Список задач на классы

Внимание. Для каждого класса должно быть перегружено несколько операций (либо как методы класса либо при помощи отдельных функций). Классы должны быть объявлены в заголовочных файлах. Реализацию "длинных" методов (больше одной команды в теле метода) нужно писать в файлах исходного кода. Программа должна содержать объявление и реализацию класса, а также функцию `main()` и примеры работы с каждым методом класса. Во всех задачах запрещается пользоваться библиотекой контейнеров C++.

1. Реализовать класс `R3Vector`, представляющий вектор в трёхмерном пространстве. В числе прочих должны быть реализованы:

   - Конструкторы;
   - Арифметические операции `+`, `+=`, `-`, `-=`, скалярное произведение `*` и умножение на число.
   - Векторное произведение, например, `&`.
   - Операции сравнения `==`, `!=`, `<`, `<=`, `>`, `>=`.
   - Операция `>>` для класса `std::istream` и операция `<<` для класса `std::ostream` для ввода/вывода вектора.
   - Должен быть реализован метод для вычисления длины вектора.

   В качестве теста написать с использованием этого класса консольную программу, которая

   - находит расстояние от точки до плоскости (плоскость задается тремя точками);
   - находит угол между двумя векторами в трехмерном пространстве;
   - находит расстояние между двумя скрещивающимися прямыми (прямая задается точкой и направляющим вектором).

   Программа должна использовать как можно больше перегруженных операций класса.

2. Реализовать класс `Real` (вещественное число), использующий представление вещественного числа с фиксированной десятичной точкой. Число представляется с точностью до 10^{-3} в виде $M \cdot 10^{-3}$, где M — целое число. Целую и дробные части следует хранить в отдельных переменных. Реализация всех методов должна использовать исключительно **целочисленную арифметику**. В числе прочих должны быть реализованы

   - Конструкторы;
   - Преобразование числа в строку и строки в число.
   - Арифметические операции `+`, `+=`, `-`, `-=`, `*`, `*=`, `/`, `/=`.
   - Операции сравнения `==`, `!=`, `<`, `<=`, `>`, `>=`.
   - Вычисление квадратного корня.
   - Операция `>>` для класса `std::istream` и операция `<<` для класса `std::ostream` для ввода/вывода числа.

   Запрещается домножать число на 10^3 для того, чтобы представить его при помощи одной переменной.

   В качестве тестовой программы написать программу решения квадратного уравнения, корни находятся с точностью 10^{-3}.

3. Реализовать класс `Polynomial`, представляющий многочлен произвольной степени с операциями сложения, умножения и получения степени, а также деления с остатком и вычисления НОД двух многочленов. В числе прочих должны быть реализованы

- Правильные конструкторы (в том числе и конструктор копирования), деструктор и операция присваивания `=`.
- Арифметические операции `+`, `+=`, `-`, `-=`, `*`, `*=`.
- Операции сравнения `==`, `!=`, `<`, `<=`, `>`, `>=`.
- Деление с остатком.
- НОД двух многочленов.
- Возведение многочлена в степень.
- Операция `>>` для класса `std::istream` и операция `<<` для класса `std::ostream` для ввода/вывода многочлена.

4. Реализовать класс `Set`, представляющий из себя множество целых чисел. Класс должен поддерживать операции объединения, пересечения и симметричной разности множеств, а также операции определения принадлежности элемента и операции сравнения множеств (входит ли множество в данное). В числе прочих должны быть реализованы

   - Правильные конструкторы (в том числе и конструктор копирования), деструктор и операция присваивания `=`.
   - Операции объединения `|`, `|=`, пересечения `&`, `&=` и симметричной разности множеств `^`, `^=`.
   - Операции сравнения `==`, `!=`, `<`, `<=`, `>`, `>=`. Последние четыре операции проверяют, является ли множество подмножеством другого.
   - Операции определения принадлежности элемента множеству.
   - Операция `>>` для класса `std::istream` и операция `<<` для класса `std::ostream` для ввода/вывода множества.

5. Реализовать класс `String`, являющийся оболочкой над строкой. Должны быть реализованы

   - Правильные конструкторы (в том числе и конструктор копирования), деструктор и операция присваивания `=`.
   - Операции `+` и `+=` для конкатенации строк.
   - Операции `[]` и `()` для чтения и записи отдельных символов.
   - Операции сравнения `==`, `!=`, `<`, `<=`, `>`, `>=`.
   - Операция `>>` для класса `std::istream` и операция `<<` для класса `std::ostream` для ввода/вывода строки.

