МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

*Системы Сбора И Обработки Данных*

Кафедра \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(полное название кафедры)

**Курсовая работа**

по дисциплине: Программирование

Тема: Дата, объекты хранят число дней, прошедших с 01.01.1970

Рецензия: (краткое обоснование оценки):

Выполнил:

Студент: Дашкин С.М.

Группа: АТ-34

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_ г.

Проверил:

Преподаватель: Воронов В.В.

Балл:\_\_\_\_\_\_\_\_\_\_, ECTS\_\_\_\_\_\_\_\_\_\_\_\_,

Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г.

Новосибирск 2024

Содержание

[Техническое задание 3](#_Toc184337134)

[Описание разработанного класса 3](#_Toc184337135)

[Члены класса 4](#_Toc184337136)

[Приложения 5](#_Toc184337137)

### Техническое задание

Разработать класс, представляющий заданный тип данных (ТД). Класс должен включать в себя (если это необходимо и подходит по смыслу к ТД):

* Конструкторы (по умолчанию, с параметрами, копии);
* Деструктор;
* Функцию вывода на экран;
* Перегруженные операторы:
  + Оператор присваивания;
  + Бинарные арифметические (+, -, \*, /) с операндами ТД и ТД, ТД и базовый тип, базовый тип и ТД;
  + Арифметика с накоплением (+=,-=);
  + Унарные (++,--) в префиксной и постфиксной форме;
  + Логические (<, >, ==, != );
  + Операторы взятия элемента ( [] или () ) по номеру или ключу;
  + Операторы преобразования типа к любому базовому.

Написать головную программу, обеспечивающую проверку всех реализованных свойств и методов класса. ТД - дата, объекты хранят число дней, прошедших с 01.01.1970.

### Описание разработанного класса

Date — это класс, который моделирует дату, представленную количеством дней, прошедших с 1 января 1970 года (эпоха Unix). Он поддерживает основные арифметические операции, такие как сложение и вычитание дней, а также операции сравнения, позволяя определять, равны ли две даты или одна дата раньше или позже другой.

Класс предоставляет возможность вывода даты в формате "год-месяц-день", что делает его удобным для использования в приложениях, требующих отображения дат. Кроме того, Date поддерживает арифметику с накоплением, что позволяет легко изменять дату на определенное количество дней.

Date также поддерживает унарные операторы инкремента и декремента, что позволяет увеличивать или уменьшать дату на один день. Класс включает в себя логические операторы для сравнения дат, что делает его удобным для использования в условных конструкциях.

### Члены класса

**private:**

1. int daysSinceEpoch - целочисленное поле, хранящее количество дней, прошедших с 1 января 1970 года (эпоха Unix). Это основное поле, которое используется для представления даты.
2. void checkValidDate() const - метод, проверяющий корректность даты. Если количество дней с эпохи Unix отрицательное, метод выбрасывает исключение `std::invalid\_argument` с сообщением "Date cannot be before 01.01.1970".
3. static bool isLeapYear(int year) - статический метод, проверяющий, является ли год високосным. Возвращает `true`, если год високосный, и `false` в противном случае. Год считается високосным, если он делится на 4, но не делится на 100, либо делится на 400.
4. static int daysInMonth(int year, int month) - статический метод, возвращающий количество дней в указанном месяце указанного года. Учитывает високосные годы и возвращает 29 дней для февраля в високосный год. В остальных случаях использует статический массив `daysInMonth`, содержащий количество дней для каждого месяца.

**public:**

Конструкторы

1. Date() - создает дату, представляющую собой 1 января 1970 года (0 дней с эпохи Unix).
2. Date(int days) - принимает количество дней с эпохи Unix и создает дату, соответствующую этому количеству дней. Если количество дней отрицательное, выбрасывает исключение `std::invalid\_argument`.
3. Используется конструктор копирования по умолчанию, предоставляемый компилятором. Явное определение конструктора копирования не требуется, так как он просто копирует значение daysSinceEpoch.

Деструктор

1. Используется деструктор по умолчанию, предоставляемый компилятором, т.к. класс Date не использует динамическую память или другие ресурсы, которые требуют явного освобождения.

Функция вывода на экран

1. void print() const - функция для вывода даты на экран в формате "год-месяц-день".

Оператор присваивания

1. Используется оператор присваивания по умолчанию, предоставляемый компилятором. Явное определение конструктора копирования не требуется, так как он просто копирует значение daysSinceEpoch.

Бинарные арифметические операторы

1. Date operator+(int days) const - возвращает новую дату, представляющую сумму текущей даты и указанного количества дней.
2. Date operator-(int days) const - возвращает новую дату, представляющую разность текущей даты и указанного количества дней. Если результат отрицательный, выбрасывает исключение `std::invalid\_argument`.

Арифметика с накоплением

1. Date& operator+=(int days) - добавляет указанное количество дней к текущей дате и сохраняет результат.
2. Date& operator-=(int days) - вычитает указанное количество дней из текущей даты и сохраняет результат. Если результат отрицательный, выбрасывает исключение `std::invalid\_argument`.

