Тема: инкапсуляция, конструкторы и перегрузка операторов

Вариант: 1.1.1

Задача: Реализовать класс для работы с квадратными матрицами целых чисел задаваемой размерности.

При этом в классе необходимо реализовать следующую функциональность:

1. Инициализация матрицы с помощью:
    a. Конструктора по умолчанию для инициализации матрицы размерности 0

    b. Конструктора, принимающего целочисленное значение, для инициализации единичной матрицы указанной размерности

    c. Конструктора, принимающего размерность и массив элементов, которые нужно расположить на главной диагонали, остальные элементы заполнить нулями

2. Перегружены операторы для:
    a. сложения и вычитания матриц

    b. умножения матриц

    c. сравнения двух матриц на равенство и неравенство

    d. транспонирования матрицы (может использоваться любой унарный оператор на выбор студента)

Замечание: при попытке сложения, вычитания или умножения матриц не совпадающих размеров программа должна заканчиваться с соответствующим сообщением об ошибке.

3. Ввод и вывод матрицы в заданный файл или на экран. При желании для этого пункта также можно перегружать операторы (на выбор студента)

4. Построение минора - новой матрицы, полученной из исходной удалением заданных строки и столбца. Для этого перегрузить оператор ():

```
Matrix a(10);
Matrix b = a(2, 3);
// матрица, полученная из a удалением второй
// строки и третьего столбца
```

5. Корректное управление динамической памятью во внутренней структуре класса. В классе не должно быть утечек памяти, некорректных указателей и т. д.

В качестве демонстрационного примера написать программу, считывающую из файла размерность **N,** значение **k** и матрицы *A*, *B*, *C*, *D* и возвращающую матрицу $(A + B \times C^T + K) \times D^T$, где *K* – диагональная матрица соответствующего размера, диагональные элементы которой равны **k**.

Входные данные:

В первой строке входного файла задается натуральное число **N** – размерность матриц. В следующей строке задается значение **k.** В следующих **4*N** строках содержатся описание матриц *A*, *B*, *C*, *D* соответственно: в каждой строке перечислены элементы через пробел.

Выходные данные:

В выходной файл записать N строк, содержащих описание матрицы $(A + B \times C^T + K) \times D^T$

Пример входных и выходных данных:

input.txt	output.txt
2 3 1 2 2 1 1 0 0 1 5 6 3 -2 1 2 3 4	19 47 12 32

Дополнительные задания:

Реализовать:

1. Обращение к строке матрицы по индексу, используя перегрузку оператора []
2. Обращение к столбцу матрицы, используя перегрузку оператора ()

Строки и столбцы матрицы в свою очередь должны давать доступ к своим элементам по индексу.

Любое изменение элементов в строках или столбцах матрицы, полученных соответствующими операциями, приводит к изменению элемента содержащей их матрицы.

Пример:

```
Matrix a(10);
a[5]; // взятие пятой строки
a[5][3] = 6; // запись в 3-ий элемент 5-ой строки
a(3); // взятие третьего столбца
a(3)[5] = 12; // 5-ый элемент 3-его столбца (тот же, что выше)
```

Тема: наследование и полиморфизм подтипов

Вариант: 2.1.2

Задача: Реализовать симуляцию «Волки и Зайцы», работающую по следующим правилам:

1. Симуляция происходит на поле размера NxM

2. Симуляция проходит в течение заданного количества ходов.

3. В симуляции участвует некоторое количество "зверей" двух видов - зайцы и волки.

   В каждый момент времени для любого зверя известны:

   - координаты, где он находится на поле
   - направление, куда он собирается двигаться (вверх, вниз, вправо, влево)
   - постоянство - раз в сколько ходов зверь захочет поменять направление своего движения
   - возраст - сколько ходов зверь уже существует. Изначально возраст зверя равен нулю.

