Формулировки задач по

Конструированию алгоритмов и структур данных

!!

Каждая задача должна быть загружена на личный git-репозиторий отдельным коммитом, возможно, не одним. Все коммиты должны иметь осмысленные названия и описания того, что в них выполнено. Защита работы возможна на любом лабораторном занятии. Наличие выполненных работ учитывается при выставлении зачёта, а также влияет на итоговую оценку на экзамене.

Если две и более задачи выполнены в один коммит, работа не проверяется. Если все коммиты сделаны в один час, работа не проверяется.

Задачи должны выполняться последовательно, и каждая задача должна быть сдана до начала работы над следующей.

!!

Задача 1 (длина вектора).

Описать метод, находящий длину x вектора в N-мерном пространстве. Пространство задаётся матрицей метрического тензора G (если интересно, то можно почитать, что это, если нет, то нет). Матрица G должна быть симметричной, её размерность совпадает с размерностью пространства. Нахождение длины: $\sqrt{x \times G \times x^T}$. Размерность пространства, матрица тензора и вектор вводятся из файла. Память выделяется динамически; проверка на то, что матрица симметрична – необходима. Результат выводится на экран.

Задача 2 (комплексные числа).

Определить структуру для хранения комплексных чисел. Методы: создание комплексного числа по вещественной и мнимой части, сложение, вычитание, умножение, деление, нахождение модуля и аргумента, возврат мнимой и вещественной частей, вывод. При необходимости передавать структуру в качестве параметра используйте ссылки.

Для демонстрации работы методов организуйте текстовое меню с помощью бесконечного цикла. Выбор действий организовать вводом символа, который обрабатывается оператором switch. Кроме методов необходимо иметь возможность выхода из программы по пункту меню. Выход может быть осуществлён по символам 'Q' и 'q'. При вводе неизвестного символа, должно быть выведено сообщение «неизвестная команда». (Операция ввода получает одно число, с которыми должны работать остальные операции. Если операции необходимо два аргумента – введите второй аргумент. Результат должен записываться в исходное число. До первого ввода считать число равным 0).

Задача 3 (сортировки массивов целых чисел).

Реализовать приложение с графическим интерфейсом для сравнения эффективности различных алгоритмов сортировки на разных размерах массивов целых чисел.

Определить, какие алгоритмы сортировки работают эффективнее всего на различных размерах массивов целых чисел.

Реализовать следующие алгоритмы сортировки:

1) сортировка пузырьком (Bubble sort);
2) шейкерная сортировка (Shaker sort);
3) сортировка расчёской (Comb sort);
4) сортировка вставками (Insertion sort);
5) сортировка Шелла (Shellsort);
6) сортировка деревом (Tree sort);
7) гномья сортировка (Gnome sort);
8) сортировка выбором (Selection sort);
9) пирамидальная сортировка (Heapsort);
10) быстрая сортировка (Quicksort);
11) сортировка слиянием (Merge sort);
12) поразрядная сортировка (Radix sort);
13) битонная сортировка (Bitonic sort).

Подготовить четыре группы тестовых данных:

1) Массив случайных чисел по модулю 1000.
2) Массивы, разбитые на несколько отсортированных подмассивов разного размера. Размеры подмассивов определяются случайным числом по модулю некоторой константы (10, 100, 1000 и т.д. вплоть до размера массива).
3) Изначально отсортированные массивы случайных чисел с некоторым числом перестановок двух случайных элементов.

4) Полностью отсортированные массивы (в прямом и обратном порядке), массивы с несколькими заменёнными элементами, массивы с большим количеством повторений одного элемента (10%, 25%, 50%, 75% и 90%).

Разделить алгоритмы сортировки на три группы:

1) пузырьком, вставками, выбором, шейкерная, гномья;
2) битонная, Шелла, деревом;
3) расчёской, пирамидальная, быстрая, слиянием, поразрядная.

Провести тестирование:

1) сортировки первой группы тестировать на массивах размером 10, 100, …, 10^4 элементов;
2) сортировки второй группы тестировать на массивах размером 10, 100, …, 10^5 элементов;
3) сортировки третьей группы тестировать на массивах размером 10, 100, …, 10^6 элементов.

Для каждого теста провести 20 запусков и вычислить среднее время работы алгоритма.

Представить результаты сравнения в виде графиков, показывающих, какие алгоритмы сортировки работают эффективнее всего на каждом типе тестовых данных.

Проанализировать полученные результаты и сделать выводы об эффективности различных алгоритмов сортировки на разных размерах массивов целых чисел.

