Howard E. Hinnant
2017-10-16

iso_week

Contents

Introduction

This paper fully documents an ISO week date calendar that is fully interoperable with `date`.

Implementation

This entire library is implemented in a single header: iso_week.h and is open source using the MIT license.

Overview

This library creates field types that hold the year, week number and weekday associated with a ISO week date. For example, to specify the Saturday of the 51st week of 2015 one can say:

```
#include "iso_week.h"
#include <iostream>

int
main()
{
    using namespace iso_week::literals;
    auto iso_date = sat/51/2015;
    std::cout << iso_date << '\n';
}
```

The output of this program is:

```
2015-W51-Sat
```

In this example `iso_date` has type `iso_week::year_weeknum_weekday`, and this type does nothing but hold the three quantities: `year`, `weeknum` and `weekday`. This is such trivial functionality that it almost seems useless. Anyone can write a type to be constructed from three values, and then print them back out when requested.

The real power of `year_weeknum_weekday` is that it can implicitly convert to and from `sys_days`. And `sys_days` is just a type alias for:

```
std::chrono::time_point<std::chrono::system_clock, days>
```

This is the exact same `sys_days` used by the `date` and time zone libraries. And so by simply having conversions to and from `sys_days`, `iso_week::year_weeknum_weekday` becomes seamlessly interoperable with all of the types in `date` and time zone such as `date::year_month_day` and `date::Zone`:

```
#include "date.h"
#include "iso_week.h"
#include "tz.h"
#include <iostream>

int
main()
{
    using namespace std::chrono;
    using namespace date;
    using namespace iso_week;
    auto zone = locate_zone("America/New_York");
    auto now = zone->to_local(system_clock::now());
```

```
    auto dp = floor<days>(now.first);
    auto iso_date = year_weeknum_weekday{dp};
    auto civil_date = year_month_day{dp};
    auto time = make_time(duration_cast<minutes>(now.first-dp));
    std::cout << "It is currently " << time << ' ' << now.second << " on "
              << iso_date << " which is also " << civil_date << '\n';
}
```

Which just output for me:

```
It is currently 18:33 EST on 2015-W51-Sat which is also 2015-12-19
```

Or if you want to find the civil date of the last day of the ISO week year for 2015, then (knowning the last day of the week is Sunday in this calendar) it is:

```
using namespace iso_week::literals;
std::cout << date::year_month_day{2015_y/last/sun} << '\n';
```

Which will output:

```
2016-01-03
```

And of course one can convert in the opposite direction:

```
using namespace date::literals;
std::cout << iso_week::year_weeknum_weekday{2016_y/1/3} << '\n';
```

Which will output:

```
2015-W53-Sun
```

In particular, note that for this day, the `iso_week::year` is 2015 and the `date::year` is 2016.

This brings us to an important type-safety feature of these libraries: The `iso_week::year` and the `date::year` are two *distinct* types. They represent concepts that are *very* similar. They almost always have the same value for the same `sys_days`. But as in this example, they are occasionally different. It is not unheard of for computer systems to conflate these two very similar types, resulting in bugs that are hard to find because they are relatively rare. Having the C++ type system help you keep these similar but different concepts distinct is a solid step towards finding bugs at compile time!

Reference

Here is a detailed specification of the entire library. This specification is detailed enough that you could write your own implementation from it if desired. But feel free to use this one instead. Each type, and each operation is simple and predictable.

durations

- days
- weeks
- years

time_point

- sys_days

types

- last_week
- weekday
- weeknum
- year
- year_weeknum
- year_lastweek
- weeknum_weekday
- lastweek_weekday
- year_weeknum_weekday

year_lastweek_weekday

date composition operators

constexpr year_weeknum operator/(const year& y, const weeknum& wn) noexcept;
constexpr year_weeknum operator/(const year& y, int wn) noexcept;

constexpr year_lastweek operator/(const year& y, last_week wn) noexcept;

constexpr weeknum_weekday operator/(const weeknum& wn, const weekday& wd) noexcept;
constexpr weeknum_weekday operator/(const weeknum& wn, int wd) noexcept;
constexpr weeknum_weekday operator/(const weekday& wd, const weeknum& wn) noexcept;
constexpr weeknum_weekday operator/(const weekday& wd, int wn) noexcept;

constexpr lastweek_weekday operator/(const last_week& wn, const weekday& wd) noexcept;
constexpr lastweek_weekday operator/(const last_week& wn, int wd) noexcept;
constexpr lastweek_weekday operator/(const weekday& wd, const last_week& wn) noexcept;

constexpr year_weeknum_weekday operator/(const year_weeknum& ywn, const weekday& wd) noexcept;
constexpr year_weeknum_weekday operator/(const year_weeknum& ywn, int wd) noexcept;
constexpr year_weeknum_weekday operator/(const weeknum_weekday& wnwd, const year& y) noexcept;
constexpr year_weeknum_weekday operator/(const weeknum_weekday& wnwd, int y) noexcept;

constexpr year_lastweek_weekday operator/(const year_lastweek& ylw, const weekday& wd) noexcept;
constexpr year_lastweek_weekday operator/(const year_lastweek& ylw, int wd) noexcept;

constexpr year_lastweek_weekday operator/(const lastweek_weekday& lwwd, const year& y) noexcept;
constexpr year_lastweek_weekday operator/(const lastweek_weekday& lwwd, int y) noexcept;

Everything here is contained in the namespace `iso_week`. The literal operators, and the constexpr field literals (e.g. `sun`, `last`, etc.) are in namespace `iso_week::literals` and imported into namespace `iso_week`.

days

`days` is a `std::chrono::duration` with a tick period of 24 hours. This definition is not an SI unit but is accepted for use with SI. `days` is the resultant type when subtracting two `sys_dayss`.

```
using days = std::chrono::duration
    <int, std::ratio_multiply<std::ratio<24>, std::chrono::hours::period>>;
```

weeks

`weeks` is a `std::chrono::duration` with a tick period of 7 days. This definition is widely recognized and predates the Gregorian calendar. It is consistent with ISO 8601. `weeks` will implicitly convert to `days` but not vice-versa.

```
using weeks = std::chrono::duration
    <int, std::ratio_multiply<std::ratio<7>, days::period>>;
```

years

`years` is a `std::chrono::duration` with a tick period of 365.2425 days. This definition accurately describes the length of the average year in the ISO week calendar. `years` is the resultant type when subtracting two `year` field-based time points. `years` is not implicitly convertible to `days` or `weeks` nor vice-versa.

```
using years = std::chrono::duration
    <int, std::ratio_multiply<std::ratio<146097, 400>, days::period>>;
```

sys_days

`sys_days` is a `std::chrono::time_point` using `std::chrono::system_clock` and `days`. This makes `sys_days` interoperable with `std::chrono::system_clock::time_point`. It is simply a count of days since the epoch of `std::chrono::system_clock` which in every implementation is Jan. 1, 1970. `sys_days` is a serial-based time point with a resolution of `days`.

```
using sys_days = std::chrono::time_point<std::chrono::system_clock, days>;
```

last_week

`last_week` is a `struct` that is `CopyConstructible`. There exists a `constexpr` instance of `last_week` named `last`. This is simply a tag type. It is used to indicate the last week of the year.

```
struct last_week
{
    explicit last_week() = default;
};

inline namespace literals
{

constexpr iso_week::last_week last{};

}
```

