Задача 3. Стеки/очереди

Требуется решить задачу в 2-х вариантах:

1. С использованием стека / очереди, самостоятельно реализованных на основе связного списка.
2. С использованием реализации стека / очереди, которая уже есть в стандартной библиотеки языка Java.

Решение задачи предполагает наличие оконного интерфейса для демонстрации работы программы. Должен быть предусмотрен ввод исходных данных из файла.

Если в условии задачи что-то непонятно – попросить пояснить преподавателя.

Варианты:

1. Аналог пузырьковой сортировки для очереди (вспомогательные списки / массивы / очереди не применять).
   Подсказка: Внутренний цикл пузырьковой сортировки можно представить следующим алгоритмом: извлекается элемент из очереди и запоминается в переменной tmp. Далее (выполняется n-1 раз, n – кол-во элементов в очереди) извлекается элемент из очереди, сравнивается с tmp: меньшее значение записывается в очередь, большее запоминается в tmp. В результате в tmp остается наибольший элемент, который записывается в очередь (в конец). Аналогично ищется 2-ой по величине элемент (кол-во итераций n-2), записывается в конец очереди, а затем из начала очереди в конец переписывается наибольший элемент. И т.д. Сложность алгоритма будет $O(n^2)$.

2. Дана очередь целых чисел. Обработать очередь таким образом, чтобы в очереди остались только четные элементы.

3. Даны две очереди целых чисел. Не используя других структур данных, добавить в конец первой очереди элементы второй очереди, а в конец второй очереди элементы первой очереди.

4. Дана очередь из строк. Обработать очередь таким образом, чтобы каждый элемент очереди был продублирован (например: { "stack", "list", "queue" } –> { "stack", "stack", "list", "list", "queue", "queue" }).

5. Для целого числа X получить целое число Y, цифры в котором расположены в обратном порядке (относительно числа X). Строки не использовать.
   Подсказка: извлекать цифры (в виде чисел) от последней к первой с помощью операции остатка целочисленного деления на 10, 100, 1000 и т.д., помещать полученные цифры в стек, затем формировать Y, умножая извлеченные из стека числа на 10, 100, 1000 и т.д. Реализовать ту же самую задачу без использования структуры стек рекурсивно.

6. Даны две очереди X и Y, содержащие вещественные числа. Из каждой очереди одновременно извлекается по одному числу, x и y соответственно. Если x < y, то число (x + y) помещается в конец очереди X, иначе число (x–y) помещается в конец очереди Y. Необходимо определить число шагов, через которое одна из очередей станет пустой.

7. (*) Имеется стопка из N карточек с написанными на них числами. Карточки раскладываются следующим образом: первая карточка кладется на стол, вторая перекладывается под низ стопки карточек, эти операции повторяются пока все карточки не окажутся разложенными на столе. Известна последовательность чисел, которая

получилась в результате данного алгоритма. Требуется найти исходную последовательность карточек (чисел) в стопке.
Подсказка: необходимо промоделировать обратную последовательность действий приведенного алгоритма.

8. Написать программу для вычисления значения выражения, представленного в обратной польской записи.

Обычная запись	Обратная польская запись
(b + c) * d	b c + d *
a + (b + c) * d	a b c + d * +
(6 + 8)/2 + 11	6 8 + 2 / 11 +

Подсказка: просматривая строку, в которой записано выражение, анализируем очередной символ. Если это число, то записываем его в стек. Если это знак операции, то достаём два элемента из стека, выполняем арифметическую операцию, определяемую этим знаком, и заносим результат в стек.

9. Дана очередь, поддерживающая только две операции – добавление элемента и извлечение элемента (соответственно нет метода, возвращающего кол-во элементов в очереди). В очереди записаны целые числа. Не используя другие структуры данных найти кол-во элементов в очереди, а также значения максимального и минимального элементов. После всех манипуляций очередь должна оказаться в первоначальном состоянии.
Подсказка: первой операцией поместить в очередь значение null (считаем, что очередь обобщенная и числа записаны как Integer-объекты), которое будет индикатором, что вся очередь уже обработана.