Требования к интерфейсу:

1) Главное окно приложения должно содержать следующие элементы управления:
    a. Выпадающий список для выбора группы тестовых данных (случайные числа, разбитые на подмассивы, отсортированные массивы и т.д.).
    b. Выпадающий список для выбора группы алгоритмов сортировки (первая группа, вторая группа, третья группа).
    c. Кнопку «Сгенерировать массивы» для создания тестовых массивов в соответствии с выбранной группой.
    d. Кнопку «Запустить тесты» для запуска сортировки выбранными алгоритмами и отображения результатов.
    e. Кнопку «Сохранить результаты» для выгрузки сгенерированных и отсортированных массивов в файл.

f. Область для отображения графиков, показывающих эффективность алгоритмов сортировки на разных размерах массивов.

2) После нажатия кнопки «Сгенерировать массивы» должны быть созданы тестовые массивы в соответствии с выбранной группой и размером. Массивы должны быть сохранены в памяти приложения.
3) После нажатия кнопки «Запустить тесты» должны быть выполнены следующие действия:
   a. Для каждого алгоритма сортировки из выбранной группы должно быть проведено 20 запусков на каждом размере массива.
   b. Для каждого теста должно быть вычислено среднее время работы алгоритма.
   c. Результаты тестирования должны быть отображены в виде графиков с использованием библиотеки ZedGraph.
4) После нажатия кнопки «Сохранить результаты» должны быть сохранены в файл следующие данные:
   a. Сгенерированные тестовые массивы.
   b. Отсортированные массивы для каждого алгоритма сортировки.
5) Для построения графиков следует использовать библиотеку ZedGraph.
6) Приложение должно быть реализовано с использованием Windows Forms.

Подсказки:

1) Создание массива заданного размера, заполненного случайными целыми числами в диапазоне от 0 до 1000:

```
int size = 10; // Размер массива
int module = 1000; // Модуль (верхняя граница случайных чисел)
int[] array = new int[size];
Random rand = new Random();

// Заполняем массив случайными числами
for (int i = 0; i < size; i++)
    array[i] = rand.Next(0, module);

// Выводим массив на экран
Console.WriteLine("Сгенерированный массив:");
for (int i = 0; i < size; i++)
    Console.Write(array[i] + " ");

Console.WriteLine();
```

2) Для измерения времени выполнения какой-либо операции можно использовать класс Stopwatch из пространства имён System.Diagnostics:

```
using System.Diagnostics;

// Создаём экземпляр класса Stopwatch
Stopwatch stopwatch = new Stopwatch();
```

```
// Засекаем время начала операции
stopwatch.Start();

// Выполняем какую-либо операцию
for (int i = 1; i <= 10000; ++i)
    Console.WriteLine(i);

// Останавливаем счётчик
stopwatch.Stop();

// Выводим время выполнения в миллисекундах
Console.WriteLine($"Время выполнения: {stopwatch.ElapsedMilliseconds} мс.");
```

Задача 4 (сортировки массивов различных типов).

На основе реализованных в задаче 3 алгоритмов сортировки, создайте обобщённые методы сортировки, которые могут работать с различными типами данных. Например, вместо сортировки массивов целых чисел, реализуйте методы, которые могут сортировать массивы любых объектов, используя компаратор для сравнения элементов.

1) Реализуйте обобщённые версии алгоритмов сортировки, рассмотренные в задаче 3. Каждый метод должен принимать массив объектов и компаратор для сравнения элементов.
2) Протестируйте эффективность обобщённых методов сортировки на различных типах данных и размерах массивов, аналогично тому, как это было сделано в задаче 3.
3) Сравните результаты с эффективностью реализаций алгоритмов сортировки из задачи 3. Проанализируйте, как использование обобщённых методов влияет на эффективность.
4) Представьте результаты сравнения в виде графиков, показывающих, какие обобщённые алгоритмы сортировки работают эффективнее всего на различных типах данных и размерах массивов.
5) Сделайте выводы об эффективности использования обобщённых методов сортировки по сравнению с реализациями, ориентированными на конкретные типы данных.

Задача 5 (куча).

Реализовать кучу в виде обобщённого класса.

Необходимо реализовать следующую функциональность:

1) Конструктор, принимающий на вход массив элементов и создающий из них кучу.

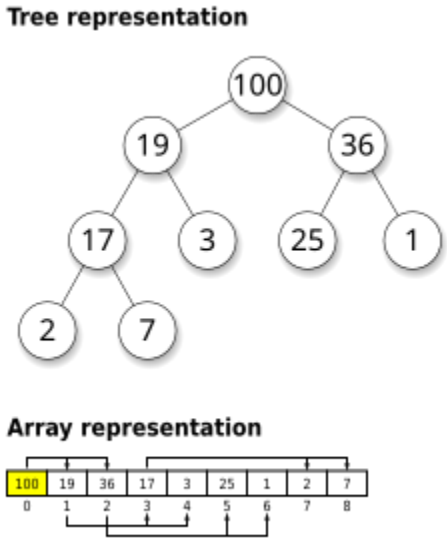
2) Метод для нахождения максимума (или минимума для min-кучи), возвращающий максимальный (или минимальный) элемент кучи без его удаления.
3) Метод для удаления максимума (или удаления минимума для min-кучи), удаляющий и возвращающий максимальный (или минимальный) элемент кучи.
4) Метод для увеличения ключа (или уменьшения ключа для min-кучи), принимающий на вход индекс элемента и новое значение ключа, обновляющий ключ элемента и восстанавливающий свойство кучи.
5) Метод для добавления, принимающий на вход новый элемент и добавляющий его в кучу с сохранением свойства кучи.
6) Метод для слияния, принимающий на вход другую кучу и объединяющий её с текущей кучей, создавая новую кучу, содержащую все элементы обеих исходных куч.