I am leading the C++ standard here a little with the `explicit` qualifier on the default constructor. This compiles today, but doesn't have quite the desired semantics. But it will when CWG 1518 passes. This will make it so that `{}` won't implicitly convert to `last_week`. This addresses the concern expressed in LWG 2510.

weekday

Synopsis

```
class weekday
{
    unsigned char wd_;  // exposition only
public:
    explicit constexpr weekday(unsigned wd) noexcept;
    constexpr weekday(const sys_days& dp) noexcept;
    constexpr weekday(date::weekday wd) noexcept;
    explicit weekday(int) = delete;

    weekday& operator++() noexcept;
    weekday operator++(int) noexcept;
    weekday& operator--() noexcept;
    weekday operator--(int) noexcept;

    weekday& operator+=(const days& d) noexcept;
    weekday& operator-=(const days& d) noexcept;

    constexpr explicit operator unsigned() const noexcept;
    constexpr operator date::weekday() const noexcept;
    constexpr bool ok() const noexcept;
};

constexpr bool operator==(const weekday& x, const weekday& y) noexcept;
constexpr bool operator!=(const weekday& x, const weekday& y) noexcept;

constexpr weekday operator+(const weekday& x, const days&    y) noexcept;
constexpr weekday operator+(const days&    x, const weekday& y) noexcept;
constexpr weekday operator-(const weekday& x, const days&    y) noexcept;
constexpr days    operator-(const weekday& x, const weekday& y) noexcept;

std::ostream& operator<<(std::ostream& os, const weekday& wd);

inline namespace literals
{
constexpr weekday mon{1u};
constexpr weekday tue{2u};
constexpr weekday wed{3u};
constexpr weekday thu{4u};
constexpr weekday fri{5u};
constexpr weekday sat{6u};
constexpr weekday sun{7u};
}  // namespace literals
```

Overview

`weekday` represents a day of the week in the ISO week calendar. It should only be representing values in the range 1 to 7, corresponding to Monday thru Sunday. However it may hold values outside this range. It can be constructed with any `unsigned` value, which will be subsequently truncated to fit into `weekday`'s internal storage. `weekday` is equality comparable. `weekday` is not less-than comparable because there is no universal consensus on which day is the first day of the week. This design

chooses the encoding of 1 to 7 to represent Monday thru Sunday only because this is consistent with ISO 8601. However `weekday`'s comparison and arithmetic operations treat the days of the week as a circular range, with no beginning and no end. One can stream out a `weekday` for debugging purposes. `weekday` has explicit conversions to and from `unsigned`. There are 7 `weekday` constants, one for each day of the week.

A `weekday` can be implicitly constructed from a `sys_days`. This is the computation that discovers the day of the week of an arbitrary date.

A `weekday` can be implicitly converted to or from a `date::weekday`. `iso_week::weekday` and `date::weekday` are distinct types, though they represent very similar concepts. `date::weekday` can be indexed, and `iso_week::weekday` can not. `date::weekday` is encoded as [0, 6] representing [Sunday, Saturday], while `iso_week::weekday` is encoded as [1, 7] representing [Monday, Sunday]. The conversion functions will correctly translate between these encodings.

Specification

`weekday` is a trivially copyable class type.
`weekday` is a standard-layout class type.
`weekday` is a literal class type.

```
explicit constexpr weekday::weekday(unsigned wd) noexcept;
```

Effects: Constructs an object of type `weekday` by constructing `wd_` with `wd`.

```
constexpr weekday(const sys_days& dp) noexcept;
```

Effects: Constructs an object of type `weekday` by computing what day of the week corresponds to the `sys_days dp`, and representing that day of the week in `wd_`.

Example: If `dp` represents 1970-01-01, the constructed `weekday` shall represent Thursday by storing 4 in `wd_`.

```
constexpr weekday(date::weekday wd) noexcept;
```

Effects: Constructs an object of type `weekday` from a `date::weekday`. This changes the underlying encoding from [0, 6] -> [sun, sat] to [1, 7] -> [mon, sun].

```
weekday& weekday::operator++() noexcept;
```

Effects: If `wd_ != 6`, `++wd_`. Otherwise sets `wd_` to 0.

Returns: `*this`.

```
weekday weekday::operator++(int) noexcept;
```

Effects: `++(*this)`.

Returns: A copy of `*this` as it existed on entry to this member function.

```
weekday& weekday::operator--() noexcept;
```

Effects: If `wd_ != 0`, `--wd_`. Otherwise sets `wd_` to 6.

Returns: `*this`.

```
weekday weekday::operator--(int) noexcept;
```

Effects: `--(*this)`.

Returns: A copy of `*this` as it existed on entry to this member function.

```
weekday& weekday::operator+=(const days& d) noexcept;
```

Effects: `*this = *this + d`.

Returns: `*this`.

```
weekday& weekday::operator-=(const days& d) noexcept;
```

Effects: `*this = *this - d`.

Returns: `*this`.

```
constexpr explicit weekday::operator unsigned() const noexcept;
```

Returns: `wd_`.

```
constexpr operator date::weekday() const noexcept;
```

Returns: An object of type `date::weekday` constructed from `*this`. This changes the underlying encoding from [1, 7] -> [mon, sun] to [0, 6] -> [sun, sat].

```
constexpr bool weekday::ok() const noexcept;
```

Returns: `1 <= wd_ && wd_ <= 7`.

```
constexpr bool operator==(const weekday& x, const weekday& y) noexcept;
```

Returns: `unsigned{x} == unsigned{y}`.

```
constexpr bool operator!=(const weekday& x, const weekday& y) noexcept;
```

Returns: `!(x == y)`.

```
constexpr weekday  operator+(const weekday&  x, const days& y) noexcept;
```

Returns: A `weekday` for which `ok() == true` and is found as if by incrementing (or decrementing if `y < days{0}`) `x`, `y` times. If `weekday.ok() == false` prior to this operation, behaves as if `*this` is first brought into the range [1, 7] by modular arithmetic. [*Note:* For example `weekday{0}` becomes `weekday{7}`. — *end note*]

Complexity: O(1) with respect to the value of `y`. That is, repeated increments or decrements is not a valid implementation.

Example: `sun + days{6} == sat`.

```
constexpr weekday  operator+(const days& x, const weekday&  y) noexcept;
```

Returns: `y + x`.

```
constexpr weekday  operator-(const weekday&  x, const days& y) noexcept;
```

Returns: `x + -y`.

```
constexpr days operator-(const weekday&  x, const weekday&  y) noexcept;
```

Requires: `x.ok() == true` and `y.ok() == true`.

Returns: A value of `days` in the range of `days{0}` to `days{6}` inclusive.