6. Реализовать класс `Substitution`, представляющий подстановку порядка N, т. е. биективное отображение конечного множества из N элементов в себя. Должны быть реализованы композиция подстановок, действие подстановки на элемент X, $0 \leqslant X \leqslant N-1$, определение четности подстановки. Должны быть реализованы

   - Правильные конструкторы (в том числе и конструктор копирования), деструктор и операция присваивания `=`.
   - Композиция подстановок `*`.
   - Действие подстановки на элемент `*`.
   - Определение четности подстановки.
   - Операция `>>` для класса `std::istream` и операция `<<` для класса `std::ostream` для ввода/вывода подстановки.
   - Напишите программу, вычисляющую определитель матрицы при помощи класса подстановок.

7. Реализовать класс `Matrix`, представляющий прямоугольную матрицу порядка $N \times N$, где N задаётся в конструкторе. Должны быть реализованы получение элемента матрицы с заданными индексами (для чтения и для записи), сумма и произведение матриц, единичная матрица, действие матрицы на вектор размерности N, транспозиция, элементарные преобразования, приведение к ступенчатому виду, вычисление определителя, обратной матрицы и т.д. Должны быть реализованы

   - Правильные конструкторы (в том числе и конструктор копирования), деструктор и операция присваивания `=`.
   - Арифметические операции `+`, `+=`, `-`, `-=`, `*`, `*=`.
   - Операции сравнения `==`, `!=`, `<`, `<=`, `>`, `>=`.
   - Операции `[]` и `()` для чтения и записи отдельных элементов.
   - Операция `>>` для класса `std::istream` и операция `<<` для класса `std::ostream` для ввода/вывода матрицы.
   - Действие матрицы на вектор размерности N, транспозиция, элементарные преобразования, приведение к ступенчатому виду, вычисление определителя, обратной матрицы и т.д.

8. Битовое множество произвольной длины `BitSet`. Для хранения битов лучше всего использовать беззнаковые типы фиксированной длины `std::uint32_t` или `std::uint64_t` из заголовочного файла `<cstdint>`. Тогда битовое множество хранится в виде массива, в каждом элементе которого лежат 32 или 64 бита соответственно. Должны быть реализованы

   - Правильные конструкторы (в том числе и конструктор копирования), деструктор и операция присваивания `=`.
   - Битовые операции `&`, `&=`, `|`, `|=`, `~`, `^`, `<<`, `<<=`, `>>`, `>>=`.
   - Операции сравнения `==`, `!=`, `<`, `<=`, `>`, `>=` как двоичных чисел.
   - Операции `[]` и `()` для чтения и записи отдельных битов.
   - Операция `>>` для класса `std::istream` и операция `<<` для класса `std::ostream` для ввода/вывода битового множества.

9. Реализовать класс рациональной дроби `Rational`. Рациональная дробь должна всегда быть представлена в виде двух целых чисел: числителя и знаменателя. В реализации класса `Rational` они должны удовлетворять следующим правилам:

   (a) Знаменатель всегда положителен.

   (b) Если числитель равен нулю, то знаменатель должен быть равен единице.

   (c) Наибольший общий делитель числителя и знаменателя должен быть равен единице, то есть дробь должна быть представлена в несократимом виде.

   Необходимо перегрузить следующие операции:

   - Унарный оператор `-`.
   - Арифметические операции `+`, `+=`, `-`, `-=`, `*`, `*=`, `/`, `/=`.
   - Операции сравнения `==`, `!=`, `<`, `<=`, `>`, `>=`.
   - Операция `>>` для класса `std::istream` и операция `<<` для класса `std::ostream` для ввода/вывода рационального числа.
   - Операции приведения к типу `int` и к типу `double`.

10. Класс `Decimal` для работы с числами фиксированной точности в десятичной системе исчисления. Длина числа может быть произвольной. Количество цифр в дробной части фиксировано. Сами цифры должны быть от 0 до 9 включительно. Для простоты реализации цифры можно хранить в массиве отдельными элементами. Необходимо реализовать:

- Правильные конструкторы (в том числе и конструктор копирования, конструктор из числа типа `int` и числа типа `double`), деструктор и операцию присваивания `=`.
- Арифметические операции `+`, `+=`, `-`, `-=`, `*`, `*=`, `/`, `/=`.
- Операции сравнения `==`, `!=`, `<`, `<=`, `>`, `>=`.
- Операции `[]` и `()` для чтения и записи отдельных цифр.
- Операция `>>` для класса `std::istream` и операция `<<` для класса `std::ostream` для ввода/вывода числа.
- Операции приведения к типу `int` и к типу `double`.