Куча должна обладать следующими преимуществами:

1) Быстрые нахождение максимума (или минимума) и добавление. Скорость таких операций – $O(\log n)$.

Предусмотреть обработку возможных ошибок.

Куча (англ. *heap*) – специализированная структура данных типа дерева, которая удовлетворяет *свойству кучи*: если B является узлом-потомком узла A, то $k(A) \geq k(B)$, где $k(X)$ – ключ узла. Из этого следует, что элемент с наибольшим значением ключа всегда является корневым узлом кучи, поэтому иногда такие кучи называют *max-кучами* (в качестве альтернативы, если сравнение перевернуть, то наименьший элемент будет всегда корневым узлом, такие кучи называют *min-кучами*).



Задача 6 (очередь с приоритетами).

Определить обобщённый класс MyPriorityQueue. Класс представляет собой реализацию очереди с приоритетами, которая может расти по мере необходимости, использующую кучу.

Необходимы поля:

1) queue – массив обобщённого (универсального) типа T для хранения элементов;
2) size – количество элементов в очереди с приоритетами;
3) comparator – компаратор для сравнения элементов в очереди с приоритетами.

Необходимо реализовать следующую функциональность:

1) Конструктор MyPriorityQueue() для создания пустой очереди с приоритетами с начальной ёмкостью 11, размещающей элементы согласно естественному порядку сортировки.
2) Конструктор MyPriorityQueue(T[] a) для создания очереди с приоритетами, содержащей элементы передаваемого массива a;
3) Конструктор MyPriorityQueue(int initialCapacity) для создания пустой очереди с приоритетами с указанной начальной ёмкостью;
4) Конструктор MyPriorityQueue(int initialCapacity, PriorityQueueComparer comparator) для создания пустой очереди с приоритетами с указанной начальной ёмкостью и компаратором;
5) Конструктор MyPriorityQueue(MyPriorityQueue<T> c) для создания очереди с приоритетами, содержащей элементы указанной очереди с приоритетами.

6) Метод add(T e) для добавления элемента в конец очереди с приоритетами. Если текущая ёмкость меньше 64, то новая ёмкость увеличивается на 2. В противном случае новая ёмкость увеличивается на 50% от текущей ёмкости.
7) Метод addAll(T[] a) для добавления элементов из массива.
8) Метод clear() для удаления всех элементов из очереди с приоритетами.
9) Метод contains(object o) для проверки, находится ли указанный объект в очереди с приоритетами.
10) Метод containsAll(T[] a) для проверки, содержатся ли указанные объекты в очереди с приоритетами.
11) Метод isEmpty() для проверки, является ли очередь с приоритетами пустой.
12) Метод remove(object o) для удаления указанного объекта из очереди с приоритетами, если он есть там.

13) Метод removeAll(T[] a) для удаления указанных объектов из очереди с приоритетами.
14) Метод retainAll(T[] a) для оставления в очереди с приоритетами только указанных объектов.
15) Метод size() для получения размера очереди с приоритетами в элементах.
16) Метод toArray() для возвращения массива объектов, содержащего все элементы очереди с приоритетами.
17) Метод toArray(T[] a) для возвращения массива объектов, содержащего все элементы очереди с приоритетами. Если аргумент a равен null, то создаётся новый массив, в который копируются элементы.

18) Метод element() для возвращения элемента из головы очереди с приоритетами без его удаления.
19) Метод offer(T obj) для попытки добавления элемента obj в очередь с приоритетами. Возвращает true, если obj добавлен, и false в противном случае.
20) Метод peek() для возврата элемента из головы очереди с приоритетами без его удаления. Возвращает null, если очередь пуста.
21) Метод poll() для удаления и возврата элемента из головы очереди с приоритетами. Возвращает null, если очередь пуста.

Предусмотреть обработку возможных ошибок.

Задача 7 (заявки в приоритетной очереди).

