Вторая домашка будет посвящена проектированию классов. Предполагается, что хотя бы одна из двух структур данных – `subvector` или `subforwardlist` – у вас уже написана. Если нет, то можно будет дописать как раз в этой домашке.

1 Вектор и Список в ООП стиле

1. Перенести все функции для структуры данных `subvector` и `subforwardlist` внутрь типа данных (то есть сделать их методами класса)
2. Написать конструкторы и корректный деструктор
3. Написать все необходимое, чтобы тип данных удовлетворял правилу пяти

Интерфейс для методов (и только для них !) будет в interface_vector.cpp и interface_list.cpp
За корректно реализованный `subvector` — 1.5 балла, за `subforwardlist` — 2.5

2 Матрицы

В данном пункте нужно реализовать класс матрицы, и написать функцию, которая вычисляет её детерминант. Здесь будет два варианта задания на выбор, заключающиеся в различном способе хранения матрицы:

1. Реализовать простой вариант, в котором данные хранятся в стандартной структуре `std::vector`.
2. Реализовать вариант посложенее, в котором данные хранятся в написанной вами структуре `subvector`.

Интерфейс у классов общий, лежит в matrix_interface.cpp. Примечателен тот факт, что в обоих случаях использование правила нуля будет отличным и правильным решением, здесь не нужно писать ту кучу конструкторов и операторов, так как матрица не владеет памятью. Вся работа с памятью зашита в классы `std::vector` и `subvector`, и так как для них уже определено все, что нужно для правильного копирования, move и так далее, то для класса матрицы подойдут стандарные сгенерированные копилятором операции. Варианты задания различаются по сложности: в первом варианте за вас уже все написано в стандартной библиотеке, во втором же варианте придется дописать какие-то методы в класс subvector, например, переопределить `operator []` (как можно заметить, в первом задании доступ к элементам вектора вообще отсутствует).

Далее класс матрицы нужно будет умело использовать: написать внешнюю функцию, которая считает детерминант матрицы. Для проверки необходимо сгенерировать матрицу с заданным детерминантом. Подумайте о том, какие конструкторы вам могут пригодится для генерации матрицы, какие для генерации подматриц и так далее. Чем лучше вы спроектируете класс, тем больше удобства будет в его использовании.

Для сдачи необходимо написать три теста с матрицами размеров 5x5, 50x50 и 200x200, которые сгенерированы с заданным детерминантом (если расчет будет очень долгим, то вместо 200x200 возьмите 100x100). Проверьте, что детерминант матрицы и транспонированной матрицы совпадают.

За реализацию класса матрицы с помощью `std::vector` дается 4 балла, за реализацию через `subvector` дается 6 баллов.

В этом пункте есть доп. задание: Давайте напишем функцию, которая берет на вход ваш класс матрицы и `std::vector` (можете `subevector`) – правую часть – и решает линейную систему методом Гаусса. Проверка того, что решение системы корректно, можно произвести с помощью вычисления невязки $r = Ax - b$. Она должна быть достаточно мала по (евклидовой) норме. Для расчета невязки нужно определить оператор умножения матрицы на вектор и функцию, вычисляющую норму вектора.

Матрицу для расчета нужно сгенерировать случайным образом. При этом размер матрицы может быть произвольным (учтите, что метода Гаусса работает за $O(n^3)$ операций, и не стоит генерировать матрицы размером более 10^3).

Hint: проще всего метод Гаусса будет сделать, запрограммировав сначала LU разложение матрицы (ссылка на википедию). Ну и в целом это поможет не только в допонительном задании.

Доп. задание будет стоить 2 балла.

3 Оценивание

1. Vector – 1.5
2. List – 2.5
3. Матрица – 4 или 6
4. Доп. к матрице – 2

Итого 1.5 + 2.5 + 4(6) + 3 = 8(10) + 2 (доп)