4. Каждый ход симуляции состоит из следующих фаз:

   1. Движение

      На данном этапе каждый **волк** перемещается на 2 клетки в выбранном направлении, **заяц** перемещается на 1 клетку в выбранном направлении.

      Граничные условия – периодические, т.е. если существо переместилось за границу поля, то оно появляется на противоположной его стороне.

После движения зверь меняет направление своего движения на следующее по часовой стрелке, если того требует его постоянство.

2. *Питание*

   Если после фазы движения **волк** и **заяц** оказываются на одной клетке, то волк съедает зайца.

   В результате заяц **погибает**, а волк **насыщается**. Если волков и зайцев в клетке несколько, то первый же волк съедает **всех** зайцев.

   Первыми едят старшие волки, что это такое объяснено ниже.

3. *Старение*

   Возраст каждого живого зверя увеличивается на 1

4. *Размножение*

   Волк, съевший как минимум **двух** зайцев, размножается, порождая еще одного волка в той точке, где он находится. После этого волк снова голоден (для следующего размножения ему нужно будет съесть еще двух зайцев).

   Зайцы размножаются **дважды** за жизнь, достигнув возраста **5** и **10** соответственно.

   При рождении новый зверь движется в том же направлении, что и родитель. Параметр постоянства также совпадает с родительским. Возраст равен нулю.

5. *Вымирание*

На этом этапе умирают звери, достигшие максимального срока жизни.
Зайцы умирают, когда их возраст достиг 10. Волки – 15.

Нумерация зверей:

В некоторых шагах симуляции важно, какой из двух зверей является старшим (например при кормлении волков).

Старшим считается зверь, родившийся в более ранний ход.
В случае совпадения хода рождения старшим считается тот, кто:

1. Был первее записан во входных данных (для зверей родившихся в нулевой ход).

2. Родился от более старшего зверя (для остальных).

Реализация симуляции:

При реализации симуляции необходимо:

1. Создать иерархию зверей с базовым абстрактным классом Animal и наследниками Wolf и Rabbit

2. Создать класс Simulation, позволяющий создавать симуляции с заданным количеством зверей, расположенных в указанных координатах. И запускать симуляцию на заданное количество ходов.

3. Реализовать печать состояния симуляции после очередного хода следующим образом:

    a. Печатается поле размера NxM
    b. В каждой клетке стоит либо:
        - Символ #, если там сейчас нет зверей
        - Число k, если там k зайцев

- Число -k, если там сейчас k волков

Входные данные:

В первой строке входного файла указаны значения N, M и T – размеры поля и количество ходов.

Во второй строке указаны значения R и W – начальное количество зайцев и волков.

В следующих R строках описываются зайцы с помощью 4-ех значений x,y,d,k, где x,y – координаты на поле (точка с координатами {0,0} – верхний левый угол поля), d – начальное направление (направления кодируются 0 (север), 1 (восток), 2 (юг), 3(запад)), k – постоянство зверя.

В следующих W строках описываются волки (аналогично).

Выходные данные:

В выходные данные записать состояние симуляции после указанного количества ходов.

Пример входных и выходных данных:

input.txt	output.txt
3 3 3 2 1 1 2 1 1 1 1 0 2 0 2 1 2	### #-2# ###
4 4 20 1 1 0 0 1 100	8### #### ####

0 3 0 100	####

Дополнительное задание:

Добавить в иерархию еще один вид зверей: гиены.

Гиены во всем похожи на волков, но при этом очень непривередливы в еде. Они с удовольствием едят: зайцев, волков, других гиен.

С другой стороны гиены довольно трусливы, поэтому они останавливаются после двух съеденных животных (остальных не убивают).

Едят других животных по старшинству: сначала самых старых (и слабых).

Тема: шаблоны

Вариант: 3.1.1

Задача: Реализовать шаблонный класс:

```
template <typename K, typename V>
class HashMap {
    ...
}
```

реализующий хеш-таблицу - контейнер для хранения пар <"ключ", "значение">, где ключи имеют тип K, а значения - тип V. При этом, все ключи *уникальные*, т.е. не существует двух пар с одинаковыми ключами. Пары <"ключ", "значение"> далее будем также называть *элементами* коллекции.