С клавиатуры вводится количество шагов добавления заявок в приоритетную очередь N. На каждом шаге производятся следующие действия: генерируются и добавляются в очередь от 1 до 10 заявок (конкретное число заявок выбирается случайно). Каждая заявка содержит следующую информацию: приоритет (случайное целое число от 1 до 5), номер заявки (заявки считаются начиная с 1, нумерация сквозная на всех шагах), номер шага на котором заявка поступила в систему. После добавления заявок, заявка с наибольшим приоритетом удаляется.

После завершения N шагов генерации заявок, шаги продолжаются без генерации (только удаление) до тех пор, пока очередь не станет пуста.

Необходимо подсчитать максимальное время ожидания заявки в системе и вывести всю информацию о заявке, которая ожидала максимальное время.

Кроме того, следует сохранять информацию о каждом добавлении заявки в очередь и удалении заявки в очередь в файл log.txt. Запись в файле имеет следующую структуру:

ADD/REMOVE НомерЗаявки Приоритет НомерШага

Для хранения и обработки данных использовать собственную реализацию очереди с приоритетами MyPriorityQueue. Использование встроенной очереди с приоритетами приведёт к незачёту задачи.

Задача 8 (динамический массив).

Определить обобщённый класс MyArrayList. Класс представляет собой реализацию динамического массива, который может расти по мере необходимости.

Необходимы поля:

1) elementData – массив обобщённого (универсального) типа T для хранения элементов динамического массива;
2) size – количество элементов в динамическом массиве.

Необходимо реализовать следующую функциональность:

1) Конструктор MyArrayList() для создания пустого динамического массива.
2) Конструктор MyArrayList(T[] a) для создания динамического массива и заполнения его элементами из передаваемого массива a.
3) Конструктор MyArrayList(int capacity) для создания пустого динамического массива с внутренним массивом, размер которого будет равен значению параметра capacity.
4) Метод add(T e) для добавления элемента в конец динамического массива. Если размер динамического массива больше размера внутреннего массива, необходимо создать новый массив размером в 1,5 раза больше исходного, плюс один элемент, скопировать все элементы из старого массива в новый и сохранить новый массив во внутренней переменной объекта MyArrayList.
5) Метод addAll(T[] a) для добавления элементов из массива.
6) Метод clear() для удаления всех элементов из динамического массива.
7) Метод contains(object o) для проверки, находится ли указанный объект в динамическом массиве.
8) Метод containsAll(T[] a) для проверки, содержатся ли указанные объекты в динамическом массиве.

9) Метод isEmpty() для проверки, является ли динамический массив пустым.
10) Метод remove(object o) для удаления указанного объекта из динамического массива, если он есть там.
11) Метод removeAll(T[] a) для удаления указанных объектов из динамического массива.
12) Метод retainAll(T[] a) для оставления в динамическом массиве только указанных объектов.
13) Метод size() для получения размера динамического массива в элементах.
14) Метод toArray() для возвращения массива объектов, содержащего все элементы динамического массива.
15) Метод toArray(T[] a) для возвращения массива объектов, содержащего все элементы динамического массива. Если аргумент a равен null, то создаётся новый массив, в который копируются элементы.
16) Метод add(int index, T e) для добавления элемента в указанную позицию.
17) Метод addAll(int index, T[] a) для добавления элементов в указанную позицию.
18) Метод get(int index) для возвращения элемента в указанной позиции.
19) Метод indexOf(object o) для возвращения индекса указанного объекта, или -1, если его нет в динамическом массиве.
20) Метод lastIndexOf(object o) для нахождения последнего вхождения указанного объекта, или -1, если его нет в динамическом массиве.
21) Метод remove(int index) для удаления и возвращения элемента в указанной позиции.
22) Метод set(int index, T e) для замены элемента в указанной позиции новым элементом.
23) Метод subList(int fromIndex, int toIndex) для возвращения части динамического массива, т.е. элементов в диапазоне [fromIndex; toIndex).

Динамический массив должен обладать следующими преимуществами:

1) Быстрый доступ по индексу. Скорость такой операции – O(1).
2) Быстрые вставка и удаление элементов с конца. Скорость операций опять же – O(1).

Предусмотреть обработку возможных ошибок.

Задача 9 (считывание из файла в динамический массив).

Из файла input.txt считывать строки (без пробелов). Из каждой строки извлечь теги (Пример: <html>, </H1>, <PrIvet>) и поместить их в динамический массив. Тег начинается символом <, заканчивается символом >, после < может иметь символ /. Остальные символы — цифры и буквы, первый символ – обязательно буква. Удалить из списка повторяющие теги (с точностью до наличия/отсутствия символа /, и регистра букв, т. е. <html> и </HtMl> – одинаковые теги).

Для хранения и обработки данных использовать собственную реализацию динамического массива MyArrayList. Использование встроенного динамического массива приведёт к незачёту задачи.

Задача 10 (вектор).

