Правительство Российской Федерации

Федеральное государственное автономное образовательное учреждение
высшего образования «Национальный исследовательский университет
«Высшая школа экономики»

Кафедра «Компьютерная безопасность»

ОТЧЕТ
К ЛАБОРАТОРНОЙ РАБОТЕ №1

по дисциплине

«Языки программирования»

Работу выполнил
студент группы СКБ-203 _____ К.А. Павкин
подпись, дата

Работу проверил _____ С.А. Булгаков
подпись, дата

Москва 2021

Содержание

Постановка задачи **2**

1 Алгоритм решения задачи **3**
1.1 Дата и время . . . 3
1.2 Целое произвольной длины . . . 3
1.3 Год «от Адама» . . . 4
1.4 Разреженная матрица . . . 5

2 Выполнение задания **6**

3 Получение исполняемых модулей **7**

4 Тестирование **8**
4.1 Прецеденты для класса date . . . 8
4.2 Прецеденты для класса number . . . 8
4.3 Прецеденты для класса adam . . . 9
4.4 Прецеденты для класса matrix . . . 9

Приложение А **10**
Файл number.h . . . 23
Файл adam.h . . . 43
Файл matrix.h . . . 52
Файл CMakeLists.txt . . . 62

«Пользовательский тип»

Разработать программу на языке Си++ (ISO/IEC 14882:2014), демонстрирующую решение поставленной задачи.

Постановка задачи

Разработать набор классов, объекты которых реализуют типы данных, указанные ниже. Для классов разработать необходимые конструкторы, деструктор, конструктор копирования, а также методы, обеспечивающие изменение отдельных составных частей объекта. Используя перегрузку операторов (operator) разработать стандартную арифметику объектов, включающую арифметические действия над объектами и стандартными типами (целыми, вещественными, строками – в зависимости от вида объектов), присваивание, ввод и вывод в стандартные потоки (используя операторы «<<» и «>>»), приведение к/от базового типа данных. Организовать операции в виде конвейера значений, с результатом (новым объектом) и сохранением значений входных операндов.

Задачи

1. Дата и время, представленные целочисленными переменными: год, месяц, день, час, минута, секунда. Базовый тип: `uint64_t` формат представления unix time. Реализовать возможность преобразования в/из формата представления filetime.

2. Целое произвольной длины (во внешней форме представления в виде строки символов-цифр). Базовый тип: `std::string`.

3. Год «от Адама», имеющий внутреннее представление в виде целочисленных переменных: индикт, круг солнцу, круг луне. Диапазоны значений (циклические): индикт 1—15, круг солнцу 1—28, круг луне 1—19. Ежегодно каждая переменная увеличивается на 1. Итоговое значение вычисляется как произведение переменных. Необходима возможность отображения/задания как в виде одного числа, так и виде трех. Реализовать возможность преобразования в/из формата представления «от рождества Христова» используя соответствие 1652 = 7160 «от Адама».

4. Разреженная матрица, представленная динамическим массивом структур, содержащих описания ненулевых коэффициентов: индексы местоположения коэффициента в матрице (целые) и значение коэффициента (вещественное).

1 Алгоритм решения задачи

В процессе анализа предметной области были выявлены основные алгоритмы, необходимые для реализации всеобъемлющей, асимптотически эффективной и отлаженной программы для решения поставленной задачи.

1.1 Дата и время

Учитывая формат представления даты в времени, становится заметен ньанс при написании валидации аргументов сеттера для дня. Дело в том, что максимальное значение месяца в февраля не является константным. Появляется необходимость понять, является ли год високосным. Более того, как будет известно далее, полезно также знать, сколько лет в некотором диапазоне были високосными. Для этого были написаны функции из листинга 1.

```
int date::leap_years_until(int year)
{
        return year / 4 - year / 100 + year / 400;
}

int date::leap_years_between(int start, int end)
{
        return leap_years_until(end) - leap_years_until(start - 1);
}

bool date::is_leap_year(int year)
{
        return leap_years_between(year, year) == 1;
}
```

Листинг 1: Функции для работы с високосными годами

Перевод из/в формат представления unix time было решено вычислять без использования каких-либо библиотек. Посчитав количество лет с начала 1970 года, становится возможным рассчитать количество дней, прошедших с начала текущего года. По аналогии рассчитывается количество секунд, прошедших с начала дня.

Перевод из/в формат представления file time имеет несущественные различия по отношению к уже написаннымм преобразованями.

1.2 Целое произвольной длины

Целое число произвольной длины, позволяющее работать с числами без ограничений, было решено хранить в виде динамического массива битов (булевых переменных), где крайний правый бит отвечает за знак. Такое решение позволяет свести к минимуму простои памяти, а также даёт возможность поддерживать побитовые операции с привычной для них логикой. В процессе разработки библиотеки был реализован широкий спектр побитовых, арифметических и логических операторов, а также полная, бесшовная совместимость со встроенным типом `int`.

Если побитовые операции были реализованы практически нативно, с поэлементным применением логических операторов к двум соседним битам двух чисел, то арифметические операции, основываясь на побитовых, уже требовали базовых знаний математики для сложения,

умножения и деления «в столбик».

При применении бинарного оператора к числам, имеющим различную вместительность, вызывется функция `resize` в отношении числа с меньшей вместительностью, которая его увеличивает. Данная функция представлена в листинге 2.

```
void number::resize(size_t capacity)
{
        bool* new_binary = new bool[capacity];

        size_t min = std::min(this->capacity, capacity);

        for (size_t i = 0; i < min - 1; i++) {
                new_binary[i] = binary[i];
        }

        for (size_t i = min - 1; i < capacity; i++) {
                new_binary[i] = binary[this->capacity - 1];
        }

        this->capacity = capacity;

        binary = new_binary;

        delete[] new_binary;
}
```

Листинг 2: Функция, меняющая вместительность числового типа

Код выше создаёт число с новой вместительностью, копирует в него все элементы из старого, а оставшееся место заполняет значением знака. Последнее действие необходимо для поддержания знака числа при изменении его вместительности. Минимальное значение вместительности ограничено 32 битами.

Отдельного внимания требует логика вычислений: вместительность числа **не пересчитывается** в процессе выполнения какого-либо оператора. Таким образом, класс не нивелирует опасность переполнения типа в случае, например, перемножения слишком больших чисел. Для контроля переполнения в классе предусмотрена функция `get_capacity`, возвращающая текущую вместительность числа.

Вдумчивый читатель уже заметил ощутимый недостаток от подобного решения, заключающийся в том, что двоичное число нельзя побитово перевести в строку и вывести в поток вывода. Пользователям просто непривычен двоичный формат представления чисел. Для перевода в десятичную систему счисления предусмотрены закрытые функции сложения двух строк и деления строки на два, пользующиеся спросом у операторов чтения и записи.

1.3 Год «от Адама»

Год «от Адама» — система летоисчисления «от сотворения мира» (начинается с 5508 года до н. э.). В качестве представления было предложено использовать независимые переменные: индикт (1—15), круг солнцу (1—28) и круг луне (1—19). Ежегодно каждая переменная увеличивается

на единицу, и, превышая заданный диапазон, сбрасывается до первоначального состояния.

Таким образом, год «от Адама» вычисляется как решение системы линейных алгебраических уравнений по трём переменным, в частности, с использованием китайской теоремы об остатках (см. листинг 3).

```
int adam::to_adam_year() const
{
        int m1 = MAX_ADAM_YEAR / 15, y1;
        int m2 = MAX_ADAM_YEAR / 28, y2;
        int m3 = MAX_ADAM_YEAR / 19, y3;

        for (int y = 0, m = m1 % 15; y < 15; y++)
                if ((m * y) % 15 == indict)
                        y1 = y;
        for (int y = 0, m = m2 % 28; y < 28; y++)
                if ((m * y) % 28 == sun)
                        y2 = y;
        for (int y = 0, m = m3 % 19; y < 19; y++)
                if ((m * y) % 19 == moon)
                        y3 = y;

        return (m1 * y1 + m2 * y2 + m3 * y3) % MAX_ADAM_YEAR;
}
```

Листинг 3: Алгоритм нахождения года «от Адама»

Сами же переменные вычисляются как остаток от деления года «от Адама» на 15, 28 и 19 соответственно.

