

# Date Class Requirements Specification

Douglas Wilhelm Harder

October 2, 2026

## 1 Purpose

The purpose of the `Date` class is to represent a valid calendar date and provide a small set of operations for creating, querying, displaying, and incrementing or decrementing dates.

This document specifies the required behavior and public interface of the class. It does not prescribe an implementation, internal representation, or algorithm.

## 2 Scope

The `Date` class shall support:

- construction of valid calendar dates,
- verification that a proposed year, month, and day form a valid date,
- access to the year, month, and day,
- formatted output of a date, and
- movement to the next or previous calendar date.

## 3 Calendar

**REQ-1.1** The class shall support positive years only, and thus, only years of the Common Era. There shall be no year zero or negative years.

**REQ-1.2** Dates up to and including October 4, 1582 shall use the Julian calendar.

**REQ-1.3** Dates beginning October 15, 1582 shall use the Gregorian calendar.

**REQ-1.4** The dates October 5 through October 14, 1582 shall not be valid dates.

**REQ-1.5** Under the Julian calendar, a year shall be a leap year if it is divisible by 4.

**REQ-1.6** Under the Gregorian calendar, a year shall be a leap year if it is divisible by 4, except that years divisible by 100 shall not be leap years unless they are also divisible by 400.

## 4 Construction

**REQ-2.1** The class shall provide a constructor of the form

```
Date( unsigned int year,  
      unsigned int month,  
      unsigned int day );
```

**REQ-2.2** The constructor shall create an object representing the specified calendar date.

**REQ-2.3** The constructor shall require the supplied year, month, and day to represent a valid date as defined by this specification.

**REQ-2.4** Once construction has completed successfully, the resulting object shall represent a valid date.

This format follows ISO 8601, the international standard for the representation of dates and times. Dates are written in decreasing order of significance: year, month, then day. Thus, October 2, 2026 is written

`2026-10-02`

rather than the day-month-year form `02-10-2026` or the month-day-year form `10-02-2026` commonly used in the United States. In addition to avoiding ambiguity between these regional conventions, this ordering naturally extends from the most significant to the least significant component when a time is included:

`2026-10-02T15:48:37`

which represents October 2, 2026 at 15:48:37 (3:48:37 p.m.). The components therefore appear in the order

year → month → day → hour → minute → second.

This ordering also has the useful property that dates written with fixed-width fields sort lexicographically in chronological order.

## 5 Date components

**REQ-4.1** The year, month, and day represented by a date shall be available through the following operations:

```
unsigned int year() const;
unsigned int month() const;
unsigned int day() const;
```

**REQ-4.2** These operations shall return the year, month, and day, respectively, represented by the object.

**REQ-4.3** These operations shall not modify the date.

## 6 Changing a date

An existing date shall not provide operations that arbitrarily replace its year, month, or day. A date may instead be changed by moving to the immediately following or immediately preceding valid calendar date.

**REQ-5.1** The class shall provide prefix and postfix increment operators:

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

Both operators shall advance the object by one calendar day.

**REQ-5.2** Prefix increment shall return the date after it has been incremented.

**REQ-5.3** Postfix increment shall return the date as it existed before it was incremented.

**REQ-5.4** The class shall provide prefix and postfix decrement operators:

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

Both operators shall move the object backward by one calendar day.

**REQ-5.5** Prefix decrement shall return the date after it has been decremented.

**REQ-5.6** Postfix decrement shall return the date as it existed before it was decremented.

**REQ-5.7** Incrementing October 4, 1582 shall produce October 15, 1582.

**REQ-5.8** Decrementing October 15, 1582 shall produce October 4, 1582.

**REQ-5.9** No operation shall leave an existing **Date** object representing an invalid date.

## 7 String representation and output

**REQ-6.1** The class shall provide

```
std::string to_string() const;
```

to return a string representation of the date.

**REQ-6.2** The string representation shall have the form

```
YYYY-MM-DD
```

with the month and day represented using two digits.

For example,

```
2026-10-02
```

shall represent October 2, 2026.

**REQ-6.3** The class shall support insertion into an output stream using

```
std::ostream &operator<<( std::ostream &out,  
                           Date const &date );
```

such that

```
std::cout << date;
```

outputs the same representation returned by `date.to_string()`.

## 8 Required public interface

The required public interface is summarized below.

```
class Date {
public:
    Date( unsigned int year,
          unsigned int month,
          unsigned int day );

    static bool is_valid( unsigned int year,
                          unsigned int month,
                          unsigned int day );

    unsigned int year() const;
    unsigned int month() const;
    unsigned int day() const;

    Date &operator++();
    Date  operator++( int );

    Date &operator--();
    Date  operator--( int );

    std::string to_string() const;

    // Internal representation is not specified.
};

std::ostream &operator<<( std::ostream &out,
                          Date const &date );
```

## 9 Examples of required behavior

The following examples illustrate required behavior. They do not prescribe how that behavior is to be implemented.

```
Date d1{ 2026, 10, 2 };
```

```
std::cout << d1;  
// 2026-10-02
```

```
++d1;
```

```
std::cout << d1;  
// 2026-10-03
```

Month and year boundaries shall be handled correctly:

```
Date d2{ 2026, 12, 31 };
```

```
++d2;
```

```
std::cout << d2;  
// 2027-01-01
```

Leap years shall be handled correctly:

```
Date d3{ 2024, 2, 28 };
```

```
++d3;
```

```
std::cout << d3;  
// 2024-02-29
```

The Julian-to-Gregorian transition shall be handled correctly:

```
Date d4{ 1582, 10, 4 };
```

```
++d4;
```

```
std::cout << d4;  
// 1582-10-15
```

```
--d4;
```

```
std::cout << d4;  
// 1582-10-04
```

## 10 Implementation

The internal representation and algorithms used to satisfy these requirements are intentionally unspecified.

A separate design document shall describe the implementation of the **Date** class.