Для данной коллекции реализовать следующие операции:

1. Поиск элемента в коллекции по заданному ключу.

2. Добавление нового элемента с заданными ключом и значением в коллекцию.

3. Удаление из коллекции элемента с заданным ключом.

Временная сложность в среднем данных операций должна быть $O(1)$(константной).

Замечания по реализации:

1. Запрещено пользоваться коллекциями из стандартной библиотеки шаблонов (`std::vector`, `std::map`, `std::set`, `std::unordered_map`, `std::unordered_set` и т.д.)

2. Для реализации хеш-функции предлагается использовать шаблонный класс `std::hash` из стандартной библиотеки

3. Для разрешения коллизий требуется использовать метод цепочек

4. Для достижения требуемой эффективности операций необходимо поддерживать разреженность хеш-таблицы. При достижении заданного в конструкторе процента заполненности, должна происходить перестройка таблицы: выделение дополнительной памяти под массив элементов, и повторное добавление в новый массив всех существующих элементов

5. Кроме того, необходимо реализовать служебный шаблонный класс Iterator, позволяющий перебрать все входящие в указанную таблицу элементы

6. В классе должно быть реализовано корректное управление динамической памятью: не должно быть утечек памяти, некорректных указателей и т. д

В качестве демонстрационного примера написать программу, которая считывает из файла набор команд для работы с хеш-таблицей, выполняет их и выводит результат.

Входные данные:

В первой строке входного файла заданы два символа,указывающих тип ключа и значения соответственно:
символ '**I**' обозначает тип `int`, '**D**' – `double`, '**S**' – `std::string`

Во второй строке задаётся число **N** – количество команд.

В следующих **N** строках описываются команды.
Команды могут быть двух типов:

- "**A** *key value*" – добавление в хеш-таблицу пары <*key*, *value*>. Если элемент с таким ключом уже есть в хеш-таблице, его значение должно быть обновлено на value

- "**R** *key*" – удаление из хеш-таблицы элемента с ключом *key*, если таковой имеется

Выходные данные:

В выходной файл записать два числа: общее количество элементов в таблице после выполнения команд, и количество "уникальных" элементов в таблице после выполнения команд.

Под "уникальными" понимаются элементы с разными value.

Пример входных и выходных данных:

input.txt	output.txt
I S 5 A 783 Ivanov A 999 Petrova A 986 Sidorov R 986 A 111 Kuznecov	3 3
S D 4 A Ivanov 1.84 A Petrova 1.66 A Ivanov 1.90 A Sidorov 1.90	3 2

Дополнительные задания:

1. Модифицировать свой класс, реализованный в задаче #1, так, чтобы он мог использоваться в качестве ключа в данной хеш-таблице

2. Реализовать шаблонный класс:

   ```
   template <typename K, typename T>
   class MultiHashMap {
       ...
   };
   ```

   Построенный по тем же принципам, что и HashMap, такой класс поддерживает хранение элементов с одинаковыми ключами.

   При этом:

   - поиск элемента по ключу возвращает *любой* элемент коллекции с заданным ключом
   - удаление по заданному ключу удаляет *все* элементы коллекции с таким ключом

   Кроме того, добавляются операции:

   - получить все элементы с заданным ключом
   - посчитать количество элементов с заданным ключом

Тема: Std, STL

Вариант: 4.1.2

Задача: Разработать класс для работы с детерминированными и недетреминированными конечными автоматам (ДКА и НКА).

НКА рассматриваются без ε-переходов.

Необходимо реализовать:

1. Считывание из файла ДКА или НКА: количество состояний, начальное состояние и множество конечных состояний, а также значения функции переходов.