Определить обобщённый класс MyVector. Класс представляет собой реализацию вектора, который может расти по мере необходимости.

Необходимы поля:

1) elementData – массив обобщённого (универсального) типа T для хранения элементов вектора;
2) elementCount – количество элементов в векторе;
3) capacityIncrement – значение, на которое увеличивается ёмкость вектора при необходимости.

Необходимо реализовать следующую функциональность:

1) Конструктор MyVector(int initialCapacity, int capacityIncrement) для создания пустого вектора с начальной ёмкостью initialCapacity и значением приращения ёмкости capacityIncrement.
2) Конструктор MyVector(int initialCapacity) для создания пустого вектора с начальной ёмкостью initialCapacity и значением приращения ёмкости по умолчанию (0).
3) Конструктор MyVector() для создания пустого вектора с начальной ёмкостью по умолчанию (10) и значением приращения ёмкости по умолчанию (0).
4) Конструктор MyVector(T[] a) для создания вектора и заполнения его элементами из передаваемого массива a.

5) Метод add(T e) для добавления элемента в конец вектора. Если размер вектора больше текущей ёмкости, необходимо увеличить ёмкость вектора на значение capacityIncrement (если оно не равно 0) или удвоить текущую ёмкость.
6) Метод addAll(T[] a) для добавления элементов из массива.
7) Метод clear() для удаления всех элементов из вектора.
8) Метод contains(object o) для проверки, находится ли указанный объект в векторе.
9) Метод containsAll(T[] a) для проверки, содержатся ли указанные объекты в векторе.
10) Метод isEmpty() для проверки, является ли вектор пустым.
11) Метод remove(object o) для удаления указанного объекта из вектора, если он есть там.
12) Метод removeAll(T[] a) для удаления указанных объектов из вектора.
13) Метод retainAll(T[] a) для оставления в векторе только указанных объектов.
14) Метод size() для получения размера вектора в элементах.
15) Метод toArray() для возвращения массива объектов, содержащего все элементы вектора.
16) Метод toArray(T[] a) для возвращения массива объектов, содержащего все элементы вектора. Если аргумент a равен null, то создаётся новый массив, в который копируются элементы.

17) Метод add(int index, T e) для добавления элемента в указанную позицию.
18) Метод addAll(int index, T[] a) для добавления элементов в указанную позицию.
19) Метод get(int index) для возвращения элемента в указанной позиции.
20) Метод indexOf(object o) для возвращения индекса указанного объекта, или -1, если его нет в векторе.
21) Метод lastIndexOf(object o) для нахождения последнего вхождения указанного объекта, или -1, если его нет в векторе.
22) Метод remove(int index) для удаления и возвращения элемента в указанной позиции.
23) Метод set(int index, T e) для замены элемента в указанной позиции новым элементом.
24) Метод subList(int fromIndex, int toIndex) для возвращения части вектора, т.е. элементов в диапазоне [fromIndex; toIndex).

25) Метод firstElement() для обращения к первому элементу вектора.

26) Метод lastElement() для обращения к последнему элементу вектора.
27) Метод removeElementAt(int pos) для удаления элемента в заданной позиции.
28) Метод removeRange(int begin, int end) для удаления нескольких подряд идущих элементов.

Вектор должен обладать следующими преимуществами:

1) Быстрый доступ по индексу. Скорость такой операции – O(1).
2) Быстрые вставка и удаление элементов с конца. Скорость операций опять же – O(1).

Предусмотреть обработку возможных ошибок.

Задача 11 (считывание из файла в вектор).

В вектор считать из файла input.txt строки (с пробелами). Из каждой строки вектора выделить подстроки, которые являются корректными IP-адресами (IPv4) и записать в новый вектор. Помимо корректности на IP-адрес в строке накладывается ограничение: IP-адреса не должны пересекаться друг с другом и числами (т.е. **121.121.121.121**.2 и **111.111.111.111**1 не должны учитываться). Реализовать **без** использования регулярных выражений. Второй вектор записать в файл output.txt.

Для хранения и обработки данных использовать собственную реализацию вектора MyVector. Использование встроенного вектора приведёт к незачёту задачи.

Задача 12 (стек).

Определить обобщённый класс MyStack, унаследованный от класса MyVector из задачи 6. Класс представляет собой реализацию стека (структуры данных, работающей по принципу LIFO – последним пришёл, первым ушёл).

Необходимо реализовать следующую функциональность:

1) Метод push(T item) для помещения элемента на вершину стека.
2) Метод pop() для извлечения верхнего элемента из стека.
3) Метод peek() для возвращения верхнего элемента стека без его извлечения.
4) Метод empty() для проверки, является ли стек пустым.

5) Метод search(T item) для поиска «глубины» объекта в стеке. Верхний элемент имеет позицию 1, находящийся под ним - 2 и т.д. Если объекта в стеке нет, метод должен возвращать -1.