Remarks: The returned value `d` shall satisfy the equality: `y + d == x`.

Example: `sat - sun == days{6}`.

```
std::ostream& operator<<(std::ostream& os, const weekday& wd);
```

Effects: If `ok() == true` outputs the same string that would be output for the weekday field by `asctime`, assuming the conversion `unsigned{date::weekday{*this}}`. Otherwise outputs `unsigned{wd} << " is not a valid weekday"`.

Returns: `os`.

`weeknum`

Synopsis

```
class weeknum
{
    unsigned char wn_;  // exposition only

public:
    explicit constexpr weeknum(unsigned wn) noexcept;

    weeknum& operator++() noexcept;
    weeknum operator++(int) noexcept;
    weeknum& operator--() noexcept;
    weeknum operator--(int) noexcept;
```

```
    weeknum& operator+=(const weeks& w) noexcept;
    weeknum& operator-=(const weeks& w) noexcept;

    constexpr explicit operator unsigned() const noexcept;
    constexpr bool ok() const noexcept;
};

constexpr bool operator==(const weeknum& x, const weeknum& y) noexcept;
constexpr bool operator!=(const weeknum& x, const weeknum& y) noexcept;
constexpr bool operator< (const weeknum& x, const weeknum& y) noexcept;
constexpr bool operator> (const weeknum& x, const weeknum& y) noexcept;
constexpr bool operator<=(const weeknum& x, const weeknum& y) noexcept;
constexpr bool operator>=(const weeknum& x, const weeknum& y) noexcept;

constexpr weeknum operator+(const weeknum& x, const weeks&   y) noexcept;
constexpr weeknum operator+(const weeks&   x, const weeknum& y) noexcept;
constexpr weeknum operator-(const weeknum& x, const weeks&   y) noexcept;
constexpr weeks   operator-(const weeknum& x, const weeknum& y) noexcept;

std::ostream& operator<<(std::ostream& os, const weeknum& wn);

inline namespace literals
{

constexpr weeknum operator "" _w(unsigned long long wn) noexcept;

}
```

Overview

`weeknum` represents a week number of of the ISO week-year [1, 53]. It should only be representing values in the range 1 to 53. However it may hold values outside this range. It can be constructed with any `unsigned` value, which will be subsequently truncated to fit into `weeknum`'s internal storage. `weeknum` is equality and less-than comparable, and participates in basic arithmetic with `weeks` representing the quantity between any two `weeknum`'s. One can stream out a `weeknum` for debugging purposes. `weeknum` has explicit conversions to and from `unsigned`.

Specification

`weeknum` is a trivially copyable class type.
`weeknum` is a standard-layout class type.
`weeknum` is a literal class type.

```
explicit constexpr weeknum::weeknum(unsigned wn) noexcept;
```

Effects: Constructs an object of type `weeknum` by constructing `wn_` with `wn`.

```
weeknum& weeknum::operator++() noexcept;
```

Effects: `++wn_`.

Returns: `*this`.

```
weeknum weeknum::operator++(int) noexcept;
```

Effects: `++(*this)`.

Returns: A copy of `*this` as it existed on entry to this member function.

```
weeknum& weeknum::operator--() noexcept;
```

Effects: `--wn_`.

Returns: `*this`.

```
weeknum weeknum::operator--(int) noexcept;
```

Effects: `--(*this)`.

Returns: A copy of `*this` as it existed on entry to this member function.

```
weeknum& weeknum::operator+=(const weeks& w) noexcept;
```

Effects: `*this = *this + w`.

Returns: `*this`.

```
weeknum& weeknum::operator-=(const weeks& w) noexcept;
```

Effects: `*this = *this - w`.

Returns: `*this`.

```
constexpr explicit weeknum::operator unsigned() const noexcept;
```

Returns: `wn_`.

```
constexpr bool weeknum::ok() const noexcept;
```

Returns: `1 <= wn_ && wn_ <= 53`.

```
constexpr bool operator==(const weeknum& x, const weeknum& y) noexcept;
```

Returns: `unsigned{x} == unsigned{y}`.

```
constexpr bool operator!=(const weeknum& x, const weeknum& y) noexcept;
```

Returns: `!(x == y)`.

```
constexpr bool operator< (const weeknum& x, const weeknum& y) noexcept;
```

Returns: `unsigned{x} < unsigned{y}`.

```
constexpr bool operator> (const weeknum& x, const weeknum& y) noexcept;
```

Returns: `y < x`.

```
constexpr bool operator<=(const weeknum& x, const weeknum& y) noexcept;
```

Returns: `!(y < x)`.

```
constexpr bool operator>=(const weeknum& x, const weeknum& y) noexcept;
```

Returns: `!(x < y)`.

```
constexpr weeknum  operator+(const weeknum&  x, const weeks& y) noexcept;
```

Returns: `weeknum{unsigned{x} + static_cast<unsigned>(y.count())}`

```
constexpr weeknum  operator+(const weeks& x, const weeknum&  y) noexcept;
```

Returns: `y + x`.

```
constexpr weeknum  operator-(const weeknum&  x, const weeks& y) noexcept;
```

Returns: `x + -y`.

```
constexpr weeks operator-(const weeknum&  x, const weeknum&  y) noexcept;
```

Returns: `weeks{static_cast<weeks::rep>(unsigned{x}) -
static_cast<weeks::rep>(unsigned{y})}`.

```
std::ostream& operator<<(std::ostream& os, const weeknum& wn);
```

Effects: Inserts 'W' followed by a decimal integral text representation of length 2, prefixed by '0' if the value is less than or equal to 9.

Returns: `os`.

Example: `5_w` is output as `W05`.

```
constexpr weeknum operator "" _w(unsigned long long wn) noexcept;
```

Returns: `weeknum{static_cast<unsigned>(wn)}`.

year

Synopsis

```
class year
{
    short y_;  // exposition only

public:
    explicit constexpr year(int y) noexcept;

    year& operator++() noexcept;
    year operator++(int) noexcept;
    year& operator--() noexcept;
    year operator--(int) noexcept;

    year& operator+=(const years& y) noexcept;
    year& operator-=(const years& y) noexcept;

    constexpr explicit operator int() const noexcept;
    constexpr bool ok() const noexcept;

    static constexpr year min() noexcept;
    static constexpr year max() noexcept;
};

constexpr bool operator==(const year& x, const year& y) noexcept;
constexpr bool operator!=(const year& x, const year& y) noexcept;
constexpr bool operator< (const year& x, const year& y) noexcept;
constexpr bool operator> (const year& x, const year& y) noexcept;
constexpr bool operator<=(const year& x, const year& y) noexcept;
constexpr bool operator>=(const year& x, const year& y) noexcept;

constexpr year  operator+(const year&  x, const years& y) noexcept;
constexpr year  operator+(const years& x, const year&  y) noexcept;
constexpr year  operator-(const year&  x, const years& y) noexcept;
constexpr years operator-(const year&  x, const year&  y) noexcept;

std::ostream& operator<<(std::ostream& os, const year& y);

inline namespace literals
{
constexpr year operator "" _y(unsigned long long y) noexcept;
}
```

Overview

`year` represents a year in the ISO week date calendar. It shall represent values in the range `[min(), max()]`. It can be constructed with any `int` value, which will be subsequently truncated to fit into `year`'s internal storage. `year` is equality and less-than comparable, and participates in basic arithmetic with `years` representing the quantity between any two `year`'s. One can form a `year` literal with `_y`. And one can stream out a `year` for debugging purposes. `year` has explicit conversions to and from `int`.