1.4 Разреженная матрица

Разреженной матрицей называют такую матрицу, которая наполнена преимущественно нулевыми элементами. Для такой матрицы наиболее оптимальным способом хранения является так называемый список смежности, где для каждой строки существуют только заполненные значения в соответствующих колонках — `map<size_t, double>[]`. В процессе вычислений итерация происходит не по всем элементам матрицы, а лишь по заполненным, что является асимтотически эффективным решением поставленной задачи.

Помимо базовых операций с матрицами были реализованы операторы сложения и умножения матриц, оператор умножения матрицы на число, функции для получения определителя, обратной и транспонированной матрицы. В логических операторах учтены ньансы при работе с типом double — для равенства двух переменных модуль их разницы должен быть меньше 1e-6.

2 Выполнение задания

На рисунке 1 представлена диаграмма классов, демонстрирующая формат хранения данных, а также критически важные методы. В целях сужения диаграммы, операторы и некоторые функции были опущены. Зависимостей между классами нет.

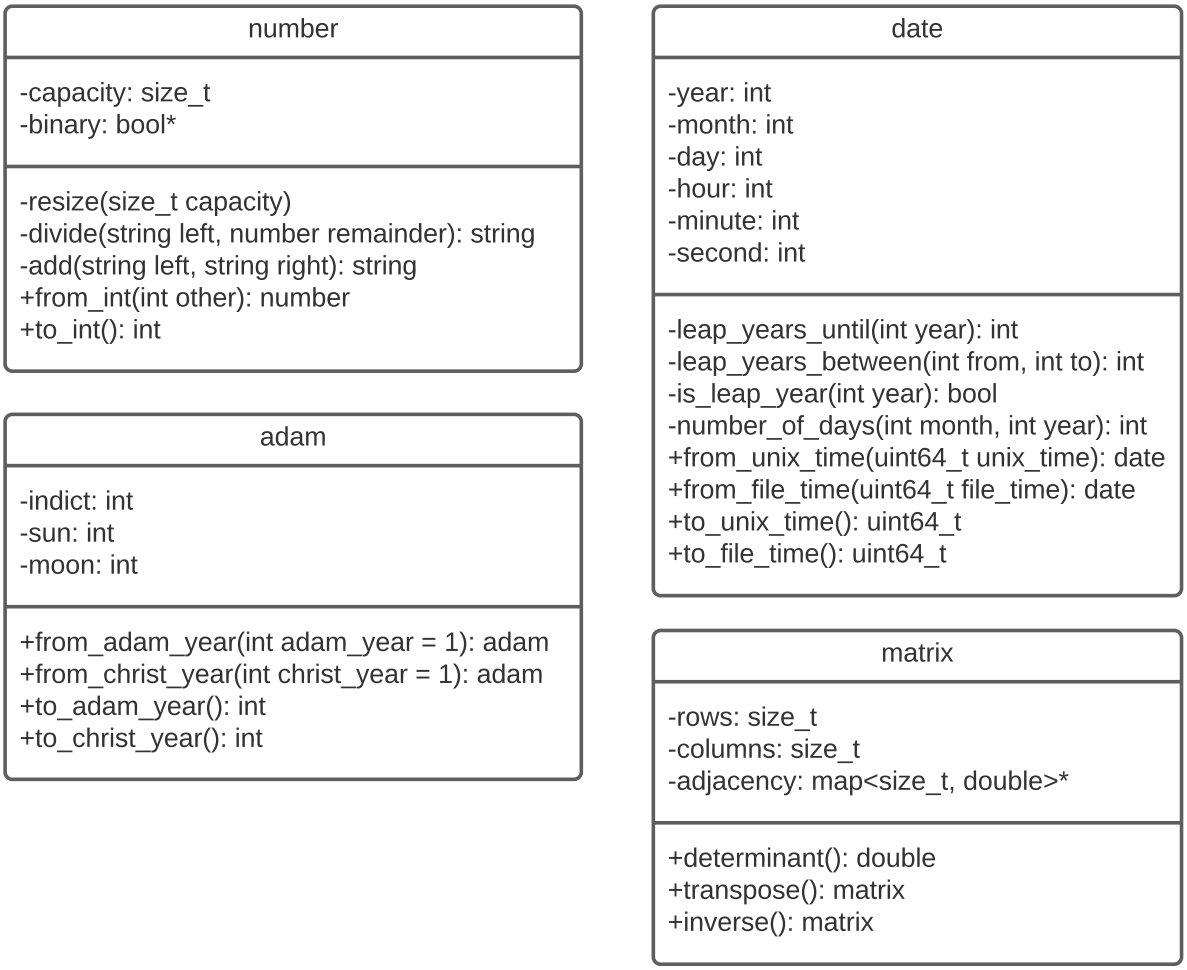

Рис. 1: Диаграмма классов

Заголовочные файлы созданных библиотек приведены в приложении А.

1. Дата и время — `date.h`4.4
2. Целое произвольной длины — `number.h`4.4
3. Год «от Адама» — `adam.h`4.4
4. Разреженная матрица — `matrix.h`4.4

3 Получение исполняемых модулей

Заголовочные файлы и их реализации были размещены в различных директориях. Для сборки проекта в каждую директорию были помещены файлы CMakeLists.txt, и, таким образом, каждая директория представляет из себя отдельную библиотеку, которая в последствии собирается в корневом каталоге. Файл сборки CMakeLists.txt из корневого каталога показан в приложении A.4.4

4 Тестирование

Файл `main.cpp` содержит юнит-тесты, целью которых является сравнение ожидаемых от функций значений с фактическими, и выдача оповещений при их несовпадении.

4.1 Прецеденты для класса date

1. unix_time_conversion — проверяет коррентность преобразований из/в формат представления unix time (количество секунд, прошедших с начала 1970 года)
2. file_time_conversion — проверяет коррентность преобразований из/в формат представления file time (количество интервалов по 100 наносекунд, прошедших с начала 1601 года)
3. addition — проверяет коррентность сложения даты с числом или другой даты
4. subtraction — проверяет коррентность вычитания из даты числа или другой даты
5. increment — проверяет коррентность инкрементов
6. decrement — проверяет коррентность декрементов

4.2 Прецеденты для класса number

1. string_conversion — проверяет коррентность преобразования из/в строковый (десятичный) формат представления `std::string`
2. int_conversion — проверяет коррентность преобразования из/во встроенный числовой тип `int`
3. bitwise_not — проверяет корректность унарного оператора побитового НЕ
4. bitwise_and — проверяет корректность оператора побитового И в отношении другого числа (number) или встроенного типа `int`
5. bitwise_or — проверяет корректность оператора побитового ИЛИ в отношении другого числа (number) или встроенного типа `int`
6. bitwise_xor — проверяет корректность оператора побитового исключающего ИЛИ в отношении другого числа (number) или встроенного типа `int`
7. bitwise_left_shift — проверяет корректность оператора побитового сдвига влево в отношении другого числа (number) или встроенного типа `int`
8. bitwise_right_shift — проверяет корректность оператора побитового сдвига вправо в отношении другого числа (number) или встроенного типа `int`
9. unary_minus — проверяет корректность унарного оператора минус
10. addition — проверяет корректность оператора сложения в отношении другого числа (number) или встроенного типа `int`
11. subtraction — проверяет корректность оператора вычитания в отношении другого числа (number) или встроенного типа `int`
12. multiplication — проверяет корректность оператора умножения в отношении другого числа (number) или встроенного типа `int`

13. division — проверяет корректность оператора деления в отношении другого числа (number) или встроенного типа `int`
14. modulo — проверяет корректность оператора взятия остатка от деления в отношении другого числа (number) или встроенного типа `int`
15. increment — проверяет коррентность инкрементов
16. decrement — проверяет коррентность декрементов

4.3 Прецеденты для класса adam

1. adam_year_conversion — проверяет коррентность преобразований из/в формат представления «год от Адама»
2. christ_year_conversion — проверяет коррентность преобразований из/в формат представления «год от рождества Христова»
3. addition — проверяет коррентность сложения года с числом или другим годом
4. subtraction — проверяет коррентность вычитания из года числа или другого года
5. increment — проверяет коррентность инкрементов
6. decrement — проверяет коррентность декрементов

