1. Stack Derived From Vector 04A

讲义中实现的Stack，是在底层引入一个Vector，并将其Front/Rear指定为Bottom/Top。

a) 试在示例代码的基础上略作修改，实现一个Front/Rear的约定颠倒的Stack；

b) 如此改动之后，Stack各接口的性能有何变化？为什么？

2. Stack Derived From List 04A

在示例代码中，其实也已基于List实现了Stack。

a) 试在示例代码包中找到对应的项目，确认其对List之Front/Rear的角色约定；

b) 如果颠倒二者的角色后再实现Stack，各接口的性能有何变化？为什么？

3. Theseus' Tools 04B1

如果将程序执行过程中的调用跟踪图比作Theseus探索的迷宫，那么调用栈就对应于他所使用的那个线团。然而为了避免“故地重游”甚至“原地转圈”，他还需要**更多工具**来标记已访问过的位置，比如一支粉笔。

在C/C++等编程语言中，是否也对应地有类似的机制？

4. Recursion Depth & Storage Cost 04B2

对于采用分治策略的算法，如果通过优化使得调用栈的每一帧都能节省**一半**空间，那么可求解问题的规模将会扩大到何等程度？如果能节省到**1/10**呢？（估计时可暂忽略其他方面的空间消耗。）

5. Recursion & Iteration 04B3

是否从理论上讲，任何程序中的递归都可消除掉？

如果不是，原因何在？如果总是可以，消除递归的通用方法是什么？

6. Binary/Linear Recursion 04B4

分治算法通常都对应于二分递归，每个递归实例都含有一前一后两次递归。如果后一次递归符合**尾递归**的特征——在每个递归实例中都是最后一步操作——那么是否也可套用讲义中的方法将其消除，从而先把算法**转化**为线性递归的模式？

7. Almost-Tail Recursion 04B4

讲义中所举的fac()算法，尽管从定义来看还不是**严格**的尾递归，但从某种以上来讲也可以归入此类——毕竟其对调用栈的使用同样是极其低效的，以致可以撤掉甩掉栈，将空间复杂度优化至$\mathcal{O}(1)$。

那么一般而言，这类**准的**尾递归从计算流程的角度看有何特点？

8. Reversed Output Sequence 04C

除了讲义中介绍的进制转换算法实例，你还知道在哪些算法中，答案的生成次序也是与输出次序**相反**，需要借助栈来**缓存**并**倒置**的？

9. Customized Parenthesis 04D

试拓展讲义中的括号匹配检测算法，使之能够：

a) 支持（如Html、Jason等格式中）可自定义的**任何**“括号”；

b) 在判断出失配的同时，还指出错误出现的**位置**，并给出**修正**的建议。

10. Parenthesis Expression ~ Stack Permutation 04D

a) 按照讲义及课上讲解的思路严格证明：合法的括号表达式与栈混洗之间存在一一**对应**关系。

b) 你是否还能从**其他**角度理解并证明这一对应关系？

11. Parenthesis Expression 04D

不难理解，由n对括号构成的所有合法表达式，可以按照字典序从1到Catalan(n)编号。试阅读D. E. Knuth的The Art Of Computer Programming第4卷第4册，了解以下问题及算法：

a) Algorithm P (I. Semba, 1981)：按字典序枚举由n对匹配括号组成的所有表达式；

b) Algorithm U (F. Ruskey, 1978)：按字典序取出其中的第k个；

c) Algorithm W (D. B. Arnold & M. R. Sleep, 1980)：等概率地从中随机选出一个。

12. Priority Table 04F2

考查本节所给的优先级表：

a) 基于这张查询表，evaluate()算法对于“2^3^4”是如何处理的？

b) 如果要与**数学课**上默认的优先级一致，这张表应该如何调整？算法同时还需如何调整？

c) 如此调整之后，时间、空间复杂度有何变化？

d) 该表中有若干项都是' '，它们分别在什么情况下会起作用？

13. Stack Size In evaluate() 04F2

a) 在表达式求值算法过程中的任何时刻，optr栈中的运算符是否必然是按优先级**单调**排列的？

b) 既然运算符不过**有限**的几种，optr栈（以及opnd栈）的最大规模是否存在一个$\mathcal{O}(1)$的上限？

c) 试对于任意的$n \gg 1$构造一个表达式，使得evaluate()算法中opnd栈的规模可以达到n。

14. RPN 04G

RPN的求值，要比对应的常规表达式更加简明、高效。然而我们也注意到，无论是手工还是自动地完成后者到前者的转换，本身也需要线性的时间。既如此，RPN的**价值**何在？

15. Polish Natation 04G

如果采用**PN**来做表达式求值，是否也能如**RPN**那样同样地简明、高效？为什么？

16. Queue Derived From Vector 04H

讲义中选择了基于**列表**来实现队列，是否也可改为基于**向量**来实现？

如果可以，具体如何实现？否则，为什么不可以？

17. More Queue Applications 04I

除了讲义中所举的实例，你还知道在哪些计算过程中使用了这种结构？

18. Maximum Rectangle In A Histogram 04J

a) 试举出Brute-force算法的**最坏**情况；其对应的运行时间是多少？

b) 不难理解，最大矩形可能同时存在**多个**。为此，讲义中约定着眼于其中的最靠左（leftmost）者。

试给出这个“最靠左”的具体定义？讲义中的算法所返回的，是否与之吻合？演示程序呢？

19. 2-Pass vs. 1-Pass 04J

本节针对直方图内最大矩形问题不仅介绍了蛮力算法，而且进而介绍了双趟扫描（2-Pass Scan）算法以及单趟扫描（1-Pass Scan）算法。

a) 单趟扫描算法所具有的特性中，有哪些是双趟扫描算法并不具备的?

b) 试分别运用04L节中的三种典型方法，分析双趟扫描、单趟扫描算法的分摊复杂度；

c) 就**最坏**情况而言，三个算法的空间复杂度各是多少?

d) 就**最好**情况而言，三个算法的空间复杂度各是多少?

e) 就**平均**情况而言，三个算法的空间复杂度各是多少?

20. Steap & Queap 04K

试分别采用04L节中介绍的方法，**分析**本节所介绍Queap结构的分摊复杂度。

21. Stacks As Queue 04L

讲义中针对“双栈当队”策略的分摊复杂度，介绍了三种典型的分析方法。然而从**字面**上看，它们所得复杂度的**常系数**还是有所区别。这种区别仅仅是常系数的放缩，还是**本质**性的?

22. Short-circuit Boolean Evaluation 01~04

我们的示例代码经常会利用到C\C++等语言支持的**短路求值**特性，试找出三处这样的实例。