第四次作业选填讲解

MitakeMoca

选择题

1.A 选项中，栈和队可以线性存储，也可以链式存储（**链式存储**参考第三次作业里的链表代码，删几个功能就是栈和队了；**顺序存储**就是这一节里给大家展示的基于数组实现的代码）；B 选项中，栈和队都是线性结构（线性结构就是每个结点若有前驱元素、后继元素，则必然有唯一的前驱元素、后继元素）；D 选项也错在这。C、D 中的限制存取点，指的是栈限制栈顶存取；队限制队头取，队尾存。

2.递归过程里，处理参数和返回地址，要用到程序地址空间中的栈内存（关于栈内存这后面再写一个文档详细讲解）。

3.我们可以假设栈的容量是无限的，找到入栈出栈过程中栈中元素最多是多少个，即为栈的最小容量。代码的实现可以参考**选填板子 - 栈队 - 计算栈的最小容量**，下面我们简单讲解一下：

首先想做上这道题一定要知道栈和队最关键的性质，即栈是后进先出，队是先进先出。而且根据题意，一个元素出栈后立刻进入到队中，根据队的先进先出性质，所以题目中给出的出队顺序 $\leftrightarrow$ 入队顺序 $\leftrightarrow$ 出栈顺序。

再根据栈的后进先出性质，所以一个元素如果想出栈，那它此时必须在栈顶，即只有栈顶的元素才能出栈。反过来讲，原题中出栈序列中的第一个元素是 e_2，那在 e_2 入栈之后，必须马上出栈，因为此时如果还让其它元素继续入栈的话，显然 e_2 如果想出栈的话，它就要到达栈顶，那就要让此时它到栈顶之间的所有元素依次出栈，显然它就不是第一个出栈的了。

所以，整个入栈出栈过程如下（用数字代替元素）：

- 1 入栈，栈中有一个元素，栈中元素为 1，未处理的出栈序列为 $2\ 4\ 3\ 6\ 5\ 1$
- 2 入栈，栈中有两个元素，栈中元素为 $1\ 2$，未处理的出栈序列为 $2\ 4\ 3\ 6\ 5\ 1$
- 2 出栈，栈中有一个元素，栈中元素为 1，未处理的出栈序列为 $4\ 3\ 6\ 5\ 1$（因为 2 是未处理的出栈序列的第一个元素）
- 3 入栈，栈中有两个元素，栈中元素为 $1\ 3$，未处理的出栈序列为 $4\ 3\ 6\ 5\ 1$
- 4 入栈，栈中有三个元素，栈中元素为 $1\ 3\ 4$，未处理的出栈序列为 $4\ 3\ 6\ 5\ 1$
- 4 出栈，栈中有两个元素，栈中元素为 $1\ 3$，未处理的出栈序列为 $3\ 6\ 5\ 1$
- 3 出栈，栈中有一个元素，栈中元素为 1，未处理的出栈序列为 $6\ 5\ 1$
- 5 入栈，栈中有两个元素，栈中元素为 $1\ 5$，未处理的出栈序列为 $6\ 5\ 1$
- 6 入栈，栈中有三个元素，栈中元素为 $1\ 5\ 6$，未处理的出栈序列为 $6\ 5\ 1$
- 6 出栈，栈中有两个元素，栈中元素为 $1\ 5$，未处理的出栈序列为 $5\ 1$
- 5 出栈，栈中有一个元素，栈中元素为 1，未处理的出栈序列为 1
- 1 出栈，处理结束

在这个过程中，栈中元素个数最多为 3，故栈的容量至少为 3（要是小于 3 的话，这个过程就会发生栈溢出）。

4.利用和上一题类似的思路，将每个选项作为出栈序列，看看哪个选项最后能清空未处理的出栈序列，这不展开说了。代码实现可以参考 **选填板子 - 栈队 - 判断是否是栈的合法输出序列。**