4.4 Прецеденты для класса matrix

1. string_conversion — проверяет коррентность преобразования из/в строковый (десятичный) формат представления `std::string`
2. identity — проверяет корректность получения единичной матрицы
3. determinant — проверяет корректность вычисления определителя матрицы
4. transpose — проверяет корректность получения транспонированной матрицы
5. inverse — проверяет корректность получения обратной матрицы
6. unary_minus — проверяет корректность унарного оператора минус
7. addition — проверяет коррентность сложения матриц
8. subtraction — проверяет коррентность вычитания матриц
9. matrix_multiplication — проверяет коррентность умножения матриц
10. number_multiplication — проверяет коррентность умножения матрицы на число

Приложение А

Файл date.h

```
#ifndef DATE_H
#define DATE_H

#include <iostream>
#include <stdexcept>
#include <string>

class date {
private:
    int year, month, day;
    int hour, minute, second;

    static int leap_years_until(int);
    static int leap_years_between(int, int);
    static bool is_leap_year(int);

    static int number_of_days(int, int);

public:
    date(uint64_t = 0);
    date(const date&);
    ~date();

    static date from_unix_time(uint64_t);
    static date from_file_time(uint64_t);
    static date copy(const date&);

    uint64_t to_unix_time() const;
    uint64_t to_file_time() const;
    std::string to_string() const;

    int get_year() const;
    int get_month() const;
    int get_day() const;
    int get_hour() const;
    int get_minute() const;
    int get_second() const;

    void set_year(int);
    void set_month(int);
    void set_day(int);
    void set_hour(int);
    void set_minute(int);
    void set_second(int);

    date operator+(const date&) const;
    date operator-(const date&) const;
```

```
    date operator+(uint64_t) const;
    date operator-(uint64_t) const;

    friend date operator+(uint64_t, const date&);
    friend date operator-(uint64_t, const date&);

    date& operator=(const date&);
    date& operator+=(const date&);
    date& operator-=(const date&);

    date& operator=(uint64_t);
    date& operator+=(uint64_t);
    date& operator-=(uint64_t);

    date& operator++();
    date operator++(int);
    date& operator--();
    date operator--(int);

    bool operator==(const date&) const;
    bool operator!=(const date&) const;
    bool operator<(const date&) const;
    bool operator>(const date&) const;
    bool operator<=(const date&) const;
    bool operator>=(const date&) const;

    bool operator==(uint64_t) const;
    bool operator!=(uint64_t) const;
    bool operator<(uint64_t) const;
    bool operator>(uint64_t) const;
    bool operator<=(uint64_t) const;
    bool operator>=(uint64_t) const;

    friend bool operator==(uint64_t, const date&);
    friend bool operator!=(uint64_t, const date&);
    friend bool operator<(uint64_t, const date&);
    friend bool operator>(uint64_t, const date&);
    friend bool operator<=(uint64_t, const date&);
    friend bool operator>=(uint64_t, const date&);

    friend std::istream& operator>>(std::istream&, date&);
    friend std::ostream& operator<<(std::ostream&, const date&);
};

#endif
```

Файл date.cpp

```
#include "date.h"

int date::leap_years_until(int year)
{
    return year / 4 - year / 100 + year / 400;
}

int date::leap_years_between(int start, int end)
{
    return leap_years_until(end) - leap_years_until(start - 1);
}

bool date::is_leap_year(int year)
{
    return leap_years_between(year, year) == 1;
}

int date::number_of_days(int month, int year)
{
    int number_of_days = 30;

    if (month == 1 || month == 3 || month == 5 || month == 7 || month == 8 || month == 1
        number_of_days = 31;
    }

    if (month == 2) {
        if (is_leap_year(year)) {
            number_of_days = 29;
        } else {
            number_of_days = 28;
        }
    }

    return number_of_days;
}

date::date(uint64_t other)
{
    uint64_t years_since_epoch = other / 31557600;

    year = 1970 + years_since_epoch;

    if (year > 9999) {
        throw std::invalid_argument("year out of range");
    }

    uint64_t days_since_year = (other - years_since_epoch * 31536000 - leap_years_betwee

    for (int m = 1; m <= 12; m++) {
```

```
        if (days_since_year > number_of_days(m, year)) {
            days_since_year -= number_of_days(m, year);
        } else {
            month = m;
            day = days_since_year + 1;
            break;
        }
    }

    uint64_t seconds_since_day = other % 86400;

    hour = seconds_since_day / 3600;

    seconds_since_day -= hour * 3600;

    minute = seconds_since_day / 60;

    seconds_since_day -= minute * 60;

    second = seconds_since_day;
}

date::date(const date& other)
{
    year = other.year;
    month = other.month;
    day = other.day;
    hour = other.hour;
    minute = other.minute;
    second = other.second;
}

date::~date()
{
}

date date::from_unix_time(uint64_t other)
{
    date date(other);

    return date;
}

date date::from_file_time(uint64_t other)
{
    other /= 10000000;

    date date;

    uint64_t years_since_epoch = other / 31557600;
```

```
    date.year = 1601 + years_since_epoch;

    if (date.year > 9999) {
        throw std::invalid_argument("year out of range");
    }

    uint64_t days_since_year = (other - years_since_epoch * 31536000 - leap_years_betwee

    for (int m = 1; m <= 12; m++) {
        if (days_since_year > number_of_days(m, date.year)) {
            days_since_year -= number_of_days(m, date.year);
        } else {
            date.month = m;
            date.day = days_since_year + 1;
            break;
        }
    }

    uint64_t seconds_since_day = other % 86400;

    date.hour = seconds_since_day / 3600;

    seconds_since_day -= date.hour * 3600;

    date.minute = seconds_since_day / 60;

    seconds_since_day -= date.minute * 60;

    date.second = seconds_since_day;

    return date;
}

date date::copy(const date& other)
{
    date date(other);

    return date;
}

uint64_t date::to_unix_time() const
{
    if (year < 1970) {
        return 0;
    }

    uint64_t unix_time = (year - 1970) * 31536000 + leap_years_between(1970, year - 1) *

    for (int m = 1; m < month; m++) {
```

```
        unix_time += number_of_days(m, year) * 86400;
    }

    unix_time += (day - 1) * 86400;
    unix_time += hour * 3600;
    unix_time += minute * 60;
    unix_time += second;

    return unix_time;
}

uint64_t date::to_file_time() const
{
    if (year < 1601) {
        return 0;
    }

    uint64_t file_time = ((uint64_t)year - 1601) * 31536000 + leap_years_between(1601, y

    for (int m = 1; m < month; m++) {
        file_time += number_of_days(m, year) * 86400;
    }

    file_time += (day - 1) * 86400;
    file_time += hour * 3600;
    file_time += minute * 60;
    file_time += second;

    return file_time * 10000000;
}

std::string date::to_string() const
{
    std::string string, temp;

    temp = std::to_string(year);
    string.append(4 - temp.length(), '0');
    string += temp;

    string += "-";

    temp = std::to_string(month);
    string.append(2 - temp.length(), '0');
    string += temp;

    string += "-";

    temp = std::to_string(day);
    string.append(2 - temp.length(), '0');
    string += temp;
```

```
    string += "T";

    temp = std::to_string(hour);
    string.append(2 - temp.length(), '0');
    string += temp;

    string += ":";

    temp = std::to_string(minute);
    string.append(2 - temp.length(), '0');
    string += temp;

    string += ":";

    temp = std::to_string(second);
    string.append(2 - temp.length(), '0');
    string += temp;

    return string;
}

int date::get_year() const
{
    return year;
}

int date::get_month() const
{
    return month;
}

int date::get_day() const
{
    return day;
}

int date::get_hour() const
{
    return hour;
}

int date::get_minute() const
{
    return minute;
}

int date::get_second() const
{
    return second;
```

```
}

void date::set_year(int year)
{
    if (year < 1 || year > 9999) {
        throw std::invalid_argument("year out of range");
    }

    if (day > number_of_days(month, year)) {
        throw std::invalid_argument("day out of range");
    }

    this->year = year;
}

void date::set_month(int month)
{
    if (month < 1 || month > 12) {
        throw std::invalid_argument("month out of range");
    }

    if (day > number_of_days(month, year)) {
        throw std::invalid_argument("day out of range");
    }

    this->month = month;
}

void date::set_day(int day)
{
    if (day < 1 || day > number_of_days(month, year)) {
        throw std::invalid_argument("day out of range");
    }

    this->day = day;
}

void date::set_hour(int hour)
{
    if (hour < 0 || hour > 23) {
        throw std::invalid_argument("hour out of range");
    }

    this->hour = hour;
}

void date::set_minute(int minute)
{
    if (minute < 0 || minute > 59) {
        throw std::invalid_argument("minute out of range");
```

```
    }

    this->minute = minute;
}

void date::set_second(int second)
{
    if (second < 0 || second > 59) {
        throw std::invalid_argument("second out of range");
    }

    this->second = second;
}

date date::operator+(const date& other) const
{
    uint64_t unix_time = to_unix_time() + other.to_unix_time();

    return from_unix_time(unix_time);
}

date date::operator-(const date& other) const
{
    uint64_t unix_time = to_unix_time() - other.to_unix_time();

    return from_unix_time(unix_time);
}

date date::operator+(uint64_t other) const
{
    return *this + from_unix_time(other);
}

date date::operator-(uint64_t other) const
{
    return *this - from_unix_time(other);
}

date operator+(uint64_t left, const date& right)
{
    return date::from_unix_time(left) + right;
}

date operator-(uint64_t left, const date& right)
{
    return date::from_unix_time(left) - right;
}

date& date::operator=(const date& other)
{
```

```
    year = other.year;
    month = other.month;
    day = other.day;
    hour = other.hour;
    minute = other.minute;
    second = other.second;

    return *this;
}

date& date::operator+=(const date& other)
{
    return *this = *this + other;
}

date& date::operator-=(const date& other)
{
    return *this = *this - other;
}

date& date::operator=(uint64_t other)
{
    return *this = from_unix_time(other);
}

date& date::operator+=(uint64_t other)
{
    return *this = *this + other;
}

date& date::operator-=(uint64_t other)
{
    return *this = *this - other;
}

date& date::operator++()
{
    return *this += 1;
}

date date::operator++(int)
{
    date date(*this);

    *this += 1;

    return date;
}

date& date::operator--()
```

```
{
    return *this -= 1;
}

date date::operator--(int)
{
    date date(*this);

    *this -= 1;

    return date;
}

bool date::operator==(const date& other) const
{
    return to_unix_time() == other.to_unix_time();
}

bool date::operator!=(const date& other) const
{
    return to_unix_time() != other.to_unix_time();
}

bool date::operator<(const date& other) const
{
    return to_unix_time() < other.to_unix_time();
}

bool date::operator>(const date& other) const
{
    return to_unix_time() > other.to_unix_time();
}

bool date::operator<=(const date& other) const
{
    return *this < other || *this == other;
}

bool date::operator>=(const date& other) const
{
    return *this > other || *this == other;
}

bool date::operator==(uint64_t other) const
{
    return *this == date::from_unix_time(other);
}

bool date::operator!=(uint64_t other) const
{
```

```
    return *this != date::from_unix_time(other);
}

bool date::operator<(uint64_t other) const
{
    return *this < date::from_unix_time(other);
}

bool date::operator>(uint64_t other) const
{
    return *this > date::from_unix_time(other);
}

bool date::operator<=(uint64_t other) const
{
    return *this <= date::from_unix_time(other);
}

bool date::operator>=(uint64_t other) const
{
    return *this >= date::from_unix_time(other);
}

bool operator==(uint64_t left, const date& right)
{
    return date::from_unix_time(left) == right;
}

bool operator!=(uint64_t left, const date& right)
{
    return date::from_unix_time(left) != right;
}

bool operator<(uint64_t left, const date& right)
{
    return date::from_unix_time(left) < right;
}

bool operator>(uint64_t left, const date& right)
{
    return date::from_unix_time(left) > right;
}

bool operator<=(uint64_t left, const date& right)
{
    return date::from_unix_time(left) <= right;
}

bool operator>=(uint64_t left, const date& right)
{
```

```
    return date::from_unix_time(left) >= right;
}

std::istream& operator>>(std::istream& stream, date& date)
{
    uint64_t unix_time;
    stream >> unix_time;

    date = date::from_unix_time(unix_time);

    return stream;
}

std::ostream& operator<<(std::ostream& stream, const date& date)
{
    return stream << date.to_unix_time();
}
```

