i][j] == 0)**

**{**

**Enquene(Q, i, j);**

**visited[i][j] = 1;**

**while (Q.front != Q.rear)**

**{**

**QElemtype change\_x[4] = { 0,0,-1,1 }; //表示上下左右的坐标**

**QElemtype change\_y[4] = { -1,1,0,0 };**

**Dequene(Q, x, y);**

**for (int k = 0; k < 4; k++) {**

**if (x + change\_x[k] < 0 || x + change\_x[k] >= m ||y + change\_y[k] < 0 ||y + change\_y[k] >= n) //超过边界则跳过**

**continue;**

**if (visited[x + change\_x[k]][y + change\_y[k]] ==1|| data[x + change\_x[k]][y + change\_y[k]] == 0) //已经标记过或者当前数据不是1则跳过**

**continue;**

**visited[x+change\_x[k]][y + change\_y[k]] = 1;**

**Enquene(Q, x + change\_x[k], y + change\_y[k]); //如果为1则把当前坐标加入队列**

**}**

**}**

**num++;**

**}**

**}**

**cout << num;**

**}**

**限于界面宽度上面的代码有些杂乱，请老师见谅。**

**2.4.5 调试分析（遇到的问题和解决方法）**

**基本就是广度搜索一开始写的逻辑有些疏漏，不过很快就就调试完成。**

**2.4.6 总结和体会**

**掌握了队列的一个重要应用，同时课外看了一下深度搜索的代码，这道题具有实际的应用场景，也比较具有趣味性。**

**2.5 队列中的最大值问题**

**2.5.1 问题描述**

给定一个队列，有下列3个基本操作：

（1）Enqueue(v): v 入队

（2）Dequeue(): 使队首元素删除，并返回此元素

（3）GetMax(): 返回队列中的最大元素

请设计一种数据结构和算法，让GetMax操作的时间复杂度尽可能地低。

要求运行时间不超过一秒

**2.5.2题目要求**

**对于队列的基本操作减低时间复杂度，因此我在这里使用两个栈进行了模拟**

**2.5.3数据结构设计**

**因为在这里使用了面向对象的编程思想，所以不太好单独列举，基本就是使用栈进行运算。**

**2.5.4功能说明（函数、类）**

1. class stack
2. {
3. public**:**
4. stack**();**
5. **~**stack**();**
6. **void** push**(**QElemtype x**);**
7. QElemtype pop**();**
8. QElemtype Max**();**
9. **void** set\_size**(int** number**);**
10. **bool** is\_empty**()** {**return** **(**top **==** 0**)?** true **:** false**;**}
11. **int** get\_size**()** {**return** top**;**}
12. private**:**
13. QElemtype**\*** item**;**
14. **int** top**;**
15. QElemtype**\*** second\_max\_pos**;**
16. **int** max\_item\_pos**;**
17. **int** size**;**
18. }**;**
20. stack**::**stack**()**
21. {
22. top **=** 0**;**
23. max\_item\_pos **=** **-**1**;**
24. item **=** **(**QElemtype**\*)(**malloc**(sizeof(int)** **\*** 5000**));**
25. second\_max\_pos **=** **(**QElemtype**\*)(**malloc**(sizeof(int)** **\*** 5000**));**
26. }
28. stack**::~**stack**()**
29. {
30. free**(**item**);**
31. free**(**second\_max\_pos**);**
32. }
33. **void** stack**::**push**(**QElemtype x**)**
34. {
35. **if** **(**top **==** 0**)**
36. {
37. item**[**top**]** **=** x**;**
38. max\_item\_pos **=** top**;**
39. second\_max\_pos**[**top**]** **=** **-**1**;***//记录的是前驱*
40. top**++;**
41. }
42. **else**
43. {
44. **if** **(**top **>=** size**)**
45. {
46. cout **<<** "Queue is Full" **<<** endl**;**
47. }
48. **else**
49. {
50. item**[**top**]** **=** x**;**
51. **if** **(**x **>** Max**())**
52. {
53. second\_max\_pos**[**top**]** **=** max\_item\_pos**;***//记录的是前驱*
54. max\_item\_pos **=** top**;**
55. }
56. **else**
57. second\_max\_pos**[**top**]** **=** **-**1**;**
58. top**++;**
59. }
60. }
61. }
62. QElemtype stack**::**pop**()**
63. {
64. QElemtype temp**;**
65. **if** **(**top **==** 0**)**
66. {
67. cout **<<** "Queue is Empty" **<<** endl**;**
68. **return** **-**1**;**
69. }
70. **else**
71. {
72. temp **=** item**[**top **-** 1**];**
73. **if** **(**top **-** 1 **==** max\_item\_pos**)**
74. {
75. max\_item\_pos **=** second\_max\_pos**[**top **-** 1**];**
76. }
77. top**--;**
78. **return** temp**;**
79. }
80. }
81. QElemtype stack**::**Max**()**
82. {
83. **if** **(**max\_item\_pos **>=** 0**)**
84. **return** item**[**max\_item\_pos**];**
85. **else**
86. **return** **-**100**;**
87. }
88. **void** stack**::**set\_size**(int** number**)**
89. {
90. size **=** number**;**
91. item **=** **(**QElemtype**\*)(**realloc**(**item**,** **sizeof(**size**)** **\*** number**));**
92. second\_max\_pos **=** **(**QElemtype**\*)(**realloc**(**second\_max\_pos**,** **sizeof(**size**)** **\*** number**));**