2. Детерминацию — получение по имеющемуся НКА нового ДКА, распознающего то же множество последовательностей

3. Функцию, по данной строке проверяющую за линейное от её длины время, распознаётся ли строка заданным конечным автоматом

Замечания по реализации:

1. Для реализации рекомендуется использовать классы из стандартной библиотеки (std::vector, std::unordered_map, std::string, …)

2. В коде должно быть реализовано корректное управление динамической памятью: не должно быть утечек памяти, некорректных указателей и т. д

Входные данные:

Во входном файле задан автомат. Он может быть как детерминированным, так и недетерминированным.

В первой строке входного файла задано значение N – количество состояний автомата.

В следующей строке задано $0 \leq k < N$– номер начального состояния.

В следующей строке задано $0 \leq f < N$– количество конечных состояний.

В следующих f строках заданы номера конечных состояний автомата.

В следующей строке задано $0 \leq p$ – количество функций переходов.

В следующих p строках заданы функции переходов в формате $from\ to\ value$, где $0 \leq from < N;\ 0 \leq to < N$ – номера состояний автомата, а $value$ – символ, соответствующий значению функции перехода.

В следующей строке задано значение **T** – количество строк, распознаваемость конечным автоматом которых необходимо проверить. Наконец, следуют **T** строк, которые необходимо проверить заданным выше автоматом.

Выходные данные:

В выходной файл записать **T** строк, в каждой из которых записано "YES", если автомат распознал соответствующую строку и "NO" в противном случае.

Пример входных и выходных данных:

input.txt	output.txt
3 0 1 0 6 0 1 a 1 0 b 0 2 b 1 2 a 2 2 a 2 2 b 3 ab aaa abababab	YES NO YES
4 0 2 1 2 12 0 1 a 0 1 b 0 1 c 1 1 a 1 2 b 1 3 c 2 2 a 2 2 b 2 2 c 3 3 a 3 3 b 3 3 c 3 baaaaaaabcbabac cbacccc bcabcabcabcabc	YES YES NO

Дополнительные задания: ↓

Реализовать создание ДКА по регулярному выражению, заданному строкой

Тема: Многопоточность

Вариант: ξ

Задача: Реализовать web crawler: программу для перебора и скачивания интернет страниц. В условии задается адрес первой интернет страницы в формате:

<протокол>://<адрес>

Например:

https://habr.com/

Единственным обязательным для сдачи задачи является протокол "file://", позволяющий проводить тестирование на локальной базе данных:

file://input.html

Crawler должен:

1. Загрузить данную страницу на диск (в случае локального файла просто скопировать файл)
2. Найти в файле все ссылки на другие страницы. Ссылке искать в формате:

   <a href="<протокол>://<адрес>">

   ссылки в другом формате допускается проигнорировать

3. Произвести сканирование найденных страниц по тем же правилам (если они не были просканированы ранее)

4. Когда сканировать больше нечего, crawler должен вывести статистику: сколько времени он потратил на работу, сколько ссылок нашел.

Требования к решению:

1. Решение должно быть многопоточным: обработка графа страниц должно производится множеством рабочих потоков.

2. Должна присутствовать возможность варьирования множества рабочих потоков. Добавление второго рабочего потока (т.е. переход к многопоточности) не должно увеличивать суммарное время работы всей программы.
3. Необходимо найти предел количества рабочих потоков, после которого добавление новых потоков не уменьшает время работы приложения.

Дополнительная задача:

Поддержать другие протоколы кроме "file://" и протестировать решение на реальных страницах в интернете. Для скачивания страниц по сети следует использовать сторонние библиотеки, например: libcurl

Выполнение этой доп. задачи для получения автомата **не требуется**.

Формат входных данных:

<адрес стартовой страницы> <количество потоков работников>

Формат выходных данных:

<количество посещенных страниц> <общее потраченное время>

Тестовая локальная база страниц:

https://drive.google.com/file/d/1OzSUBeEMSNAazwgP397cjNKla0YsZjI1/