10. (*) Найти минимальное кол-во шагов, за которое шахматная фигура конь может из одной заданной клетки шахматного поля попасть в другую клетку шахматного поля, а также последовательность таких шагов (если их несколько, то любую).
Подсказка: Использовать очередь, в которую поместить начальное положение коня. Далее, пока не попадем в требуемую клетку, извлекаем из очереди клетку, для которой перебираем все возможные ходы конем (клетки), которые помещаем в очередь. Для хранения последовательности ходов используем стек или просто массив/список.

11. (*) Поиск кратчайшего пути от одной клетки до другой клетки в лабиринте на плоскости с использованием очереди и без применения рекурсии. Должна быть возможность задать лабиринт на форме (с помощью мышки), а также прочитать / записать в текстовый файл.
Подсказка: использовать алгоритм поиска в ширину (англ. breadth-first search, BFS): https://ru.wikipedia.org/wiki/Поиск_в_ширину

12. В стеке записаны строки. Переставить элементы стека в обратном порядке. Допускается использование в качестве вспомогательных структур данных только стеков.

13. Реализовать очередь с использованием 2-х стеков таким образом, чтобы операции добавления и извлечения из очереди в среднем (обратите внимание: в среднем) работали за время O(1) (в предположении, что основные методы используемых реализаций стека также работают за время O(1)).
Подсказка: http://e-maxx.ru/algo/stacks_for_minima (Модификация очереди. Способ 2)

14. В очереди записаны целые числа. Не используя других структур данных поменять в очереди элементы: 1-ый со 2-ым, 3-ий с 4-ым и т.д. Если кол-во элементов в очереди нечетное, то последний элемент после всех преобразований должен остаться на своем месте.

15. В очереди целых чисел переставить элементы очереди таким образом, чтобы все отрицательные элементы оказались в начале списка, все положительные – в конце (нулевые – между положительными и отрицательными). При этом взаимный порядок следования отрицательных элементов, также как и положительных, поменяться не должен. В качестве вспомогательных структур данных можно использовать только очереди.

16. В очереди записаны строки. Не используя другие структуры данных, переставить элементы очереди в обратном порядке.
    Подсказка: рекурсивная реализация (будет неявно использоваться стек вызова методов).

17. Создать очередь длины n, информационные элементы которой будут очередями целых чисел вида: { {1}, {1, 2}, {1, 2, 3}, … , {1, 2, …, N} }.

18. Смоделировать игру ход карточной игры «Пьяница»:
    https://ru.wikipedia.org/wiki/Пьяница_(карточная_игра). 36 перемешиваются и раздаются 2-м игрокам поровну. Карты каждого игрока представлены в виде очереди. Каждый ход оба игрока берут по карте из очереди сравнивают и тот, у кого карта старше, добавляет обе карты в конец очереди (остальные правила, как минимум «Спор», надо также смоделировать).
    «Рисовать» анимацию не требуется, но в каком-то виде требуется показывать карты одного и другого игрока (как вариант, в строке через запятую) и по кнопке осуществлять ход.

19. Смоделировать следующую игру. Берется колода карт, перемешивается. Верхняя карта кладется на стол. Далее берется следующая карта, если она той же масти или достоинства, то кладется на карту на столе, в противном случае перекладывается вниз стопки карт. Действия продолжаются, пока в стопке карт остаются карты, которые можно положить на самую верхнюю карту на столе по описанным правилам. Необходимо найти, а) сколько будет сделано ходов, б) какие карты и в каком порядке окажутся на столе, в) какие карты и в каком порядке останутся в стопке карт.
    «Рисовать» анимацию не требуется, только в каком-либо виде показать, какие карты и в каком порядке были вначале, а также результат «игры» (пункты а, б, в).

20. (*) Смоделировать работу магазина. Отсчет времени начинается с 0, все время дискретно (т.е. события случаются только в «целочисленные» моменты времени, длительность любого действия также «целочисленна»). Покупатели приходят в магазин, набирают товары и идут на кассу. Для каждого покупателя известно, когда он пришел в магазин (S), сколько по времени выбирал товары (T), сколько товаров выбрал (N, будем считать, что время обслуживания на кассе пропорционально кол-ву набранных товаров и также составляет N). Когда покупатель подходит к кассе, если касса обслуживает другого покупателя, то подошедший становится в очередь. Если к кассе одновременно подходят несколько покупателей, то они могут встать в очередь в случайном порядке.
    Для указанных входных данных для каждого покупателя найти момент времени, когда он покинет (может покинуть) магазин.
    «Рисовать» анимацию не требуется, достаточно, чтобы можно было ввести входные данные (или загрузить из файла) и просмотреть результат моделирования.