Файл number.h

```
#ifndef NUMBER_H
#define NUMBER_H

#include <algorithm>
#include <cmath>
#include <iostream>
#include <stdexcept>
#include <string>

#define DEFAULT_CAPACITY 8 * sizeof(int)

class number {
private:
    size_t capacity;
    bool* binary;

    void resize(size_t);

    static number copy(const number&, size_t);
    static number shift(const number&, int);
    static number divide(const number&, const number&, number&);

    static std::string divide(const std::string&, number&);
    static std::string add(const std::string&, const std::string&);

public:
    number(int = 0);
    number(const number&);
    ~number();

    static number from_int(int);
    static number copy(const number&);

    int to_int() const;
    std::string to_string() const;

    size_t get_capacity() const;

    number operator-() const;
    number operator~() const;
    number operator+(const number&) const;
    number operator-(const number&) const;
    number operator*(const number&) const;
    number operator/(const number&) const;
    number operator%(const number&) const;
    number operator&(const number&) const;
    number operator|(const number&) const;
    number operator^(const number&) const;
    number operator<<(const number&) const;
```

```
number operator>>(const number&) const;

number operator+(int) const;
number operator-(int) const;
number operator*(int) const;
number operator/(int) const;
number operator%(int) const;
number operator&(int) const;
number operator|(int) const;
number operator^(int) const;
number operator<<(int) const;
number operator>>(int) const;

friend number operator+(int, const number&);
friend number operator-(int, const number&);
friend number operator*(int, const number&);
friend number operator/(int, const number&);
friend number operator%(int, const number&);
friend number operator&(int, const number&);
friend number operator|(int, const number&);
friend number operator^(int, const number&);
friend number operator<<(int, const number&);
friend number operator>>(int, const number&);

number& operator=(const number&);
number& operator+=(const number&);
number& operator-=(const number&);
number& operator*=(const number&);
number& operator/=(const number&);
number& operator%=(const number&);
number& operator&=(const number&);
number& operator|=(const number&);
number& operator^=(const number&);
number& operator<<=(const number&);
number& operator>>=(const number&);

number& operator=(int);
number& operator+=(int);
number& operator-=(int);
number& operator*=(int);
number& operator/=(int);
number& operator%=(int);
number& operator&=(int);
number& operator|=(int);
number& operator^=(int);
number& operator<<=(int);
number& operator>>=(int);

number& operator++();
number operator++(int);
```

```
    number& operator--();
    number operator--(int);

    bool operator!() const;
    bool operator&&(const number&) const;
    bool operator||(const number&) const;
    bool operator==(const number&) const;
    bool operator!=(const number&) const;
    bool operator<(const number&) const;
    bool operator>(const number&) const;
    bool operator<=(const number&) const;
    bool operator>=(const number&) const;

    bool operator&&(int) const;
    bool operator||(int) const;
    bool operator==(int) const;
    bool operator!=(int) const;
    bool operator<(int) const;
    bool operator>(int) const;
    bool operator<=(int) const;
    bool operator>=(int) const;

    friend bool operator&&(int, const number&);
    friend bool operator||(int, const number&);
    friend bool operator==(int, const number&);
    friend bool operator!=(int, const number&);
    friend bool operator<(int, const number&);
    friend bool operator>(int, const number&);
    friend bool operator<=(int, const number&);
    friend bool operator>=(int, const number&);

    friend std::istream& operator>>(std::istream&, number&);
    friend std::ostream& operator<<(std::ostream&, const number&);
};

#endif
```