11. Реализовать класс очереди `Queue` целых чисел. Этот класс позволяет добавить элемент в конец очереди и вытащить элемент из начала очереди. Сами элементы следует хранить в обычном массиве. Необходимо реализовать:

    - Правильные конструкторы (в том числе и конструктор копирования), деструктор и операция присваивания `=`.
    - Арифметические операции `+`, `+=` для объединения двух очередей в одну.
    - Операции для поэлементного сравнения и сравнения в лексикографическом порядке `==`, `!=`, `<`, `<=`, `>`, `>=`.
    - Операцию `>>` для класса `std::istream` и операцию `<<` для класса `std::ostream` для ввода/вывода очереди.
    - Операцию `>>` для того, чтобы вытащить из очереди первый элемент и операцию `<<` для того, чтобы положить в очередь элемент.

12. Реализовать класс для кодирования и декодирования текста при помощи азбуки Морзе

    https://ru.wikipedia.org/wiki/Азбука_Морзе.

    Класс должен хранить закодированный текст во внутренней памяти (в обычном массиве элементов типа `char`). Количество информации не ограничено. Необходимо реализовать:

    - Правильные конструкторы (в том числе и конструктор копирования), деструктор и операцию присваивания `=`.
    - Арифметические операции `+`, `+=` для объединения двух кодов в один.
    - Операции для сравнения `==`, `!=`.
    - Операцию `>>` для класса `std::istream` и операцию `<<` для класса `std::ostream` для ввода/вывода кода. Эта операция предназначена для ввода/вывода закодированной информации, хранящейся в классе.
    - Операцию `>>` для того, чтобы вытащить из кода первый символ или первое слово и операцию `<<` для того, чтобы добавить к коду один символ или строку. Иными словами, операция `<<` кодирует один входной символ или входное слово и добавляет его в конец уже закодированной последовательности, хранящейся в классе. Операция `>>` вытаскивает из начала закодированной последовательности, хранящейся в классе, один символ или слово (декодирует его), оставшийся код сдвигается в начало массива. Требуется реализовать по одной операции как для отдельных символов (тип `char`) так и для слов (тип `char*`).

13. Реализовать класс для приближения функции при помощи интерполяционного многочлена Лагранжа. Вычисление значения многочлена Лагранжа должно выполняться при помощи заранее вычисленных коэффициентов многочлена Лагранжа, которые хранятся в классе. Необходимо реализовать

    - Правильные конструкторы (в том числе и конструктор копирования), деструктор и операцию присваивания `=`.

- Операции для сравнения `==`, `!=`.
- Операцию `<<` для класса `std::ostream` для вывода всех коэффициентов многочлена.
- Операцию `double& operator[](double)` при помощи которой в класс можно было бы добавить новую точку и значение. Например, `approximator[x] = y;`.
- Операцию функционального вызова `()` при помощи которой в класс можно было бы добавить новую точку и значение. Например, `approximator(x, y)`.
- Операцию функционального вызова `()` при помощи которой можно было бы получить значение многочлена Лагранжа в указанной точке.
- Метод для удаления точки.

14. Реализовать класс для приближения функции при помощи интерполяционного многочлена Лагранжа. Вычисление значения многочлена Лагранжа должно выполняться при помощи интерполяционной формулы Ньютона. Разделённые разности следует вычислить заранее и хранить в классе. Необходимо реализовать

    - Правильные конструкторы (в том числе и конструктор копирования), деструктор и операцию присваивания `=`.
    - Операции для сравнения `==`, `!=`.
    - Операцию `<<` для класса `std::ostream` для вывода всех разделённых разностей.
    - Операцию `double& operator[](double)` при помощи которой в класс можно было бы добавить новую точку и значение. Например, `approximator[x] = y;`.
    - Операцию функционального вызова `()` при помощи которой в класс можно было бы добавить новую точку и значение. Например, `approximator(x, y)`.
    - Операцию функционального вызова `()` при помощи которой можно было бы получить значение многочлена Лагранжа в указанной точке.
    - Метод для удаления точки.

15. Реализовать класс для приближения функции при помощи кусочно-линейной интерполяции. Необходимо реализовать

    - Правильные конструкторы (в том числе и конструктор копирования), деструктор и операцию присваивания `=`.
    - Операции для сравнения `==`, `!=`.
    - Операцию `<<` для класса `std::ostream` для вывода всех точек и значений функции в этих точках.
    - Операцию `double& operator[](double)` при помощи которой в класс можно было бы добавить новую точку и значение. Например, `approximator[x] = y;`.
    - Операцию функционального вызова `()` при помощи которой в класс можно было бы добавить новую точку и значение. Например, `approximator(x, y)`.
    - Операцию функционального вызова `()` при помощи которой можно было бы получить значение приближения в указанной точке.
    - Метод для удаления точки.