Specification

`year` is a trivially copyable class type.
`year` is a standard-layout class type.
`year` is a literal class type.

```
explicit constexpr year::year(int y) noexcept;
```

Effects: Constructs an object of type `year` by constructing `y_` with `y`.

```
year& year::operator++() noexcept;
```

Effects: `++y_`.

Returns: `*this`.

```
year year::operator++(int) noexcept;
```

Effects: `++(*this)`.

Returns: A copy of `*this` as it existed on entry to this member function.

```
year& year::operator--() noexcept;
```

Effects: `--y_`.

Returns: `*this`.

```
year year::operator--(int) noexcept;
```

Effects: `--(*this)`.

Returns: A copy of `*this` as it existed on entry to this member function.

```
year& year::operator+=(const years& y) noexcept;
```

Effects: `*this = *this + y`.

Returns: `*this`.

```
year& year::operator-=(const years& y) noexcept;
```

Effects: `*this = *this - y`.

Returns: `*this`.

```
constexpr explicit year::operator int() const noexcept;
```

Returns: `y_`.

```
constexpr bool year::ok() const noexcept;
```

Returns: `min() <= *this && *this <= max()`.

```
static constexpr year year::min() noexcept;
```

Returns: A `year` constructed with the minimum representable year number. This year shall be a value such that `sys_days{min()/1_w/mon} + Unit{0}`, where `Unit` is one of `microseconds`, `milliseconds`, `seconds`, `minutes`, or `hours`, there shall be no overflow. [*Note:* `nanoseconds` is intentionally omitted from this list. — *end note*]

```
static constexpr year year::max() noexcept;
```

Returns: A `year` constructed with the maximum representable year number. This year shall be a value such that `sys_days{max()/last/sun} + Unit{0}`, where `Unit` is one of `microseconds`, `milliseconds`, `seconds`, `minutes`, or `hours`, there shall be no overflow. [*Note:* `nanoseconds` is intentionally omitted from this list. — *end note*]

```
constexpr bool operator==(const year& x, const year& y) noexcept;
```

Returns: `int{x} == int{y}`.

```
constexpr bool operator!=(const year& x, const year& y) noexcept;
```

Returns: `!(x == y)`.

```
constexpr bool operator< (const year& x, const year& y) noexcept;
```

Returns: `int{x} < int{y}`.

```
constexpr bool operator> (const year& x, const year& y) noexcept;
```

Returns: `y < x`.

```
constexpr bool operator<=(const year& x, const year& y) noexcept;
```

Returns: `!(y < x)`.

```
constexpr bool operator>=(const year& x, const year& y) noexcept;
```

Returns: `!(x < y)`.

```
constexpr year  operator+(const year&  x, const years& y) noexcept;
```

Returns: `year{int{x} + y.count()}`.

```
constexpr year  operator+(const years& x, const year&  y) noexcept;
```

Returns: `y + x`.

```
constexpr year  operator-(const year&  x, const years& y) noexcept;
```

Returns: `x + -y`.

```
constexpr years operator-(const year&  x, const year&  y) noexcept;
```

Returns: `years{int{x} - int{y}}`.

```
constexpr year operator "" _y(unsigned long long y) noexcept;
```

Returns: `year{static_cast<int>(y)}`.

```
std::ostream& operator<<(std::ostream& os, const year& y);
```

Effects: Inserts a signed decimal integral text representation of `y` into `os`. If the year is less than four decimal digits, pads the year with `'0'` to four digits. If the year is negative, prefixes with `'-'`.

Returns: `os`.

```
constexpr year operator "" _y(unsigned long long y) noexcept;
```

Returns: `year{static_cast<int>(y)}`.

`year_weeknum`

Synopsis

```
class year_weeknum
{
    iso_week::year    y_;   // exposition only
    iso_week::weeknum wn_;  // exposition only

public:
    constexpr year_weeknum(const iso_week::year& y, const iso_week::weeknum& wn) noexcept;

    constexpr iso_week::year    year()    const noexcept;
    constexpr iso_week::weeknum weeknum() const noexcept;

    year_weeknum& operator+=(const years& dy) noexcept;
    year_weeknum& operator-=(const years& dy) noexcept;

    constexpr bool ok() const noexcept;
};

constexpr bool operator==(const year_weeknum& x, const year_weeknum& y) noexcept;
constexpr bool operator!=(const year_weeknum& x, const year_weeknum& y) noexcept;
constexpr bool operator< (const year_weeknum& x, const year_weeknum& y) noexcept;
constexpr bool operator> (const year_weeknum& x, const year_weeknum& y) noexcept;
constexpr bool operator<=(const year_weeknum& x, const year_weeknum& y) noexcept;
constexpr bool operator>=(const year_weeknum& x, const year_weeknum& y) noexcept;

constexpr year_weeknum operator+(const year_weeknum& ym, const years& dy) noexcept;
constexpr year_weeknum operator+(const years& dy, const year_weeknum& ym) noexcept;
constexpr year_weeknum operator-(const year_weeknum& ym, const years& dy) noexcept;

std::ostream& operator<<(std::ostream& os, const year_weeknum& ym);
```

Overview

`year_weeknum` is simply a collection of a `year` and a `weeknum`. It represents a specific week of a year, but not the day of the week. One can perform year-oriented arithmetic with a `year_weeknum`. And `year_weeknum` is equality and less-than comparable.

Specification

`year_weeknum` is a trivially copyable class type.
`year_weeknum` is a standard-layout class type.
`year_weeknum` is a literal class type.

```
constexpr year_weeknum::year_weeknum(const iso_week::year& y, const iso_week::weeknum& wn) noexcept;
```

Effects: Constructs an object of type `year_weeknum` by constructing `y_` with `y` and `wn_` with `wn`.

```
constexpr iso_week::year year_weeknum::year() const noexcept;
```

Returns: `y_`.

```
constexpr iso_week::weeknum year_weeknum::weeknum() const noexcept;
```

Returns: `wn_`.

```
year_weeknum& year_weeknum::operator+=(const years& dy) noexcept;
```

Effects: `*this = *this + dy`.

Returns: `*this`.

```
year_weeknum& year_weeknum::operator-=(const years& dy) noexcept;
```

Effects: `*this = *this - dy`.