Файл number.cpp

```
#include "number.h"

void number::resize(size_t capacity)
{
    bool* new_binary = new bool[capacity];

    size_t min = std::min(this->capacity, capacity);

    for (size_t i = 0; i < min - 1; i++) {
        new_binary[i] = binary[i];
    }

    for (size_t i = min - 1; i < capacity; i++) {
        new_binary[i] = binary[this->capacity - 1];
    }

    this->capacity = capacity;

    binary = new bool[capacity];
    for (size_t i = 0; i < capacity; i++) {
        binary[i] = new_binary[i];
    }

    delete[] new_binary;
}

number number::copy(const number& other, size_t capacity)
{
    number number(other);
    number.resize(capacity);

    return number;
}

number number::shift(const number& left, int right)
{
    number number(left);

    for (size_t i = 0; i + right < left.capacity; i++) {
        number.binary[i] = left.binary[i + right];
    }

    for (size_t i = left.capacity - right; i < left.capacity; i++) {
        number.binary[i] = 0;
    }

    return number;
}
```

```
number number::divide(const number& left, const number& right, number& remainder)
{
    if (right == 0) {
        throw std::invalid_argument("cannot divide by zero");
    }

    if (left.capacity < right.capacity) {
        return divide(copy(left, right.capacity), right, remainder);
    }

    if (left.capacity > right.capacity) {
        return divide(left, copy(right, left.capacity), remainder);
    }

    number quotient(left);
    for (size_t i = 0; i < left.capacity; i++) {
        quotient.binary[i] = 0;
    }

    number dividend = left < 0 ? -left : left;
    number divisor = right < 0 ? -right : right;

    if (dividend < divisor) {
        return 0;
    }

    size_t divisor_bit = 0;
    for (size_t i = divisor.capacity - 1; i > 0; i--) {
        if (divisor.binary[i]) {
            divisor_bit = i;
            break;
        }
    }

    size_t space = left.capacity - divisor_bit - 2;
    divisor <<= space;

    for (size_t i = 0; i <= space; i++, divisor = shift(divisor, 1)) {
        if (divisor <= dividend) {
            quotient.binary[space - i] = 1;
            dividend -= divisor;
        }
    }

    remainder = dividend;

    if ((left < 0) != (right < 0)) {
        return -quotient;
    }
```

```
    return quotient;
}

number::number(int other)
{
    capacity = DEFAULT_CAPACITY;

    binary = new bool[capacity];
    for (size_t i = 0; i < capacity; i++) {
        binary[i] = other & (1 << i);
    }
}

std::string number::divide(const std::string& left, number& remainder)
{
    size_t carry = 0;
    std::string string = left;

    if (left[0] == '1') {
        carry = 1;
        string.assign(left, 1, left.length() - 1);
    }

    for (size_t i = 0; i < string.length(); i++) {
        size_t n = ((string[i] - '0') + (10 * carry));
        string[i] = n / 2 + '0';
        carry = n % 2;
    }

    remainder = carry > 0;

    return string;
}

number::number(const number& other)
{
    capacity = other.capacity;

    binary = new bool[capacity];
    for (size_t i = 0; i < capacity; i++) {
        binary[i] = other.binary[i];
    }
}

number::~number()
{
    delete[] binary;
}

number number::from_int(int other)
```

```
{
    number number(other);

    return number;
}

number number::copy(const number& other)
{
    number number(other);

    return number;
}

int number::to_int() const
{
    int integer = 0;

    number narrowed = copy(*this, DEFAULT_CAPACITY);

    for (size_t i = 0; i < DEFAULT_CAPACITY; i++) {
        integer |= narrowed.binary[i] << i;
    }

    return integer;
}

std::string number::add(const std::string& left, const std::string& right)
{
    if (left.length() < right.length()) {
        return add(right, left);
    }

    if (left.length() > right.length()) {
        std::string resized = right;
        resized.append(left.length() - right.length(), '0');
        return add(left, resized);
    }

    std::string string(left.length(), '0');

    size_t carry = 0;
    for (size_t i = 0; i < left.length(); i++) {
        size_t n = (left[i] - '0') + (right[i] - '0') + (carry > 0);
        string[i] = n % 10 + '0';

        if (n / 10) {
            carry += carry == 0;
        } else {
            carry -= carry > 0;
        }
```

```
    }

    if (carry > 0) {
        string += "1";
    }

    return string;
}

std::string number::to_string() const
{
    number number = *this < 0 ? -*this : *this;

    std::string string = "0", power = "1";
    for (size_t i = 0; i < capacity - 1; i++) {
        if (number.binary[i]) {
            string = add(string, power);
        }

        power = add(power, power);
    }

    if (*this < 0) {
        string += "-";
    }

    std::reverse(string.begin(), string.end());

    return string;
}

size_t number::get_capacity() const
{
    return capacity;
}

number number::operator-() const
{
    return ~copy(*this) + 1;
}

number number::operator~() const
{
    number number(*this);

    for (size_t i = 0; i < capacity; i++) {
        number.binary[i] = !binary[i];
    }

    return number;
```

```
}

number number::operator+(const number& other) const
{
    if (capacity < other.capacity) {
        return copy(*this, other.capacity) + other;
    }

    if (capacity > other.capacity) {
        return *this + copy(other, capacity);
    }

    number sum(*this);

    bool carry = 0;
    for (size_t i = 0; i < capacity; i++) {
        sum.binary[i] = (binary[i] != other.binary[i]) != carry;
        carry = ((binary[i] || other.binary[i]) && carry) || (binary[i] && other.binary[
    }

    return sum;
}

number number::operator-(const number& other) const
{
    return *this + -other;
}

number number::operator*(const number& other) const
{
    if (capacity < other.capacity) {
        return copy(*this, other.capacity) * other;
    }

    if (capacity > other.capacity) {
        return *this * copy(other, capacity);
    }

    number product(*this);
    for (size_t i = 0; i < capacity; i++) {
        product.binary[i] = 0;
    }

    for (size_t i = 0; i < capacity; i++) {
        if (other.binary[i]) {
            product += *this << i;
        }
    }

    return product;
```

```
}

number number::operator/(const number& other) const
{
    number remainder;

    return divide(*this, other, remainder);
}

number number::operator%(const number& other) const
{
    number remainder;

    divide(*this, other, remainder);

    return remainder;
}

number number::operator&(const number& other) const
{
    if (capacity < other.capacity) {
        return copy(*this, other.capacity) & other;
    }

    if (capacity > other.capacity) {
        return *this & copy(other, capacity);
    }

    number number(*this);

    for (size_t i = 0; i < capacity; i++) {
        number.binary[i] = binary[i] && other.binary[i];
    }

    return number;
}

number number::operator|(const number& other) const
{
    if (capacity < other.capacity) {
        return copy(*this, other.capacity) | other;
    }

    if (capacity > other.capacity) {
        return *this | copy(other, capacity);
    }

    number number(*this);

    for (size_t i = 0; i < capacity; i++) {
```

```
        number.binary[i] = binary[i] || other.binary[i];
    }

    return number;
}

number number::operator^(const number& other) const
{
    if (capacity < other.capacity) {
        return copy(*this, other.capacity) ^ other;
    }

    if (capacity > other.capacity) {
        return *this ^ copy(other, capacity);
    }

    number number(*this);

    for (size_t i = 0; i < capacity; i++) {
        number.binary[i] = binary[i] != other.binary[i];
    }

    return number;
}

number number::operator<<(const number& other) const
{
    return *this << other.to_int();
}

number number::operator>>(const number& other) const
{
    return *this >> other.to_int();
}

number number::operator+(int other) const
{
    return *this + from_int(other);
}

number number::operator-(int other) const
{
    return *this - from_int(other);
}

number number::operator*(int other) const
{
    return *this * from_int(other);
}
```

```
number number::operator/(int other) const
{
    return *this / from_int(other);
}

number number::operator%(int other) const
{
    return *this % from_int(other);
}

number number::operator&(int other) const
{
    return *this & from_int(other);
}

number number::operator|(int other) const
{
    return *this | from_int(other);
}

number number::operator^(int other) const
{
    return *this ^ from_int(other);
}

number number::operator<<(int other) const
{
    number number(*this);

    for (size_t i = 0; i < other; i++) {
        number.binary[i] = 0;
    }

    for (size_t i = other; i < capacity; i++) {
        number.binary[i] = binary[i - other];
    }

    return number;
}

number number::operator>>(int other) const
{
    number number(*this);

    for (size_t i = 0; i + other < capacity; i++) {
        number.binary[i] = binary[i + other];
    }

    for (size_t i = capacity - other; i < capacity; i++) {
        number.binary[i] = binary[capacity - 1];
```

```
    }

    return number;
}

number operator+(int left, const number& right)
{
    return number::from_int(left) + right;
}

number operator-(int left, const number& right)
{
    return number::from_int(left) - right;
}

number operator*(int left, const number& right)
{
    return number::from_int(left) * right;
}

number operator/(int left, const number& right)
{
    return number::from_int(left) / right;
}

number operator%(int left, const number& right)
{
    return number::from_int(left) % right;
}

number operator&(int left, const number& right)
{
    return number::from_int(left) & right;
}

number operator|(int left, const number& right)
{
    return number::from_int(left) | right;
}

number operator^(int left, const number& right)
{
    return number::from_int(left) ^ right;
}

number operator<<(int left, const number& right)
{
    return number::from_int(left) << right;
}
```

```
number operator>>(int left, const number& right)
{
    return number::from_int(left) >> right;
}

number& number::operator=(const number& other)
{
    capacity = other.capacity;

    binary = new bool[capacity];
    for (size_t i = 0; i < capacity; i++) {
        binary[i] = other.binary[i];
    }

    return *this;
}

number& number::operator+=(const number& other)
{
    return *this = *this + other;
}

number& number::operator-=(const number& other)
{
    return *this = *this - other;
}

number& number::operator*=(const number& other)
{
    return *this = *this * other;
}

number& number::operator/=(const number& other)
{
    return *this = *this / other;
}

number& number::operator%=(const number& other)
{
    return *this = *this % other;
}

number& number::operator&=(const number& other)
{
    return *this = *this & other;
}

number& number::operator|=(const number& other)
{
    return *this = *this | other;
```

```
}

number& number::operator^=(const number& other)
{
    return *this = *this ^ other;
}

number& number::operator<<=(const number& other)
{
    return *this = *this << other;
}

number& number::operator>>=(const number& other)
{
    return *this = *this >> other;
}

number& number::operator=(int other)
{
    return *this = from_int(other);
}

number& number::operator+=(int other)
{
    return *this = *this + other;
}

number& number::operator-=(int other)
{
    return *this = *this - other;
}

number& number::operator*=(int other)
{
    return *this = *this * other;
}

number& number::operator/=(int other)
{
    return *this = *this / other;
}

number& number::operator%=(int other)
{
    return *this = *this % other;
}

number& number::operator&=(int other)
{
    return *this = *this & other;
```

```
}

number& number::operator|=(int other)
{
    return *this = *this | other;
}

number& number::operator^=(int other)
{
    return *this = *this ^ other;
}

number& number::operator<<=(int other)
{
    return *this = *this << other;
}

number& number::operator>>=(int other)
{
    return *this = *this >> other;
}

number& number::operator++()
{
    return *this += 1;
}

number number::operator++(int)
{
    number number(*this);

    *this += 1;

    return number;
}

number& number::operator--()
{
    return *this -= 1;
}

number number::operator--(int)
{
    number number(*this);

    *this -= 1;

    return number;
}
```

```
bool number::operator!() const
{
    return *this == 0;
}

bool number::operator&&(const number& other) const
{
    return *this != 0 && other != 0;
}

bool number::operator||(const number& other) const
{
    return *this != 0 || other != 0;
}

bool number::operator==(const number& other) const
{
    if (capacity < other.capacity) {
        return copy(*this, other.capacity) == other;
    }

    if (capacity > other.capacity) {
        return *this == copy(other, capacity);
    }

    for (size_t i = 0; i < capacity; i++) {
        if (binary[i] != other.binary[i]) {
            return false;
        }
    }

    return true;
}

bool number::operator!=(const number& other) const
{
    return !(*this == other);
}

bool number::operator<(const number& other) const
{
    if (capacity < other.capacity) {
        return copy(*this, other.capacity) < other;
    }

    if (capacity > other.capacity) {
        return *this < copy(other, capacity);
    }

    if (binary[capacity - 1] && !other.binary[other.capacity - 1]) {
```

```
        return true;
    }

    if (other.binary[other.capacity - 1] && !binary[capacity - 1]) {
        return false;
    }

    for (size_t i = 0; i < capacity; i++) {
        if (binary[capacity - i - 1] < other.binary[capacity - i - 1]) {
            return true;
        }

        if (binary[capacity - i - 1] > other.binary[capacity - i - 1]) {
            return false;
        }
    }

    return false;
}

bool number::operator>(const number& other) const
{
    return other < *this;
}

bool number::operator<=(const number& other) const
{
    return *this < other || *this == other;
}

bool number::operator>=(const number& other) const
{
    return *this > other || *this == other;
}

bool number::operator&&(int other) const
{
    return *this && number::from_int(other);
}

bool number::operator||(int other) const
{
    return *this || number::from_int(other);
}

bool number::operator==(int other) const
{
    return *this == number::from_int(other);
}
```

```
bool number::operator!=(int other) const
{
    return *this != number::from_int(other);
}

bool number::operator<(int other) const
{
    return *this < number::from_int(other);
}

bool number::operator>(int other) const
{
    return *this > number::from_int(other);
}

bool number::operator<=(int other) const
{
    return *this <= number::from_int(other);
}

bool number::operator>=(int other) const
{
    return *this >= number::from_int(other);
}

bool operator&&(int left, const number& right)
{
    return number::from_int(left) && right;
}

bool operator||(int left, const number& right)
{
    return number::from_int(left) || right;
}

bool operator==(int left, const number& right)
{
    return number::from_int(left) == right;
}

bool operator!=(int left, const number& right)
{
    return number::from_int(left) != right;
}

bool operator<(int left, const number& right)
{
    return number::from_int(left) < right;
}
```

```
bool operator>(int left, const number& right)
{
    return number::from_int(left) > right;
}

bool operator<=(int left, const number& right)
{
    return number::from_int(left) <= right;
}

bool operator>=(int left, const number& right)
{
    return number::from_int(left) >= right;
}

std::istream& operator>>(std::istream& stream, number& num)
{
    std::string string;
    stream >> string;

    num = number::from_int(0);

    std::string temp = string;

    if (string[0] == '-') {
        temp.assign(string, 1, string.length() - 1);
    }

    number remainder;
    for (size_t i = 0; !temp.empty(); i++) {
        if (i == num.capacity - 1) {
            num.resize(2 * num.capacity);
        }

        temp = number::divide(temp, remainder);

        num.binary[i] = remainder == 1;
    }

    if (string[0] == '-') {
        num = -num;
    }

    return stream;
}

std::ostream& operator<<(std::ostream& stream, const number& num)
{
    return stream << num.to_string();
}
```