16. Реализовать класс для сохранения наблюдений о погоде. Класс должен хранить список дат и температур, соответствующих этим датам. Необходимо реализовать

    - Правильные конструкторы (в том числе и конструктор копирования), деструктор и операцию присваивания `=`.

- Операции для сравнения `==`, `!=`.
- Операцию `<<` для класса `std::ostream` для вывода всех наблюдений.
- Операцию `+` для объединения двух журналов в один. Реализовать аналогичную операцию `+=`.
- Операцию функционального вызова `()` при помощи которой в класс можно было бы добавить новое наблюдение. Например, `weatherJournal(year, month, day) = temperature`. Операция должна делать проверку того, что температура не меньше абсолютного нуля. Если температура меньше абсолютного нуля, то операция оставляет объект в исходном состоянии. **Подсказка:** подумайте, какой тип данных должна возвращать эта операция.
- Метод для удаления наблюдения.
- Метод для вывода всех наблюдений от одной даты до другой в порядке увеличения даты.

17. Реализовать класс — толковый словарь. Он должен содержать список пар (слово, перевод). Необходимо реализовать

    - Правильные конструкторы (в том числе и конструктор копирования), деструктор и операцию присваивания `=`.
    - Операции для сравнения `==`, `!=`.
    - Операцию `<<` для класса `std::ostream` для вывода всех слов и их переводов в алфавитном порядке.
    - Операцию `+` для объединения двух словарей в один. Реализовать аналогичную операцию `+=`.
    - Операцию `[]` при помощи которой в класс можно было бы добавить новую пару (слово, перевод). Например, `dictionary[word] = translation`.
    - Метод для вывода всех пар (слово, перевод) в алфавитном порядке от одного слова до другого.
    - Метод для удаления слова.

18. Реализовать класс — базу заработных плат сотрудников. Класс должен содержать фамилии сотрудников и их оклады. Необходимо реализовать

    - Правильные конструкторы (в том числе и конструктор копирования), деструктор и операцию присваивания `=`.
    - Операции для сравнения `==`, `!=`.
    - Операцию `<<` для класса `std::ostream` для вывода всех сотрудников и их окладов в алфавитном порядке.
    - Операцию `+` для объединения двух баз словарей в одну. Реализовать аналогичную операцию `+=`.
    - Операцию `[]` при помощи которой в класс можно было бы добавить новую пару (фамилия, оклад). Например, `salaryDB[name] = value`. Операция должна проверять, что значение не меньше МРОТ. Если меньше, то операция оставляет класс в исходном состоянии. **Подсказка:** подумайте о том, какой тип данных должна возвращать операция.
    - Метод для удаления записи.
    - Метод для вывода всех сотрудников, чей оклад находится в заданных границах, в алфавитном порядке.

19. Реализовать класс множества точек на плоскости. Необходимо реализовать

    - Правильные конструкторы (в том числе и конструктор копирования), деструктор и операцию присваивания `=`.

- Операции для сравнения `==`, `!=`.
- Операцию `<<` для класса `std::ostream` для вывода всех точек.
- Операцию функционального вызова `()` для добавления точки. Например, `points(x, y);`.
- Метод для удаления точки.
- Метод, который позволяет вывести все точки, которые находятся внутри указанного прямоугольника.
- Операции `|`, `&`, `-` для объединения, пересечения и разности двух множеств. Также реализовать операции `|=`, `&=`, `-=`.

20. Реализовать класс для разбиения строк на подстроки по заданным разделителям. Необходимо реализовать

    - Правильные конструкторы (в том числе и конструктор копирования), деструктор и операцию присваивания `=`.
    - Операции для сравнения `==`, `!=`.
    - Метод, который позволяет задать список разделителей.
    - Операцию `<<`, которая добавляет строку во внутреннюю память класса. Операция должна добавить строку-аргумент в конец строки, записанной во внутренней памяти класса.
    - Операцию `>>`, которая возвращает часть строки до следующего разделителя. Операция извлекает из начала строки, записанной во внутренней памяти класса, до первого разделителя, и записывает эту часть в аргумент. Оставшаяся часть строки (после первого разделителя) во внутренней памяти класса сдвигается в начало массива.