5.跟上一题一模一样，代码实现可以参考 **选填板子 - 栈队 - 判断是否是栈的合法输出序列。**

6.中缀表达式转后缀表达式大家上课应该都讲了，这里我们简单的过一下流程。这道题的代码实现可以参考**选填板子 - 栈队 -中缀表达式转后缀表达式**，同样还为大家准备了中缀转前缀、前缀转中缀、后缀转中缀的板子。

- 读入一个操作数 A，直接输出；符号栈为空，输出序列为 A
- 读入一个符号 `-`，由于符号栈为空直接入符号栈；符号栈为 `-`，输出序列为 A
- 读入一个符号 `(`，直接入栈；符号栈为 `- (`，输出序列为 A
- 读入一个操作数 B，直接输出；符号栈为 `- (`，输出序列为 $\mathrm{A\ B}$
- 读入一个符号 `+`，符号栈顶是 `(`，所以直接入栈；符号栈为 `- ( +`，输出序列为 $\mathrm{A\ B}$
- 读入一个操作数 C，直接输出；符号栈为 `- ( +`，输出序列为 $\mathrm{A\ B\ C}$
- 读入一个符号 `/`，符号栈顶是 `+`，优先级比 `/` 低，所以直接入栈；符号栈为 `- ( + /`，输出序列为 $\mathrm{A\ B\ C}$
- 读入一个操作数 D，直接输出；符号栈为 `- ( + /`，输出序列为 $\mathrm{A\ B\ C\ D}$
- 读入一个符号 `)`，所以要一直让符号栈出栈，直到遇到 `(`；符号栈变为 `-`，输出序列为 $\mathrm{A\ B\ C\ D\ /\ +}$
- 读入一个符号 `*`，符号栈顶是 `-`，优先级比 `*` 低，所以直接入栈；符号栈为 `- *`，输出序列为 $\mathrm{A\ B\ C\ D\ /\ +}$
- 读入一个操作数 E，直接输出；符号栈为 `- *`，输出序列为 $\mathrm{A\ B\ C\ D\ /\ +\ E}$
- 输入结束，符号栈非空，将所有符号依次出栈，所以输出序列为 $\mathrm{A\ B\ C\ D\ /\ +\ E\ \times\ -}$

7.对于一个双向链表，在 q 前插入一个结点 p，要修改四个指针域：

- q 的前驱元素指针
- q 前驱元素的后继元素指针
- p 的前驱元素和后继元素指针

题目中三条语句依次修改的是 p 的后继元素指针；p 的前驱元素指针；q 的前驱元素指针。还差一个 q 前驱元素的后继元素指针，根据选项不难看出前驱指针用 `llink`，后继指针用 `rlink` 表示。所以选 B · · · 吗?

这里就是一个非常常见的 bug 了，你已经把 q 的前驱元素指针改了，让它变成 p 了，所以现在你用 `q -> llink` 找到的已经不是原来的那个 q 的前驱元素了，而是 p。原来的 q 的前驱元素现在已进成为了 p 的前驱元素了，所以正确答案其实是选 D。