Файл adam.h

```
#ifndef ADAM_H
#define ADAM_H

#include <iostream>
#include <stdexcept>

#define MAX_ADAM_YEAR 7980
#define CHRIST_YEAR_DEVIATION 5508

class adam {
private:
    int indict, sun, moon;

public:
    adam(int = 1);
    adam(int, int, int);
    adam(const adam&);
    ~adam();

    static adam from_adam_year(int = 1);
    static adam from_christ_year(int = 1);
    static adam from_cycle(int, int, int);
    static adam copy(const adam&);

    int to_adam_year() const;
    int to_christ_year() const;

    int get_indict() const;
    int get_sun() const;
    int get_moon() const;

    void set_indict(int);
    void set_sun(int);
    void set_moon(int);

    adam operator+(const adam&) const;
    adam operator-(const adam&) const;

    adam operator+(int) const;
    adam operator-(int) const;

    friend adam operator+(int, const adam&);
    friend adam operator-(int, const adam&);

    adam& operator=(const adam&);
    adam& operator+=(const adam&);
    adam& operator-=(const adam&);

    adam& operator=(int);
```

```
    adam& operator+=(int);
    adam& operator-=(int);

    adam& operator++();
    adam operator++(int);
    adam& operator--();
    adam operator--(int);

    bool operator==(const adam&) const;
    bool operator!=(const adam&) const;
    bool operator<(const adam&) const;
    bool operator>(const adam&) const;
    bool operator<=(const adam&) const;
    bool operator>=(const adam&) const;

    bool operator==(int) const;
    bool operator!=(int) const;
    bool operator<(int) const;
    bool operator>(int) const;
    bool operator<=(int) const;
    bool operator>=(int) const;

    friend bool operator==(int, const adam&);
    friend bool operator!=(int, const adam&);
    friend bool operator<(int, const adam&);
    friend bool operator>(int, const adam&);
    friend bool operator<=(int, const adam&);
    friend bool operator>=(int, const adam&);

    friend std::istream& operator>>(std::istream&, adam&);
    friend std::ostream& operator<<(std::ostream&, const adam&);
};

#endif
```