Унарные операторы

1. Date& operator++() - префиксный инкремент. Увеличивает дату на один день.
2. Date operator++(int) - постфиксный инкремент. Увеличивает дату на один день и возвращает старое значение.
3. Date& operator--() - префиксный декремент. Уменьшает дату на один день. Если результат отрицательный, выбрасывает исключение `std::invalid\_argument`.
4. Date operator--(int) - постфиксный декремент. Уменьшает дату на один день и возвращает старое значение. Если результат отрицательный, выбрасывает исключение `std::invalid\_argument`.

Логические операторы

1. bool operator<(const Date& other) const - проверяет, меньше ли текущая дата, чем другая дата.
2. bool operator>(const Date& other) const - проверяет, больше ли текущая дата, чем другая дата.
3. bool operator==(const Date& other) const - проверяет, равны ли две даты.
4. bool operator!=(const Date& other) const - проверяет, не равны ли две даты.

Операторы преобразования типа к базовому

1. explicit operator int() const - преобразование даты в количество дней с эпохи Unix. Возвращает количество дней с 1 января 1970 года.

### Приложения

class Date **{**

private**:**

int daysSinceEpoch**;**

void checkValidDate**()** const **{**

**if** **(**daysSinceEpoch **<** 0**)** **{**

**throw** std**::**invalid\_argument**(**"Date cannot be before 01.01.1970"**);**

**}**

**}**

static bool isLeapYear**(**int year**)** **{**

**return** **(**year **%** 4 **==** 0 **&&** year **%** 100 **!=** 0**)** **||** **(**year **%** 400 **==** 0**);**

**}**

static int daysInMonth**(**int year**,** int month**)** **{**

static const int daysInMonth**[]** **=** **{** 31**,** 28**,** 31**,** 30**,** 31**,** 30**,** 31**,** 31**,** 30**,** 31**,** 30**,** 31 **};**

**if** **(**isLeapYear**(**year**)** **&&** month **==** 2**)** **{**

**return** 29**;**

**}**

**return** daysInMonth**[**month **-** 1**];**

**}**

public**:**

// Конструкторы

Date**()** **:** daysSinceEpoch**(**0**)** **{}**

Date**(**int days**)** **:** daysSinceEpoch**(**days**)** **{**

checkValidDate**();**

**}**

// Метод вывода на экран

void print**()** const **{**

int year **=** 1970**;**

int days **=** daysSinceEpoch**;**

**while** **(**days **>=** 365**)** **{**

**if** **(**isLeapYear**(**year**))** **{**

**if** **(**days **>=** 366**)** **{**

days **-=** 366**;**

year**++;**

**}**

**else** **{**

**break;**

**}**

**}**

**else** **{**

days **-=** 365**;**

year**++;**

**}**

**}**

int month **=** 1**;**

**while** **(**days **>=** daysInMonth**(**year**,** month**))** **{**

days **-=** daysInMonth**(**year**,** month**);**

month**++;**

**}**

std**::**cout **<<** year **<<** "-" **<<** month **<<** "-" **<<** **(**days **+** 1**)** **<<** std**::**endl**;**

**}**

// Бинарные арифметические операторы

Date **operator+(**int days**)** const **{**

**return** Date**(**daysSinceEpoch **+** days**);**

**}**

Date **operator-(**int days**)** const **{**

int newDays **=** daysSinceEpoch **-** days**;**

checkValidDate**();**

**return** Date**(**newDays**);**

**}**

// Арифметика с накоплением

Date**&** **operator+=(**int days**)** **{**

daysSinceEpoch **+=** days**;**

**return** **\*this;**

**}**

Date**&** **operator-=(**int days**)** **{**

daysSinceEpoch **-=** days**;**

checkValidDate**();**

**return** **\*this;**

**}**

// Унарные операторы

Date**&** **operator++()** **{**

**++**daysSinceEpoch**;**

**return** **\*this;**

**}**

Date **operator++(**int**)** **{**

Date temp **=** **\*this;**

**++**daysSinceEpoch**;**

**return** temp**;**

**}**

Date**&** **operator--()** **{**

**--**daysSinceEpoch**;**

checkValidDate**();**

**return** **\*this;**

**}**

Date **operator--(**int**)** **{**

Date temp **=** **\*this;**

**--**daysSinceEpoch**;**

checkValidDate**();**

**return** temp**;**

**}**

// Логические операторы

bool **operator<(**const Date**&** other**)** const **{**

**return** daysSinceEpoch **<** other**.**daysSinceEpoch**;**

**}**

bool **operator>(**const Date**&** other**)** const **{**

**return** daysSinceEpoch **>** other**.**daysSinceEpoch**;**

**}**

bool **operator==(**const Date**&** other**)** const **{**

**return** daysSinceEpoch **==** other**.**daysSinceEpoch**;**

**}**

bool **operator!=(**const Date**&** other**)** const **{**

**return** daysSinceEpoch **!=** other**.**daysSinceEpoch**;**

**}**

// Операторы преобразования типа

explicit **operator** int**()** const **{**

**return** daysSinceEpoch**;**

**}**

**};**