При реализации методов MyStack следует использовать методы и поля родительского класса MyVector там, где это возможно и целесообразно.

Предусмотреть обработку возможных ошибок.

Задача 13 (обратная польская запись).

Напишите программу, которая вычисляет значение математического выражения, используя обратную польскую нотацию. Программа должна принимать на вход строку, содержащую математическое выражение, и выводить его значение.

Требования:

1) Программа должна быть реализована с использованием класса MyStack.
2) Программа должна включать метод, который переводит входное выражение в обратную польскую нотацию. Метод должен учитывать приоритеты операций и скобки.
3) Программа должна содержать метод, который вычисляет значение выражения в обратной польской нотации с использованием класса MyStack.
4) Программа должна поддерживать следующие операции: сложение, вычитание, умножение, деление, возведение в степень, квадратный корень, модуль числа, знак числа, синус, косинус, тангенс, натуральный логарифм, десятичный логарифм, минимум из двух чисел, максимум из двух чисел, остаток от деления, частное от деления, экспонента, отбрасывание дробной части числа.
5) Программа должна работать за O(n), где n – длина входной строки.
6) Программа должна обрабатывать возможные ошибки, такие как некорректный формат выражения, деление на ноль и т.д.
7) Программа должна принимать следующие аргументы командной строки:
    a. Математическое выражение, которое нужно вычислить (например, «3 + 4 * 2 / (1 - 5) ^ 2»).
    b. Список переменных, используемых в выражении, и их значения (например, «a=5 b=10 c=3»).

Пример входных данных:

3 + 4 * 2 / (1 - 5) ^ 2

Пример выходных данных:

3.5

Значения переменных должны быть запрошены у пользователя.

Подсказки:

1) используйте два стека: один для чисел, другой для операций и скобок;
2) при обработке входной строки, если текущий элемент – число или переменная, положите его значение в стек чисел;
3) если текущий элемент – открывающая скобка, положите её в стек операций;
4) если текущий элемент – закрывающая скобка, выполните все операции до открывающей скобки;
5) если текущий элемент – операция, то пока на вершине стека операций присутствует операция с приоритетом, большим либо равным приоритету текущей операции, выталкивайте операции из стека и выполняйте их;
6) после обработки всей строки выполните оставшиеся операции из стека операций.

Для хранения и обработки данных использовать собственную реализацию стека MyStack. Использование встроенного стека приведёт к незачёту задачи.

Задача 14 (двунаправленная очередь).

Определить обобщённый класс MyArrayDeque. Класс представляет собой реализацию двунаправленной очереди, которая может расти по мере необходимости.

Необходимы поля:

1) elements – массив обобщённого (универсального) типа E для хранения элементов;
2) head – индекс головы двунаправленной очереди;
3) tail – индекс хвоста двунаправленной очереди.

Необходимо реализовать следующую функциональность:

1) Конструктор MyArrayDeque() для создания пустой двунаправленной очереди с начальной вместимостью 16 элементов.
2) Конструктор MyArrayDeque(T[] a) для создания двунаправленной очереди из элементов передаваемого массива a.

3) Конструктор MyArrayDeque(int numElements) для создания пустой двунаправленной очереди с указанной вместимостью.

4) Метод add(T e) для добавления элемента в конец двунаправленной очереди. Если текущая ёмкость очереди недостаточна для добавления нового элемента, то ёмкость очереди удваивается. Затем элемент добавляется в конец очереди, и при необходимости индексы начала и конца очереди обновляются.
5) Метод addAll(T[] a) для добавления элементов из массива.
6) Метод clear() для удаления всех элементов из двунаправленной очереди.
7) Метод contains(object o) для проверки, находится ли указанный объект в двунаправленной очереди.
8) Метод containsAll(T[] a) для проверки, содержатся ли указанные объекты в двунаправленной очереди.
9) Метод isEmpty() для проверки, является ли двунаправленная очередь пустой.
10) Метод remove(object o) для удаления указанного объекта из двунаправленной очереди, если он есть там.
11) Метод removeAll(T[] a) для удаления указанных объектов из двунаправленной очереди.
12) Метод retainAll(T[] a) для оставления в двунаправленной очереди только указанных объектов.
13) Метод size() для получения размера двунаправленной очереди в элементах.
14) Метод toArray() для возвращения массива объектов, содержащего все элементы двунаправленной очереди.
15) Метод toArray(T[] a) для возвращения массива объектов, содержащего все элементы двунаправленной очереди. Если аргумент a равен null, то создаётся новый массив, в который копируются элементы.