Файл adam.cpp

```
#include "adam.h"

adam::adam(int other)
{
    if (other < 1 || other > MAX_ADAM_YEAR) {
        throw std::invalid_argument("year out of range");
    }

    indict = other % 15;
    sun = other % 28;
    moon = other % 19;
}

adam::adam(int indict, int sun, int moon)
{
    set_indict(indict);
    set_sun(sun);
    set_moon(moon);
}

adam::adam(const adam& other)
{
    indict = other.indict;
    sun = other.sun;
    moon = other.moon;
}

adam::~adam()
{
}

adam adam::from_adam_year(int other)
{
    adam adam(other);

    return adam;
}

adam adam::from_christ_year(int other)
{
    adam adam(other + CHRIST_YEAR_DEVIATION);

    return adam;
}

adam adam::from_cycle(int indict, int sun, int moon)
{
    adam adam(indict, sun, moon);
```

```
    return adam;
}

adam adam::copy(const adam& other)
{
    adam adam(other);

    return adam;
}

int adam::to_adam_year() const
{
    int m1 = MAX_ADAM_YEAR / 15, y1;
    int m2 = MAX_ADAM_YEAR / 28, y2;
    int m3 = MAX_ADAM_YEAR / 19, y3;

    for (int y = 0, m = m1 % 15; y < 15; y++)
        if ((m * y) % 15 == indict)
            y1 = y;
    for (int y = 0, m = m2 % 28; y < 28; y++)
        if ((m * y) % 28 == sun)
            y2 = y;
    for (int y = 0, m = m3 % 19; y < 19; y++)
        if ((m * y) % 19 == moon)
            y3 = y;

    return (m1 * y1 + m2 * y2 + m3 * y3) % MAX_ADAM_YEAR;
}

int adam::to_christ_year() const
{
    return to_adam_year() - CHRIST_YEAR_DEVIATION;
}

int adam::get_indict() const
{
    return indict + 1;
}

int adam::get_sun() const
{
    return sun + 1;
}

int adam::get_moon() const
{
    return moon + 1;
}

void adam::set_indict(int indict)
```

```
{
    if (indict < 1 || indict > 15) {
        throw std::invalid_argument("indict out of range");
    }

    this->indict = indict - 1;
}

void adam::set_sun(int sun)
{
    if (sun < 1 || sun > 28) {
        throw std::invalid_argument("sun out of range");
    }

    this->sun = sun - 1;
}

void adam::set_moon(int moon)
{
    if (moon < 1 || moon > 19) {
        throw std::invalid_argument("moon out of range");
    }

    this->moon = moon - 1;
}

adam adam::operator+(const adam& other) const
{
    int adam_year = to_adam_year() + other.to_adam_year();

    return from_adam_year(adam_year);
}

adam adam::operator-(const adam& other) const
{
    int adam_year = to_adam_year() - other.to_adam_year();

    return from_adam_year(adam_year);
}

adam adam::operator+(int other) const
{
    return *this + from_adam_year(other);
}

adam adam::operator-(int other) const
{
    return *this - from_adam_year(other);
}
```

```
adam operator+(int left, const adam& right)
{
    return adam::from_adam_year(left) + right;
}

adam operator-(int left, const adam& right)
{
    return adam::from_adam_year(left) - right;
}

adam& adam::operator=(const adam& other)
{
    indict = other.indict;
    sun = other.sun;
    moon = other.moon;

    return *this;
}

adam& adam::operator+=(const adam& other)
{
    return *this = *this + other;
}

adam& adam::operator-=(const adam& other)
{
    return *this = *this - other;
}

adam& adam::operator=(int other)
{
    return *this = from_adam_year(other);
}

adam& adam::operator+=(int other)
{
    return *this = *this + other;
}

adam& adam::operator-=(int other)
{
    return *this = *this - other;
}

adam& adam::operator++()
{
    return *this += 1;
}

adam adam::operator++(int)
```

```
{
    adam adam(*this);

    *this += 1;

    return adam;
}

adam& adam::operator--()
{
    return *this -= 1;
}

adam adam::operator--(int)
{
    adam adam(*this);

    *this -= 1;

    return adam;
}

bool adam::operator==(const adam& other) const
{
    return to_adam_year() == other.to_adam_year();
}

bool adam::operator!=(const adam& other) const
{
    return to_adam_year() != other.to_adam_year();
}

bool adam::operator<(const adam& other) const
{
    return to_adam_year() < other.to_adam_year();
}

bool adam::operator>(const adam& other) const
{
    return to_adam_year() > other.to_adam_year();
}

bool adam::operator<=(const adam& other) const
{
    return *this < other || *this == other;
}

bool adam::operator>=(const adam& other) const
{
    return *this > other || *this == other;
```

```
}

bool adam::operator==(int other) const
{
    return *this == adam::from_adam_year(other);
}

bool adam::operator!=(int other) const
{
    return *this != adam::from_adam_year(other);
}

bool adam::operator<(int other) const
{
    return *this < adam::from_adam_year(other);
}

bool adam::operator>(int other) const
{
    return *this > adam::from_adam_year(other);
}

bool adam::operator<=(int other) const
{
    return *this <= adam::from_adam_year(other);
}

bool adam::operator>=(int other) const
{
    return *this >= adam::from_adam_year(other);
}

bool operator==(int left, const adam& right)
{
    return adam::from_adam_year(left) == right;
}

bool operator!=(int left, const adam& right)
{
    return adam::from_adam_year(left) != right;
}

bool operator<(int left, const adam& right)
{
    return adam::from_adam_year(left) < right;
}

bool operator>(int left, const adam& right)
{
    return adam::from_adam_year(left) > right;
```

```
}

bool operator<=(int left, const adam& right)
{
    return adam::from_adam_year(left) <= right;
}

bool operator>=(int left, const adam& right)
{
    return adam::from_adam_year(left) >= right;
}

std::istream& operator>>(std::istream& stream, adam& adam)
{
    int adam_year;
    stream >> adam_year;

    adam = adam::from_adam_year(adam_year);

    return stream;
}

std::ostream& operator<<(std::ostream& stream, const adam& adam)
{
    return stream << adam.to_adam_year();
}
```

Файл matrix.h

```
#ifndef MATRIX_H
#define MATRIX_H

#include <algorithm>
#include <cmath>
#include <iostream>
#include <map>
#include <stdexcept>
#include <string>

class matrix {
private:
    size_t rows, columns;
    std::map<size_t, double>* adjacency;

public:
    matrix(size_t = 0, size_t = 0);
    matrix(const matrix&);
    ~matrix();

    static matrix identity(size_t);
    static matrix copy(const matrix&);
    static matrix copy(const matrix&, size_t, size_t);

    std::string to_string() const;

    size_t get_rows() const;
    size_t get_columns() const;

    double get(size_t, size_t) const;

    void set(size_t, size_t, double);

    double determinant() const;

    matrix transpose() const;
    matrix inverse() const;

    matrix operator-() const;
    matrix operator+(const matrix&) const;
    matrix operator-(const matrix&) const;
    matrix operator*(const matrix&) const;

    matrix operator*(double) const;

    friend matrix operator*(double, const matrix&);

    matrix& operator=(const matrix&);
    matrix& operator+=(const matrix&);
```

```
    matrix& operator-=(const matrix&);
    matrix& operator*=(const matrix&);

    matrix& operator*=(double);

    bool operator==(const matrix&) const;
    bool operator!=(const matrix&) const;

    friend std::istream& operator>>(std::istream&, matrix&);
    friend std::ostream& operator<<(std::ostream&, const matrix&);
};

#endif
```