8.A 在栈空时不能出栈；B 在栈满时不能入栈；D 在队满时不能入队。C 正确。

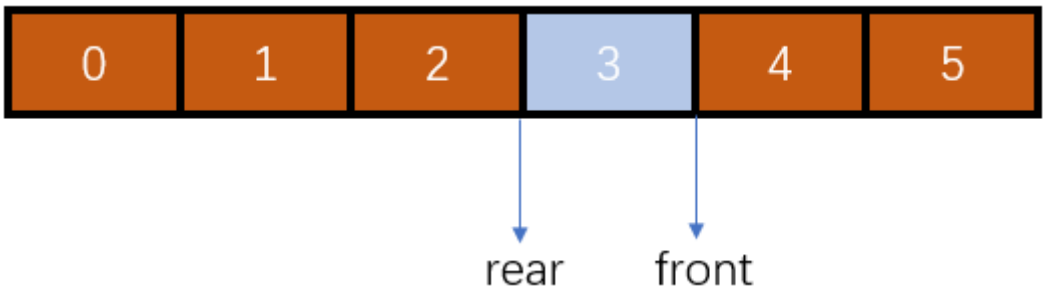
9.缓冲区应该满足先进先出，所以选择队。

10.首先，元素入队只改变 rear 的值，元素出队只改变 front 的值。以在一个元素入队后，由于队中只有一个元素，所以队头元素和队尾元素都是它，所以此时 front = rear = 0，而 front 始终没变，所以 front 一开始就为 0。rear $+1$ 再 $\mod n$ 后变成 0，不难知道 rear 的初始值为 n - 1。

11.普通的队列只支持队头取，队尾插。B 中最近入队的元素是队尾的元素，所以错误；C 队头处只支持取；所以选 D。

12.参考题目 3，代码的实现可以参考**选填板子 - 栈队 - 计算栈的最小容量。**

13.元素入队只改变 rear 的值，元素出队只改变 front 的值。据此，删除一个元素，front + 1 变成 4；插入两个元素，rear + 2 变成 2（整个过程中都没有超过 5，所以不用考虑取模的问题），所以选 B。最后存了元素的位置如下（一定要弄清这一点，在这个例子里从数组的视角上看 front 的位置是比 rear 靠后的）：



填空题

1.第一个空应该传参；第二个空用来找字符串的长度；第三个空用来根据 i、j 的值计算出返回值，如果 i、j 的值反映出字符串对称就返回 1，否则返回 0。

2.相当于选择题的第三题里，把每一步操作用一个字母表示，入栈就用 S，出栈就用 X，不多讲了。代码的实现可以参考**选填板子 - 栈队 - 给定入栈顺序出栈顺序求操作串。**

3.如果第 30 个元素是第一个出栈的，那显然在 $1 \sim 30$ 元素入栈的过程中没有元素出栈，最后栈顶到栈底就是 $30 \sim 1$，所以 p10 就应该是倒数第十个数字，也就是 21。

4.模拟一下这个流程即可：

- PUSH：a 入栈，栈中元素为 `a`，出栈序列为空
- PUSH：b 入栈，栈中元素为 `a b`，出栈序列为空
- POP：b 出栈，栈中元素为 `a`，出栈序列为 `b`
- PUSH：c 入栈，栈中元素为 `a c`，出栈序列为 `b`
- POP：c 出栈，栈中元素为 `a`，出栈序列为 `b,c`
- （其实这就可以结束了，因为后面没有 pop，不可能有元素出栈了）
- PUSH：d 入栈，栈中元素为 `a d`，出栈序列为 `b,c`
- PUSH：e 入栈，栈中元素为 `a d e`，出栈序列为 `b,c`

5.参考选择题第六题，代码实现可以参考**选填板子 - 栈队 -中缀表达式转后缀表达式。**

6.这题的核心在于理解栈是后进先出，队是先进先出；同时先放入栈的元素更靠近栈底，后放入栈的元素更靠近栈顶：

- 出栈顺序（入队顺序）：2 4 6 8 10
- 出队顺序（入栈顺序）：2 4 6 8 10

所以栈顶到栈底：10 8 6 4 2（2 最先入栈，10 最后入栈）

7.后缀表达式转中缀表达式大家上课应该都讲了，这里我们简单的过一下流程。这道题的代码实现可以参考**选填板子 - 栈队 - 后缀表达式转中缀表达式**：

- 读入一个操作数 `a`；数字栈为 `a`，输出序列为空
- （省略读操作数的过程）
- 读完五个操作数后，数字栈为 `a b c d e`
- 读入一个操作符 `+`；数字栈变为 `a b c d+e`
- 读入一个操作符 `*`；数字栈变为 `a b c*(d+e)`
- 读入一个操作符 `/`；数字栈变为 `a b/(c*(d+e))`（注意后出栈的作为被除数）
- 读入一个操作符 `*`；数字栈变为 `a*(b/c*(d+e))`