Returns: `*this`.

```
constexpr bool year_weeknum::ok() const noexcept;
```

Returns: `y_.ok() && 1_w <= wn_ && wn_ <= (y_/last).weeknum()`.

```
constexpr bool operator==(const year_weeknum& x, const year_weeknum& y) noexcept;
```

Returns: `x.year() == y.year() && x.weeknum() == y.weeknum()`.

```
constexpr bool operator!=(const year_weeknum& x, const year_weeknum& y) noexcept;
```

Returns: `!(x == y)`.

```
constexpr bool operator<(const year_weeknum& x, const year_weeknum& y) noexcept;
```

Returns:
```
x.year() < y.year() ? true
    : (x.year() > y.year() ? false
    : (x.weeknum() < y.weeknum())).
```

```
constexpr bool operator>(const year_weeknum& x, const year_weeknum& y) noexcept;
```

Returns: `y < x`.

```
constexpr bool operator<=(const year_weeknum& x, const year_weeknum& y) noexcept;
```

Returns: `!(y < x)`.

```
constexpr bool operator>=(const year_weeknum& x, const year_weeknum& y) noexcept;
```

Returns: `!(x < y)`.

```
constexpr year_weeknum operator+(const year_weeknum& ym, const years& dy) noexcept;
```

Returns: `(ym.year() + dy) / ym.weeknum()`.

```
constexpr year_weeknum operator+(const years& dy, const year_weeknum& ym) noexcept;
```

Returns: `ym + dy`.

```
constexpr year_weeknum operator-(const year_weeknum& ym, const years& dy) noexcept;
```

Returns: `ym + -dy`.

```
std::ostream& operator<<(std::ostream& os, const year_weeknum& ywn);
```

Effects: `os << ywn.year() << '-' << ywn.weeknum()`.

Returns: `os`.

year_lastweek

Synopsis

```
class year_lastweek
{
    iso_week::year y_;  // exposition only

public:
```

```
    explicit constexpr year_lastweek(const iso_week::year& y) noexcept;

    constexpr iso_week::year    year()    const noexcept;
    constexpr iso_week::weeknum weeknum() const noexcept;

    year_lastweek& operator+=(const years& dy) noexcept;
    year_lastweek& operator-=(const years& dy) noexcept;

    constexpr bool ok() const noexcept;
};

constexpr bool operator==(const year_lastweek& x, const year_lastweek& y) noexcept;
constexpr bool operator!=(const year_lastweek& x, const year_lastweek& y) noexcept;
constexpr bool operator< (const year_lastweek& x, const year_lastweek& y) noexcept;
constexpr bool operator> (const year_lastweek& x, const year_lastweek& y) noexcept;
constexpr bool operator<=(const year_lastweek& x, const year_lastweek& y) noexcept;
constexpr bool operator>=(const year_lastweek& x, const year_lastweek& y) noexcept;

constexpr year_lastweek operator+(const year_lastweek& ym, const years& dy) noexcept;
constexpr year_lastweek operator+(const years& dy, const year_lastweek& ym) noexcept;
constexpr year_lastweek operator-(const year_lastweek& ym, const years& dy) noexcept;

std::ostream& operator<<(std::ostream& os, const year_lastweek& ym);
```

Overview

`year_lastweek` represents a ISO week `year` and the last week of that year. One can perform year-oriented arithmetic with a `year_lastweek`. And `year_lastweek` is equality and less-than comparable.

Specification

`year_lastweek` is a trivially copyable class type.
`year_lastweek` is a standard-layout class type.
`year_lastweek` is a literal class type.

```
explicit constexpr year_lastweek::year_lastweek(const iso_week::year& y) noexcept;
```

Effects: Constructs an object of type `year_lastweek` by constructing `y_` with `y`.

```
constexpr iso_week::year year_lastweek::year() const noexcept;
```

Returns: `y_`.

```
constexpr iso_week::weeknum year_lastweek::weeknum() const noexcept;
```

Returns: The ISO week number for the last week of the year `y_`.

```
year_lastweek& year_lastweek::operator+=(const years& dy) noexcept;
```

Effects: `*this = *this + dy`.

Returns: `*this`.

```
year_lastweek& year_lastweek::operator-=(const years& dy) noexcept;
```

Effects: `*this = *this - dy`.

Returns: `*this`.

```
constexpr bool year_lastweek::ok() const noexcept;
```

Returns: `y_.ok()`.

```
constexpr bool operator==(const year_lastweek& x, const year_lastweek& y) noexcept;
```

Returns: `x.year() == y.year()`.

```
constexpr bool operator!=(const year_lastweek& x, const year_lastweek& y) noexcept;
```

Returns: `!(x == y)`.

```
constexpr bool operator<(const year_lastweek& x, const year_lastweek& y) noexcept;
```

Returns: `x.year() < y.year()`.

```
constexpr bool operator>(const year_lastweek& x, const year_lastweek& y) noexcept;
```

Returns: `y < x`.

```
constexpr bool operator<=(const year_lastweek& x, const year_lastweek& y) noexcept;
```

Returns: `!(y < x)`.

```
constexpr bool operator>=(const year_lastweek& x, const year_lastweek& y) noexcept;
```

Returns: `!(x < y)`.

```
constexpr year_lastweek operator+(const year_lastweek& ym, const years& dy) noexcept;
```

Returns: `year_lastweek{ym.year() + dy}`.

```
constexpr year_lastweek operator+(const years& dy, const year_lastweek& ym) noexcept;
```

Returns: `ym + dy`.

```
constexpr year_lastweek operator-(const year_lastweek& ym, const years& dy) noexcept;
```

Returns: `ym + -dy`.

```
std::ostream& operator<<(std::ostream& os, const year_lastweek& ywn);
```

Effects: `os << ywn.year() << "-W last"`.

Returns: `os`.

`weeknum_weekday`

Synopsis

```
class weeknum_weekday
{
    iso_week::weeknum wn_;  // exposition only
    iso_week::weekday wd_;  // exposition only

public:
    constexpr weeknum_weekday(const iso_week::weeknum& wn,
                              const iso_week::weekday& wd) noexcept;

    constexpr iso_week::weeknum weeknum() const noexcept;
    constexpr iso_week::weekday weekday() const noexcept;

    constexpr bool ok() const noexcept;
};

constexpr bool operator==(const weeknum_weekday& x, const weeknum_weekday& y) noexcept;
constexpr bool operator!=(const weeknum_weekday& x, const weeknum_weekday& y) noexcept;
constexpr bool operator< (const weeknum_weekday& x, const weeknum_weekday& y) noexcept;
constexpr bool operator> (const weeknum_weekday& x, const weeknum_weekday& y) noexcept;
constexpr bool operator<=(const weeknum_weekday& x, const weeknum_weekday& y) noexcept;
constexpr bool operator>=(const weeknum_weekday& x, const weeknum_weekday& y) noexcept;

std::ostream& operator<<(std::ostream& os, const weeknum_weekday& md);
```

Overview

`weeknum_weekday` is simply a collection of a `weeknum` and a `weekday`. It represents a specific week and day of the week, but an unspecified year. `weeknum_weekday` is equality and less-than comparable.