Файл matrix.cpp

```
#include "matrix.h"

matrix::matrix(size_t rows, size_t columns)
{
    this->rows = rows;
    this->columns = columns;

    adjacency = new std::map<size_t, double>[rows];
}

matrix::matrix(const matrix& other)
{
    rows = other.rows;
    columns = other.columns;

    adjacency = new std::map<size_t, double>[rows];
    for (size_t r = 0; r < rows; r++) {
        adjacency[r] = other.adjacency[r];
    }
}

matrix::~matrix()
{
    delete[] adjacency;
}

matrix matrix::identity(size_t size)
{
    matrix matrix(size, size);

    for (size_t i = 0; i < size; i++) {
        matrix.set(i, i, 1.);
    }

    return matrix;
}

matrix matrix::copy(const matrix& other)
{
    matrix matrix(other);

    return matrix;
}

matrix matrix::copy(const matrix& other, size_t row, size_t column)
{
    if (row >= other.rows) {
        throw std::invalid_argument("row out of range");
    }
```

```
    if (column >= other.columns) {
        throw std::invalid_argument("column out of range");
    }

    matrix matrix(other.rows - 1, other.columns - 1);

    for (size_t r = 0; r < row; r++) {
        for (std::pair<size_t, double> e : other.adjacency[r]) {
            if (e.first != column) {
                matrix.set(r, e.first - (e.first > column), e.second);
            }
        }
    }

    for (size_t r = row + 1; r < other.rows; r++) {
        for (std::pair<size_t, double> e : other.adjacency[r]) {
            if (e.first != column) {
                matrix.set(r - 1, e.first - (e.first > column), e.second);
            }
        }
    }

    return matrix;
}

std::string matrix::to_string() const
{
    std::string string = std::to_string(rows) + " " + std::to_string(columns);

    for (size_t r = 0; r < rows; r++) {
        string += "\n";
        for (size_t c = 0; c < columns; c++) {
            string += std::to_string(get(r, c)) + " ";
        }
    }

    return string;
}

size_t matrix::get_rows() const
{
    return rows;
}

size_t matrix::get_columns() const
{
    return columns;
}
```

```
double matrix::get(size_t row, size_t column) const
{
    if (row >= rows) {
        throw std::invalid_argument("row out of range");
    }

    if (column >= columns) {
        throw std::invalid_argument("column out of range");
    }

    if (adjacency[row].count(column)) {
        return adjacency[row][column];
    }

    return .0;
}

void matrix::set(size_t row, size_t column, double value)
{
    if (row >= rows) {
        throw std::invalid_argument("row out of range");
    }

    if (column >= columns) {
        throw std::invalid_argument("column out of range");
    }

    if (value != .0) {
        adjacency[row][column] = value;
    } else if (adjacency[row].count(column)) {
        adjacency[row].erase(column);
    }
}

double matrix::determinant() const
{
    if (rows != columns) {
        throw std::logic_error("rows are not equal to columns");
    }

    if (rows == 1) {
        return get(0, 0);
    }

    double det = .0;
    for (std::pair<size_t, double> e : adjacency[0]) {
        double minor = copy(*this, 0, e.first).determinant();
        det += e.second * (e.first % 2 ? -minor : minor);
    }
```

```
    return det;
}

matrix matrix::transpose() const
{
    matrix matrix(columns, rows);

    for (size_t r = 0; r < rows; r++) {
        for (std::pair<size_t, double> e : adjacency[r]) {
            matrix.set(e.first, r, e.second);
        }
    }

    return matrix;
}

matrix matrix::inverse() const
{
    double det = determinant();

    if (det == .0) {
        throw std::logic_error("inverse matrix does not exist");
    }

    matrix matrix(rows, columns);

    for (size_t r = 0; r < rows; r++) {
        for (std::pair<size_t, double> e : adjacency[r]) {
            double minor = copy(*this, r, e.first).determinant();
            matrix.set(r, e.first, (r + e.first) % 2 ? -minor : minor);
        }
    }

    return 1. / det * matrix.transpose();
}

matrix matrix::operator-() const
{
    matrix matrix(*this);

    for (size_t r = 0; r < rows; r++) {
        for (std::pair<size_t, double> e : adjacency[r]) {
            matrix.set(r, e.first, -e.second);
        }
    }

    return matrix;
}

matrix matrix::operator+(const matrix& other) const
```

```
{
    if (rows != other.rows) {
        throw std::invalid_argument("rows are not equal");
    }

    if (columns != other.columns) {
        throw std::invalid_argument("columns are not equal");
    }

    matrix matrix(*this);

    for (size_t r = 0; r < rows; r++) {
        for (std::pair<size_t, double> e : other.adjacency[r]) {
            matrix.set(r, e.first, get(r, e.first) + e.second);
        }
    }

    return matrix;
}

matrix matrix::operator-(const matrix& other) const
{
    if (rows != other.rows) {
        throw std::invalid_argument("rows are not equal");
    }

    if (columns != other.columns) {
        throw std::invalid_argument("columns are not equal");
    }

    matrix matrix(*this);

    for (size_t r = 0; r < rows; r++) {
        for (std::pair<size_t, double> e : other.adjacency[r]) {
            matrix.set(r, e.first, get(r, e.first) - e.second);
        }
    }

    return matrix;
}

matrix matrix::operator*(const matrix& other) const
{
    if (columns != other.rows) {
        throw std::invalid_argument("columns are not equal to rows");
    }

    matrix matrix(rows, other.columns);

    for (size_t r = 0; r < rows; r++) {
```

```
        for (size_t c = 0; c < other.columns; c++) {
            double product = .0;
            for (size_t i = 0; i < columns; i++) {
                product += get(r, i) * other.get(i, c);
            }

            matrix.set(r, c, product);
        }
    }

    return matrix;
}

matrix matrix::operator*(double other) const
{
    matrix matrix(rows, columns);

    for (size_t r = 0; r < rows; r++) {
        for (std::pair<size_t, double> e : adjacency[r]) {
            matrix.set(r, e.first, e.second * other);
        }
    }

    return matrix;
}

matrix operator*(double left, const matrix& right)
{
    return right * left;
}

matrix& matrix::operator=(const matrix& other)
{
    rows = other.rows;
    columns = other.columns;

    adjacency = new std::map<size_t, double>[rows];
    for (size_t r = 0; r < rows; r++) {
        adjacency[r] = other.adjacency[r];
    }

    return *this;
}

matrix& matrix::operator+=(const matrix& other)
{
    return *this = *this + other;
}

matrix& matrix::operator-=(const matrix& other)
```

```
{
    return *this = *this - other;
}

matrix& matrix::operator*=(const matrix& other)
{
    return *this = *this * other;
}

matrix& matrix::operator*=(double other)
{
    return *this = *this * other;
}

bool matrix::operator==(const matrix& other) const
{
    if (rows != other.rows) {
        return false;
    }

    if (columns != other.columns) {
        return false;
    }

    for (size_t r = 0; r < rows; r++) {
        for (std::pair<size_t, double> e : adjacency[r]) {
            if (std::abs(e.second - other.get(r, e.first)) > 1e-6) {
                return false;
            }
        }

        for (std::pair<size_t, double> e : other.adjacency[r]) {
            if (std::abs(e.second - get(r, e.first)) > 1e-6) {
                return false;
            }
        }
    }

    return true;
}

bool matrix::operator!=(const matrix& other) const
{
    return !(*this == other);
}

std::istream& operator>>(std::istream& stream, matrix& matrix)
{
    stream >> matrix.rows;
    stream >> matrix.columns;
```

```
    matrix.adjacency = new std::map<size_t, double>[matrix.rows];
    for (size_t r = 0; r < matrix.rows; r++) {
        for (size_t c = 0; c < matrix.columns; c++) {
            double value;
            stream >> value;
            matrix.set(r, c, value);
        }
    }

    return stream;
}

std::ostream& operator<<(std::ostream& stream, const matrix& matrix)
{
    return stream << matrix.to_string();
}
```

Файл CMakeLists.txt

```
cmake_minimum_required(VERSION 3.0)

project(laboratory-work-01-kapavkin)

set(CMAKE_CXX_STANDARD 14)

include(FetchContent)
FetchContent_Declare(
  googletest
  URL https://github.com/google/googletest/archive/609281088cfefc76f9d0ce82e1ff6c30cc359
)

set(gtest_force_shared_crt ON CACHE BOOL "" FORCE)
FetchContent_MakeAvailable(googletest)

set(MAIN main.cpp)

enable_testing()

add_executable(main ${MAIN})

add_subdirectory(date)
add_subdirectory(number)
add_subdirectory(adam)
add_subdirectory(matrix)

target_link_libraries(main date)
target_link_libraries(main number)
target_link_libraries(main adam)
target_link_libraries(main matrix)
target_link_libraries(main gtest_main)

include(GoogleTest)
gtest_discover_tests(main)
```