16) Метод element() для возвращения элемента из головы двунаправленной очереди без его удаления.
17) Метод offer(T obj) для попытки добавления элемента obj в двунаправленную очередь. Возвращает true, если obj добавлен, и false в противном случае.
18) Метод peek() для возврата элемента из головы двунаправленной очереди без его удаления. Возвращает null, если двунаправленная очередь пуста.
19) Метод poll() для удаления и возврата элемента из головы двунаправленной очереди. Возвращает null, если двунаправленная очередь пуста.

20) Метод addFirst(T obj) для добавления obj в голову двунаправленной очереди.
21) Метод addLast(T obj) для добавления obj в хвост двунаправленной очереди.
22) Метод getFirst() для возвращения первого элемента двунаправленной очереди без его удаления.
23) Метод getLast() для возвращения последнего элемента двунаправленной очереди без его удаления.
24) Метод offerFirst(T obj) для попытки добавления obj в голову двунаправленной очереди. Возвращает true, если obj добавлен, и false при попытке добавить obj в полную двунаправленную очередь ограниченной ёмкости.
25) Метод offerLast(T obj) для попытки добавления obj в хвост двунаправленной очереди. Возвращает true, если obj добавлен, и false в противном случае.
26) Метод pop() для возвращения элемента из головы двунаправленной очереди с его удалением.
27) Метод push(T obj) для добавления элемента в голову двунаправленной очереди.
28) Метод peekFirst() для возвращения элемента из головы двунаправленной очереди без его удаления. Возвращает null, если двунаправленная очередь пуста.
29) Метод peekLast() для возвращения элемента из хвоста двунаправленной очереди без его удаления. Возвращает null, если двунаправленная очередь пуста.
30) Метод pollFirst() для возвращения элемента из головы двунаправленной очереди с его удалением. Возвращает null, если двунаправленная очередь пуста.
31) Метод pollLast() для возвращения элемента из хвоста двунаправленной очереди с его удалением. Возвращает null, если двунаправленная очередь пуста.
32) Метод removeLast() для возвращения элемента из конца двунаправленной очереди с его удалением.
33) Метод removeFirst() для возвращения элемента из головы двунаправленной очереди с его удалением.
34) Метод removeLastOccurrence(object obj) для удаления последнего вхождения obj из двунаправленной очереди. Возвращает true в случае успеха и false, если двунаправленная очередь не содержала obj.
35) Метод removeFirstOccurrence(object obj) для удаления первого вхождения obj из двунаправленной очереди. Возвращает true в случае успеха и false, если двунаправленная очередь не содержала obj.

Предусмотреть обработку возможных ошибок.

Задача 15 (считывание из файла в двунаправленную очередь).

Из файла input.txt вводить в двунаправленную очередь строки с пробелами следующим образом: если строка содержит больше десятичных цифр, чем первая строка в деке, то вставить её в конец, иначе в начало. Вывести результат в файл sorted.txt. После этого удалить из дека строки, содержащие более n пробелов (n – вводится с клавиатуры) и вывести оставшиеся строки на экран.

Для хранения и обработки данных использовать собственную реализацию двунаправленной очереди MyArrayDeque. Использование встроенной двунаправленной очереди приведёт к незачёту задачи.

Задача 16 (двунаправленный список).

Определить обобщённый класс MyLinkedList. Класс представляет собой реализацию двунаправленного списка, где каждый элемент структуры содержит указатели на предыдущий и следующий элементы.

Необходимы поля:

1) first – указатель на первый элемент двунаправленного списка;
2) last – указатель на последний элемент двунаправленного списка;
3) size – количество элементов в двунаправленном списке.

Необходимо реализовать следующую функциональность:

1) Конструктор MyLinkedList() для создания пустого двунаправленного списка.
2) Конструктор MyLinkedList(T[] a) для создания двунаправленного списка и заполнения его элементами из передаваемого массива a.
3) Метод add(T e) для добавления элемента в конец двунаправленного списка.
4) Метод addAll(T[] a) для добавления элементов из массива.
5) Метод clear() для удаления всех элементов из двунаправленного списка.
6) Метод contains(object o) для проверки, находится ли указанный объект в двунаправленном списке.
7) Метод containsAll(T[] a) для проверки, содержатся ли указанные объекты в двунаправленном списке.
8) Метод isEmpty() для проверки, является ли двунаправленный список пустым.

9) Метод remove(object o) для удаления указанного объекта из двунаправленного списка, если он есть там.
10) Метод removeAll(T[] a) для удаления указанных объектов из двунаправленного списка.
11) Метод retainAll(T[] a) для оставления в двунаправленном списке только указанных объектов.
12) Метод size() для получения размера двунаправленного списка в элементах.
13) Метод toArray() для возвращения массива объектов, содержащего все элементы двунаправленного списка.
14) Метод toArray(T[] a) для возвращения массива объектов, содержащего все элементы двунаправленного списка. Если аргумент a равен null, то создаётся новый массив, в который копируются элементы.