93. }
94. class queue
95. {
96. public**:**
97. queue**()** {}
98. **~**queue**()** {}
99. **void** en\_queue**(int** val**,** **int** number**);**
100. **int** de\_queue**();**
101. **void** set\_stack\_size**(int** number**);**
102. **int** max**();**
103. **int** get\_size**();**
104. **void** exit\_pop\_all**();**
105. private**:**
106. stack stack\_in**;**
107. stack stack\_out**;**
108. }**;**
109. **void** queue**::**en\_queue**(int** val**,** **int** number**)**
110. {
111. **if** **(**stack\_in**.**get\_size**()** **+** stack\_out**.**get\_size**()** **>=** number**)**
112. cout **<<** "Queue is Full" **<<** endl**;**
113. **else**
114. {
115. stack\_in**.**push**(**val**);**
116. }
117. }
119. **void** queue**::**set\_stack\_size**(int** number**)**
120. {
121. stack\_in**.**set\_size**(**number**);**
122. stack\_out**.**set\_size**(**number**);**
123. }
125. **int** queue**::**get\_size**()**
126. {
127. **return** stack\_in**.**get\_size**()** **+** stack\_out**.**get\_size**();**
128. }
130. **int** queue**::**de\_queue**()**
131. {
132. **if** **(!**stack\_in**.**is\_empty**()** **&&** stack\_out**.**is\_empty**())**
133. {
134. **while** **(!**stack\_in**.**is\_empty**())**
135. stack\_out**.**push**(**stack\_in**.**pop**());**
136. **return** stack\_out**.**pop**();**
137. }
138. **else** **if** **(**stack\_in**.**is\_empty**()** **&&** stack\_out**.**is\_empty**())**
139. **return** **-**1**;**
140. **else** **if** **(**stack\_in**.**is\_empty**()** **&&** **!**stack\_out**.**is\_empty**())**
141. **return** stack\_out**.**pop**();**
142. **else**
143. **return** stack\_out**.**pop**();**
144. }
146. **int** queue**::**max**()**
147. {
148. **if** **(!**stack\_in**.**is\_empty**()** **&&** stack\_out**.**is\_empty**())**
149. {
150. **while** **(!**stack\_in**.**is\_empty**())**
151. stack\_out**.**push**(**stack\_in**.**pop**());**
152. **return** stack\_out**.**Max**();**
153. }
154. **else** **if** **(**stack\_in**.**is\_empty**()** **&&** stack\_out**.**is\_empty**())**
155. **return** stack\_out**.**Max**();**
156. **else** **if** **(**stack\_in**.**is\_empty**()** **&&** **!**stack\_out**.**is\_empty**())**
157. **return** stack\_out**.**Max**();**
158. **else**
159. **return** **(**stack\_in**.**Max**()** **>** stack\_out**.**Max**())** **?** stack\_in**.**Max**()** **:** stack\_out**.**Max**();**
160. }
162. **void** queue**::**exit\_pop\_all**()**
163. {
164. **if** **(!**stack\_in**.**is\_empty**()** **&&** stack\_out**.**is\_empty**())**
165. {
166. **while** **(!**stack\_in**.**is\_empty**())**
167. stack\_out**.**push**(**stack\_in**.**pop**());**
168. **while** **(!**stack\_out**.**is\_empty**())**
169. cout **<<** stack\_out**.**pop**()** **<<** " "**;**
170. }
171. **else** **if** **(**stack\_in**.**is\_empty**()** **&&** stack\_out**.**is\_empty**())**
172. **return;**
173. **else** **if** **(**stack\_in**.**is\_empty**()** **&&** **!**stack\_out**.**is\_empty**())**
174. {
175. **while** **(!**stack\_out**.**is\_empty**())**
176. cout **<<** stack\_out**.**pop**()** **<<** " "**;**
177. }
178. **else**
179. {
180. **while** **(!**stack\_out**.**is\_empty**())**
181. cout **<<** stack\_out**.**pop**()** **<<** " "**;**
182. **while** **(!**stack\_in**.**is\_empty**())**
183. stack\_out**.**push**(**stack\_in**.**pop**());**
184. **while** **(!**stack\_out**.**is\_empty**())**
185. cout **<<** stack\_out**.**pop**()** **<<** " "**;**
186. }
187. }

**2.5.5 调试分析（遇到的问题和解决方法）**

**这道题总体来说，我调试了比较长的时间。因为使用两个栈来实现队列的基本操作需要比较复杂的功能实现，而我没有使用stl容器，所以很多细节容易遗漏。**

**2.5.6 总结和体会**

**我在这道题里体会了队列和栈的关系，同时对于队列的基本操作进行了优化，我认为是很有好处的。**

**3. 实验总结**