21. (*) Имеется сетевой принтер. На принтер поступают задания на печать. Отсчет времени начинается с 0, задания могут поступать только в «целочисленные» момент времени (в момент поступления задания информация о задании уже доступна принтеру). Каждое задание состоит из какого-то кол-ва страниц – N, имеет целочисленный приоритет – P и некоторый случайный идентификатор задания – X. Одну страницу принтер печатает за одну единицу времени. Если принтер начал печатать какое-то задание, то он последовательно печатает все страницы этого задания. При выборе очередного задания принтер выбирает задание с наивысшим приоритетом в очереди печати; если заданий с наивысшим приоритетом несколько, то первое по времени поступления; если же два предыдущих параметра совпадают, то выбирается задание с меньшим идентификатором. Вам известна вся информация о поступавших на принтер заданиях для печати. Необходимо для каждого задания определить интервал времени, в течение которого оно выполнялось. Для моделирования последовательности выполнения заданий использовать очередь с приоритетом. Очередь с приоритетом при добавлении объекта помещает его в позицию, в соответствии с его приоритетом (необходимо реализовать операцию сравнения для объектов, чтобы определять, кто из них приоритетнее в конкретной задаче).
    «Рисовать» анимацию не требуется, достаточно, чтобы можно было ввести входные данные (или загрузить из файла) и просмотреть результат моделирования.

22. (*) Дана очередь символов. Написать рекурсивную процедуру проверки, образуют ли символы очереди палиндром (например: { 'a', 'b', 'b', 'c', 'b', 'b', 'a' }. Вспомогательных структур данных не использовать (в рекурсивной реализации неявно будет использован стек вызова методов). После всех манипуляций очередь должна содержать те же самые элементы и в том же порядке, что и в начале.
    Подсказка: необходимо написать рекурсивную функцию, которая будет выбирать и запоминать очередной элемент из очереди, помещать выбранное значение образно в очередь, затем рекурсивно вызывать саму себя, а после возврата выбирать еще один элемент очереди, сравнивать его с запомненным элементом и также помещать обратно в очередь. Подумать, как правильно ограничить глубину рекурсии и корректно обработать случаи четного и нечетного кол-ва элементов в очереди.

23. Найти все файлы с заданным расширением в папке и ее подпапках. Для чтения набора файлов и подпапок в папке использовать класс java.io.File (метод listFiles()). Реализация с использованием очереди. Вначале в очередь помещается папка, в которой ищем файлы. Затем, в цикле, пока в очереди есть элементы, выбираем из очереди элемент (папку). Для данной папки находим все вложенные элементы: если это нужный файл, помещаем его в итоговый список, если это папка, то добавляем ее в очередь. Заменить в реализации очередь на стек, объяснить, каким образом и почему изменится порядок найденных файлов.

24. С использованием стека «перевернуть» односвязный список строк (первый элемент должен стать последним, 2-ой предпоследним и т.д.). Необходимо реализовать дополнительный метод в списке, который используя стек выполняет данную задачу, модифицируя ссылки next в элементах списка (также необходимо поменять head и tail в самом списке).
    Реализовать ту же самую задачу без использования структуры стек рекурсивно.

25. Поместить в очередь с приоритетом слова из текстового файла. Приоритет определяется длиной слова (длинные слова более приоритетны).

26. Реализовать функцию проверки правильности скобочной структуры. Скобки могут быть

круглыми, квадратными и фигурными. В реализации использовать стек.
Примеры правильных скобочных выражений: (), ([])\{\}, []\{\}, ((())\{\}), ([\{\}]), неправильных –)(, ())((), (,))]), ((\{\}), [\{]\} и т.д.
Реализовать ту же самую задачу без использования структуры стек рекурсивно.

27. (*) Найти взаимное расположение 8-ми ферзей на шахматной доске такое, чтобы ферзи не «били» друг друга. Рекурсию не использовать, но можно использовать стек.

28.