15) Метод add(int index, T e) для добавления элемента в указанную позицию.
16) Метод addAll(int index, T[] a) для добавления элементов в указанную позицию.
17) Метод get(int index) для возвращения элемента в указанной позиции.
18) Метод indexOf(object o) для возвращения индекса указанного объекта, или -1, если его нет в двунаправленном списке.
19) Метод lastIndexOf(object o) для нахождения последнего вхождения указанного объекта, или -1, если его нет в двунаправленном списке.
20) Метод remove(int index) для удаления и возвращения элемента в указанной позиции.
21) Метод set(int index, T e) для замены элемента в указанной позиции новым элементом.
22) Метод subList(int fromIndex, int toIndex) для возвращения части двунаправленного списка, т.е. элементов в диапазоне [fromIndex; toIndex).

23) Метод element() для возвращения элемента из головы двунаправленного списка без его удаления.
24) Метод offer(T obj) для попытки добавления элемента obj в двунаправленный список. Возвращает true, если obj добавлен, и false в противном случае.
25) Метод peek() для возврата элемента из головы двунаправленного списка без его удаления. Возвращает null, если двунаправленный список пуст.

26) Метод poll() для удаления и возврата элемента из головы двунаправленного списка. Возвращает null, если двунаправленный список пуст.

27) Метод addFirst(T obj) для добавления obj в голову двунаправленного списка.
28) Метод addLast(T obj) для добавления obj в хвост двунаправленного списка.
29) Метод getFirst() для возвращения первого элемента двунаправленного списка без его удаления.
30) Метод getLast() для возвращения последнего элемента двунаправленного списка без его удаления.
31) Метод offerFirst(T obj) для попытки добавления obj в голову двунаправленного списка. Возвращает true, если obj добавлен, и false в противном случае.
32) Метод offerLast(T obj) для попытки добавления obj в хвост двунаправленного списка. Возвращает true, если obj добавлен, и false в противном случае.
33) Метод pop() для возвращения элемента из головы двунаправленного списка с его удалением.
34) Метод push(T obj) для добавления элемента в голову двунаправленного списка.
35) Метод peekFirst() для возвращения элемента из головы двунаправленного списка без его удаления. Возвращает null, если двунаправленный список пуст.
36) Метод peekLast() для возвращения элемента из хвоста двунаправленного списка без его удаления. Возвращает null, если двунаправленный список пуст.
37) Метод pollFirst() для возвращения элемента из головы двунаправленного списка с его удалением. Возвращает null, если двунаправленный список пуст.
38) Метод pollLast() для возвращения элемента из хвоста двунаправленного списка с его удалением. Возвращает null, если двунаправленный список пуст.
39) Метод removeLast() для возвращения элемента из конца двунаправленного списка с его удалением.
40) Метод removeFirst() для возвращения элемента из головы двунаправленного списка с его удалением.
41) Метод removeLastOccurrence(object obj) для удаления последнего вхождения obj из двунаправленного списка. Возвращает true в случае успеха и false, если двунаправленный список не содержал obj.

42) Метод removeFirstOccurrence(object obj) для удаления первого вхождения obj из двунаправленного списка. Возвращает true в случае успеха и false, если двунаправленный список не содержал obj.

Двунаправленный список должен обладать следующими преимуществами:

1) Быстрые вставка и удаление элементов с начала и конца списка. Скорость операций – O(1).
2) Быстрое получение первого и последнего элементов списка. Скорость операций – O(1).

Предусмотреть обработку возможных ошибок.

Задача 17 (сравнение динамического массива и двунаправленного списка).

Сравнить эффективность динамического массива (MyArrayList) и двунаправленного списка (MyLinkedList) для различных операций на разных размерах данных структур данных.

Описание:

1) Сравнить эффективность динамического массива и двунаправленного списка для операций взятия элемента (get), присваивания элемента (set), добавления элемента (add), вставки элемента (add(i, value)), удаления элемента (remove).
2) Для операции добавления элемента (add) создать пустые динамический массив и двунаправленный список и выполнить 10^5, 10^6, 10^7, 10^8 операций добавления.
3) Для остальных операций (get, set, add(i, value), remove) создать динамический массив и двунаправленный список размером 10^5, 10^6, 10^7, 10^8 элементов.
4) Для каждого размера динамического массива и двунаправленного списка и каждой операции провести 20 запусков и вычислить среднее время выполнения.
5) Представить результаты сравнения в виде графиков, показывающих, какая из данных структур (динамический массив или двунаправленный список) работает эффективнее для каждой операции на разных размерах списков.

6) Проанализировать полученные результаты и сделать выводы об эффективности использования динамического массива и двунаправленного списка для различных операций на разных размерах данных структур. Объяснить, почему одна структура данных может быть более эффективной для определённых операций, чем другая, и как это связано с реализацией каждой структуры данных.