Specification

`weeknum_weekday` is a trivially copyable class type.
`weeknum_weekday` is a standard-layout class type.
`weeknum_weekday` is a literal class type.

```
constexpr weeknum_weekday::weeknum_weekday(const iso_week::weeknum& wn,
                                           const iso_week::weekday& wd) noexcept;
```

Effects: Constructs an object of type `weeknum_weekday` by constructing `wn_` with `wn` and `wd_` with `wd`.

```
constexpr iso_week::weeknum weeknum_weekday::weeknum() const noexcept;
```

Returns: `wn_`.

```
constexpr iso_week::weekday weeknum_weekday::weekday() const noexcept;
```

Returns: `wd_`.

```
constexpr bool weeknum_weekday::ok() const noexcept;
```

Returns: `wn_.ok() && wd_.ok()`.

```
constexpr bool operator==(const weeknum_weekday& x, const weeknum_weekday& y) noexcept;
```

Returns: `x.weeknum() == y.weeknum() && x.weekday() == y.weekday()`.

```
constexpr bool operator!=(const weeknum_weekday& x, const weeknum_weekday& y) noexcept;
```

Returns: `!(x == y)`.

```
constexpr bool operator<(const weeknum_weekday& x, const weeknum_weekday& y) noexcept;
```

Returns:
```
x.weeknum() < y.weeknum() ? true
     : (x.weeknum() > y.weeknum() ? false
     : (unsigned{x.weekday()} < unsigned{y.weekday()})).
```

```
constexpr bool operator>(const weeknum_weekday& x, const weeknum_weekday& y) noexcept;
```

Returns: `y < x`.

```
constexpr bool operator<=(const weeknum_weekday& x, const weeknum_weekday& y) noexcept;
```

Returns: `!(y < x)`.

```
constexpr bool operator>=(const weeknum_weekday& x, const weeknum_weekday& y) noexcept;
```

Returns: `!(x < y)`.

```
std::ostream& operator<<(std::ostream& os, const weeknum_weekday& md);
```

Effects: `os << md.weeknum() << '-' << md.weekday()`.

Returns: `os`.

`lastweek_weekday`

Synopsis

```
class lastweek_weekday
{
    iso_week::weekday wd_;  // exposition only

public:
    explicit constexpr lastweek_weekday(const iso_week::weekday& wd) noexcept;

    constexpr iso_week::weekday weekday() const noexcept;

    constexpr bool ok() const noexcept;
};

constexpr bool operator==(const lastweek_weekday& x, const lastweek_weekday& y) noexcept;
constexpr bool operator!=(const lastweek_weekday& x, const lastweek_weekday& y) noexcept;
constexpr bool operator< (const lastweek_weekday& x, const lastweek_weekday& y) noexcept;
constexpr bool operator> (const lastweek_weekday& x, const lastweek_weekday& y) noexcept;
constexpr bool operator<=(const lastweek_weekday& x, const lastweek_weekday& y) noexcept;
constexpr bool operator>=(const lastweek_weekday& x, const lastweek_weekday& y) noexcept;

std::ostream& operator<<(std::ostream& os, const lastweek_weekday& md);
```

Overview

`lastweek_weekday` represents a weekday in the last week of an unspecified year. `lastweek_weekday` is equality and less-than comparable.

Specification

`lastweek_weekday` is a trivially copyable class type.
`lastweek_weekday` is a standard-layout class type.
`lastweek_weekday` is a literal class type.

```
explicit constexpr lastweek_weekday::lastweek_weekday(const iso_week::weekday& wd) noexcept;
```

Effects: Constructs an object of type `lastweek_weekday` by constructing `wd_` with `wd`.

```
constexpr iso_week::weekday lastweek_weekday::weekday() const noexcept;
```

Returns: `wd_`.

```
constexpr bool lastweek_weekday::ok() const noexcept;
```

Returns: `wd_.ok()`.

```
constexpr bool operator==(const lastweek_weekday& x, const lastweek_weekday& y) noexcept;
```

Returns: `x.weekday() == y.weekday()`.

```
constexpr bool operator!=(const lastweek_weekday& x, const lastweek_weekday& y) noexcept;
```

Returns: `!(x == y)`.

```
constexpr bool operator<(const lastweek_weekday& x, const lastweek_weekday& y) noexcept;
```

Returns: `unsigned{x.weekday()} < unsigned{y.weekday()}`.

```
constexpr bool operator>(const lastweek_weekday& x, const lastweek_weekday& y) noexcept;
```

Returns: `y < x`.

```
constexpr bool operator<=(const lastweek_weekday& x, const lastweek_weekday& y) noexcept;
```

Returns: `!(y < x)`.

```
constexpr bool operator>=(const lastweek_weekday& x, const lastweek_weekday& y) noexcept;
```

Returns: `!(x < y)`.

```
std::ostream& operator<<(std::ostream& os, const lastweek_weekday& md);
```

Effects: `os << "W last-" << md.weekday()`.

Returns: `os`.

`year_weeknum_weekday`

Synopsis

```
class year_weeknum_weekday
{
    iso_week::year    y_;   // exposition only
    iso_week::weeknum wn_;  // exposition only
    iso_week::weekday wd_;  // exposition only

public:
    constexpr year_weeknum_weekday(const iso_week::year& y, const iso_week::weeknum& wn,
                                   const iso_week::weekday& wd) noexcept;
    constexpr year_weeknum_weekday(const year_lastweek_weekday& x) noexcept;
    constexpr year_weeknum_weekday(const sys_days& dp) noexcept;

    year_weeknum_weekday& operator+=(const years& y) noexcept;
    year_weeknum_weekday& operator-=(const years& y) noexcept;

    constexpr iso_week::year    year()    const noexcept;
    constexpr iso_week::weeknum weeknum() const noexcept;
    constexpr iso_week::weekday weekday() const noexcept;

    constexpr operator sys_days() const noexcept;
    constexpr bool ok() const noexcept;
};
```

```
constexpr bool operator==(const year_weeknum_weekday& x, const year_weeknum_weekday& y) noexcept;
constexpr bool operator!=(const year_weeknum_weekday& x, const year_weeknum_weekday& y) noexcept;
constexpr bool operator< (const year_weeknum_weekday& x, const year_weeknum_weekday& y) noexcept;
constexpr bool operator> (const year_weeknum_weekday& x, const year_weeknum_weekday& y) noexcept;
constexpr bool operator<=(const year_weeknum_weekday& x, const year_weeknum_weekday& y) noexcept;
constexpr bool operator>=(const year_weeknum_weekday& x, const year_weeknum_weekday& y) noexcept;

constexpr year_weeknum_weekday operator+(const year_weeknum_weekday& x, const years& y) noexcept;
constexpr year_weeknum_weekday operator+(const years& y, const year_weeknum_weekday& x) noexcept;
constexpr year_weeknum_weekday operator-(const year_weeknum_weekday& x, const years& y) noexcept;

std::ostream& operator<<(std::ostream& os, const year_weeknum_weekday& x);
```

Overview

`year_weeknum_weekday` represents a `year`, `weeknum`, and `weekday` in the ISO week date calendar. One can observe each field. `year_weeknum_weekday` supports year-oriented arithmetic. There is an implicit conversion to and from `sys_days`. There is also an implicit conversion from `year_lastweek_weekday`. `year_weeknum_weekday` is equality and less-than comparable.

Specification

`year_weeknum_weekday` is a trivially copyable class type.
`year_weeknum_weekday` is a standard-layout class type.
`year_weeknum_weekday` is a literal class type.

```
constexpr year_weeknum_weekday::year_weeknum_weekday(const iso_week::year& y,
                                                     const iso_week::weeknum& wn,
                                                     const iso_week::weekday& wd) noexcept;
```

Effects: Constructs an object of type `year_weeknum_weekday` by constructing `y_` with `y`, `wn_` with `wn`, and `wd_` with `wd`.

```
constexpr year_weeknum_weekday::year_weeknum_weekday(const year_lastweek_weekday& x) noexcept;
```

Effects: Constructs an object of type `year_weeknum_weekday` by constructing `y_` with `x.year()`, `wn_` with `x.weeknum()`, and `wd_` with `x.weekday()`.

```
constexpr year_weeknum_weekday::year_weeknum_weekday(const sys_days& dp) noexcept;
```

Effects: Constructs an object of type `year_weeknum_weekday` which corresponds to the date represented by `dp`.

Remarks: For any value of `year_weeknum_weekday`, `x`, for which `x.ok()` is `true`, this equality will also be `true`: `x == year_weeknum_weekday{sys_days{x}}`.

```
year_weeknum_weekday& year_weeknum_weekday::operator+=(const years& y) noexcept;
```

Effectx: `*this = *this + y`.

Returns: `*this`.

```
year_weeknum_weekday& year_weeknum_weekday::operator-=(const years& y) noexcept;
```

Effectx: `*this = *this - y`.

Returns: `*this`.

```
constexpr iso_week::year year_weeknum_weekday::year() const noexcept;
```

Returns: `y_`.

```
constexpr iso_week::weeknum year_weeknum_weekday::weeknum() const noexcept;
```

Returns: `wn_`.

```
constexpr iso_week::weekday year_weeknum_weekday::weekday() const noexcept;
```

Returns: `wd_`.

```
constexpr year_weeknum_weekday::operator sys_days() const noexcept;
```

Returns: A `sys_days` which represents the date represented by `*this`.

```
constexpr bool year_weeknum_weekday::ok() const noexcept;
```

Returns: `y_.ok() &&
wd_.ok() &&
1_w <= wn_ && wn_ <= year_lastweek{y_}.weeknum()`.

```
constexpr bool operator==(const year_weeknum_weekday& x, const year_weeknum_weekday& y) noexcept;
```

Returns: `x.year() == y.year() &&
x.weeknum() == y.weeknum() &&
x.weekday() == y.weekday()`.

```
constexpr bool operator!=(const year_weeknum_weekday& x, const year_weeknum_weekday& y) noexcept;
```

Returns: `!(x == y)`.

```
constexpr bool operator<(const year_weeknum_weekday& x, const year_weeknum_weekday& y) noexcept;
```

Returns: `x.year() < y.year() ? true
: (x.year() > y.year() ? false
: (x.weeknum() < y.weeknum() ? true
: (x.weeknum() > y.weeknum() ? false
: (unsigned{x.weekday()} < unsigned{y.weekday()}))))`.

```
constexpr bool operator>(const year_weeknum_weekday& x, const year_weeknum_weekday& y) noexcept;
```

Returns: `y < x`.

```
constexpr bool operator<=(const year_weeknum_weekday& x, const year_weeknum_weekday& y) noexcept;
```

Returns: `!(y < x)`.

```
constexpr bool operator>=(const year_weeknum_weekday& x, const year_weeknum_weekday& y) noexcept;
```

Returns: `!(x < y)`.

```
constexpr year_weeknum_weekday operator+(const year_weeknum_weekday& x, const years& y) noexcept;
```

Returns: `(x.year() + y) / x.weeknum() / x.weekday()`.

```
constexpr year_weeknum_weekday operator+(const years& y, const year_weeknum_weekday& x) noexcept;
```

Returns: `x + y`.

```
constexpr year_weeknum_weekday operator-(const year_weeknum_weekday& x, const years& y) noexcept;
```

Returns: `x + -y`.

```
std::ostream& operator<<(std::ostream& os, const year_weeknum_weekday& x);
```

Effects: `os << x.year() << '-' << x.weeknum() << '-' << x.weekday()`.

Returns: `os`.

year_lastweek_weekday

Synopsis

```
class year_lastweek_weekday
{
    iso_week::year    y_;   // exposition only
    iso_week::weekday wd_;  // exposition only

public:
    constexpr year_lastweek_weekday(const iso_week::year& y,
                                    const iso_week::weekday& wd) noexcept;

    year_lastweek_weekday& operator+=(const years& y) noexcept;
    year_lastweek_weekday& operator-=(const years& y) noexcept;

    constexpr iso_week::year    year()    const noexcept;
    constexpr iso_week::weeknum weeknum() const noexcept;
    constexpr iso_week::weekday weekday() const noexcept;

    constexpr operator sys_days() const noexcept;
    constexpr bool ok() const noexcept;
```

```
};

constexpr bool operator==(const year_lastweek_weekday& x, const year_lastweek_weekday& y) noexcept;
constexpr bool operator!=(const year_lastweek_weekday& x, const year_lastweek_weekday& y) noexcept;
constexpr bool operator< (const year_lastweek_weekday& x, const year_lastweek_weekday& y) noexcept;
constexpr bool operator> (const year_lastweek_weekday& x, const year_lastweek_weekday& y) noexcept;
constexpr bool operator<=(const year_lastweek_weekday& x, const year_lastweek_weekday& y) noexcept;
constexpr bool operator>=(const year_lastweek_weekday& x, const year_lastweek_weekday& y) noexcept;

constexpr year_lastweek_weekday operator+(const year_lastweek_weekday& x, const years& y) noexcept;
constexpr year_lastweek_weekday operator+(const years& y, const year_lastweek_weekday& x) noexcept;
constexpr year_lastweek_weekday operator-(const year_lastweek_weekday& x, const years& y) noexcept;

std::ostream& operator<<(std::ostream& os, const year_lastweek_weekday& x);
```

Overview

`year_lastweek_weekday` represents a `year`, and `weekday` in the last week of the [ISO week date calendar](). One can observe each field. `year_lastweek_weekday` supports year-oriented arithmetic. There is an implicit conversion to `sys_days`. `year_lastweek_weekday` is equality and less-than comparable.

Specification

`year_lastweek_weekday` is a trivially copyable class type.
`year_lastweek_weekday` is a standard-layout class type.
`year_lastweek_weekday` is a literal class type.

```
constexpr year_lastweek_weekday::year_lastweek_weekday(const iso_week::year& y,
                                                       const iso_week::weekday& wd) noexcept;
```

Effects: Constructs an object of type `year_lastweek_weekday` by constructing `y_` with `y`, and `wd_` with `wd`.

```
year_lastweek_weekday& year_lastweek_weekday::operator+=(const years& y) noexcept;
```

Effectx: `*this = *this + y`.

Returns: `*this`.

```
year_lastweek_weekday& year_lastweek_weekday::operator-=(const years& y) noexcept;
```

Effectx: `*this = *this - y`.

Returns: `*this`.

```
constexpr iso_week::year year_lastweek_weekday::year() const noexcept;
```

Returns: `y_`.

```
constexpr iso_week::weeknum year_lastweek_weekday::weeknum() const noexcept;
```

Returns: `(y_ / last).weeknum()`.

```
constexpr iso_week::weekday year_lastweek_weekday::weekday() const noexcept;
```

Returns: `wd_`.

```
constexpr year_lastweek_weekday::operator sys_days() const noexcept;
```

Returns: A `sys_days` which represents the date represented by `*this`.

```
constexpr bool year_lastweek_weekday::ok() const noexcept;
```

Returns: `y_.ok() && wd_.ok()`.

```
constexpr bool operator==(const year_lastweek_weekday& x, const year_lastweek_weekday& y) noexcept;
```

Returns: `x.year() == y.year() && x.weekday() == y.weekday()`.

```
constexpr bool operator!=(const year_lastweek_weekday& x, const year_lastweek_weekday& y) noexcept;
```

Returns: `!(x == y)`.

```
constexpr bool operator<(const year_lastweek_weekday& x, const year_lastweek_weekday& y) noexcept;
```

Returns: `x.year() < y.year() ? true : (x.year() > y.year() ? false : (unsigned{x.weekday()} < unsigned{y.weekday()}))`.

```
constexpr bool operator>(const year_lastweek_weekday& x, const year_lastweek_weekday& y) noexcept;
```

Returns: `y < x`.

```
constexpr bool operator<=(const year_lastweek_weekday& x, const year_lastweek_weekday& y) noexcept;
```

Returns: `!(y < x)`.

```
constexpr bool operator>=(const year_lastweek_weekday& x, const year_lastweek_weekday& y) noexcept;
```

Returns: `!(x < y)`.

```
constexpr year_lastweek_weekday operator+(const year_lastweek_weekday& x, const years& y) noexcept;
```

Returns: `(x.year() + y) / last / x.weekday()`.

```
constexpr year_lastweek_weekday operator+(const years& y, const year_lastweek_weekday& x) noexcept;
```

Returns: `x + y`.

```
constexpr year_lastweek_weekday operator-(const year_lastweek_weekday& x, const years& y) noexcept;
```

Returns: `x + -y`.

```
std::ostream& operator<<(std::ostream& os, const year_lastweek_weekday& x);
```

Effects: `os << x.year() << "-W last-" << x.weekday()`.

Returns: `os`.

date composition operators

To understand this API it is not necessary for you to memorize each of these operators. Indeed, that would be detrimental to understanding this API. Instead it is sufficient to known that this collection of operators implement constructions in 3 orders:

1. `year/weeknum/weekday`
2. `weeknum/weekday/year`
3. `weekday/weeknum/year`

The first component in each order must be properly typed, the following components may be specified with the proper type or an `int`.

Anywhere a "weeknum" is required one can also specify `last` to indicate the last week of the year.

Sub-field-types such as `year_weeknum` and `weeknum_weekday` can be created by simply not applying the second division operator for any of the three orders. For example:

```
year_weeknum ym = 2015_y/52_w;
weeknum_weekday md1 = 52_w/thu;
weeknum_weekday md2 = thu/52_w;
```

Everything not intended as above is caught as a compile-time error, with the notable exception of an expression that consists of nothing but `int`, which of course has type `int`.

```
auto a = 2015/4/4;       // a == int(125)
auto b = 2015_y/4/4;     // b == year_weeknum_weekday{year(2015), week(4), weekday(4)}
auto c = 2015_y/thu/4_w; // error: invalid operands to binary expression ('iso_week::year' and 'iso_week::weekday')
auto d = 2015/4_w/4;     // error: invalid operands to binary expression ('int' and 'const iso_week::weeknum')
```

The last example may be clear to a human reader. But the compiler doesn't know if `2015` refers to a `year` or a `weekday`. Instead of guessing, the compiler flags it as an error.

In short, you will either write unambiguous and readable code, or you will get a compile-time error.

`year_weeknum`:

```
constexpr year_weeknum operator/(const year& y, const weeknum& wn) noexcept;
```

Returns: {y, wn}.

constexpr year_weeknum operator/(const year& y, int wn) noexcept;

Returns: y / weeknum(wn).

year_lastweek:

constexpr year_lastweek operator/(const year& y, last_week) noexcept;

Returns: year_lastweek{y}.

weeknum_weekday:

constexpr weeknum_weekday operator/(const weeknum& wn, const weekday& wd) noexcept;

Returns: {wn, wd}.

constexpr weeknum_weekday operator/(const weeknum& wn, int wd) noexcept;

Returns: wn / weekday{static_cast<unsigned>(wd)}.

constexpr weeknum_weekday operator/(const weekday& wd, const weeknum& wn) noexcept;

Returns: wn / wd.

constexpr weeknum_weekday operator/(const weekday& wd, int wn) noexcept;

Returns: weeknum{static_cast<unsigned>(wn)} / wd.

lastweek_weekday:

constexpr lastweek_weekday operator/(const last_week&, const weekday& wd) noexcept;

Returns: lastweek_weekday{wd}.

constexpr lastweek_weekday operator/(const last_week& wn, int wd) noexcept;

Returns: wn / weekday{static_cast<unsigned>(wd)}.

constexpr lastweek_weekday operator/(const weekday& wd, const last_week& wn) noexcept;

Returns: wn / wd.

year_weeknum_weekday:

constexpr year_weeknum_weekday operator/(const year_weeknum& ywn, const weekday& wd) noexcept;

Returns: {ywn.year(), ywn.weeknum(), wd}.

constexpr year_weeknum_weekday operator/(const year_weeknum& ywn, int wd) noexcept;

Returns: ywn / weekday{static_cast<unsigned>(wd)}.

constexpr year_weeknum_weekday operator/(const weeknum_weekday& wnwd, const year& y) noexcept;

Returns: {y, wnwd.weeknum(), wnwd.weekday()}.

constexpr year_weeknum_weekday operator/(const weeknum_weekday& wnwd, int y) noexcept;

Returns: wnwd / year{y}.

year_lastweek_weekday:

constexpr year_lastweek_weekday operator/(const year_lastweek& ylw, const weekday& wd) noexcept;

Returns: {ylw.year(), wd}.

constexpr year_lastweek_weekday operator/(const year_lastweek& ylw, int wd) noexcept;

Returns: ylw / weekday{static_cast<unsigned>(wd)}.

constexpr year_lastweek_weekday operator/(const lastweek_weekday& lwwd, const year& y) noexcept;

Returns: `{y, lwwd.weekday()}`.

```
constexpr year_lastweek_weekday operator/(const lastweek_weekday& lwwd, int y) noexcept;
```

Returns: `lwwd / year{y}`.