Calendrical C++

`std::chrono`, History, Mathematics, and the Computus

Ben Deane, 8th May 2023
(The moon is 18 days old today)

Disclaimers

No AI was used in the preparation of this talk. Any mistakes are mine.

This talk contains plenty of C++, but is not primarily *about* C++. Now is your chance to leave and go see Sean or Zach talk about "real" C++ instead. I won't be offended.

If you like this talk, go see Cassio Neri's talk on Wednesday after lunch for more calendar geekery.

If you don't like this talk, go see Cassio Neri's talk on Wednesday after lunch; maybe it will be better.

Things I am interested in

1. Everything

But especially today:

2. Time & Calendars

Also:

3. Heraldry (a different talk)

In 1996

I was playing a lot of *Nethack* ("You are lucky! Full moon tonight.")

I was keeping a phase-of-the-moon widget in my Win 95 system tray.

I was (still am) friends with the lead programmer on *Dungeon Keeper*.

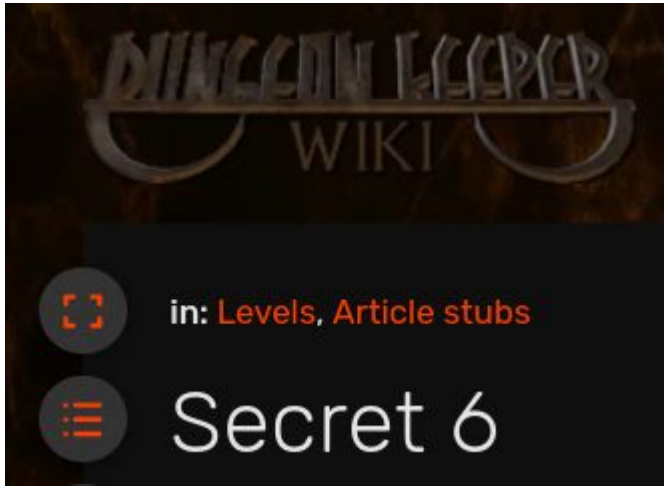
I owned (still own) a copy of *Numerical Recipes in C*.

In 1996

"You aren't really intended to understand this algorithm, but it does work!"

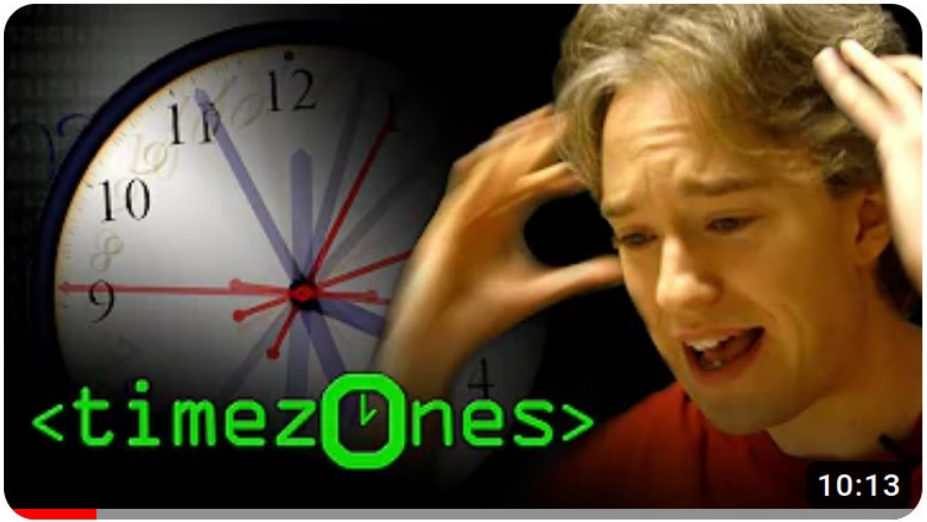
```
void flmoon(int n, int nph, long *jd, float *frac) {
  int i;
  float am,as,c,t,t2,xtra;

  c=n+nph/4.0;
  t=c/1236.85;
  t2=t*t;
  as=359.2242+29.105356*c;
  am=306.0253+385.816918*c+0.010730*t2;
  *jd=2415020+28L*n+7L*nph;
  xtra=0.75933+1.53058868*c+((1.178e-4)-(1.55e-t)*t)*t2;
  if (nph == 0 || nph == 2)
    xtra += (0.1734-3.93e-4*t)*sin(RAD*as)-0.4068*sin(RAD*am);
  else if (nph == 1 || nph == 3)
    xtra += (0.1721-4.0e-4*t)*sin(RAD*as)-0.6280*sin(RAD*am);
  i=(int)(xtra >= 0.0 ? floor(xtra) : ceil(xtra-1.0));
  *jd += i;
  *frac=xtra-i;
}
```

The result



This realm appears on the main map screen on the day of a full moon. These days are determined using the system clock. For a list of full moon dates, see Current events.

Let's talk a bit about time



The Problem with Time & Timezones - Computerphile

3.5M views • 9 years ago

Computerphile

A web app that works out how many seconds ago something happened. How hard can coding that be? **Tom Scott** explains...

Subtitles

If you haven't seen this video, go watch it at your earliest convenience! https://www.youtube.com/watch?v=-5wpm-gesOY

A Quick Introduction to `std::chrono`

std::chrono::duration

Defined in header <chrono>

```
template<
    class Rep,
    class Period = std::ratio<1>
> class duration;
```

(since C++11)

Helper types

Type	Definition
std::chrono::nanoseconds	`duration</*signed integer type of at least 64 bits*/, std::nano>`
std::chrono::microseconds	`duration</*signed integer type of at least 55 bits*/, std::micro>`
std::chrono::milliseconds	`duration</*signed integer type of at least 45 bits*/, std::milli>`
std::chrono::seconds	`duration</*signed integer type of at least 35 bits*/>`
std::chrono::minutes	`duration</*signed integer type of at least 29 bits*/, std::ratio<60>>`
std::chrono::hours	`duration</*signed integer type of at least 23 bits*/, std::ratio<3600>>`
`std::chrono::days` (since C++20)	`duration</*signed integer type of at least 25 bits*/, std::ratio<86400>>`
`std::chrono::weeks` (since C++20)	`duration</*signed integer type of at least 22 bits*/, std::ratio<604800>>`
`std::chrono::months` (since C++20)	`duration</*signed integer type of at least 20 bits*/, std::ratio<2629746>>`
`std::chrono::years` (since C++20)	`duration</*signed integer type of at least 17 bits*/, std::ratio<31556952>>`

Note: each of the predefined duration types up to `hours` covers a range of at least ±292 years.

Each of the predefined duration types `days`, `weeks`, `months` and `years` covers a range of at least ±40000 years. `years` is equal to 365.2425 days (the average length of a Gregorian year). `months` is equal to 30.436875 days (exactly 1/12 of `years`). (since C++20)

A Quick Introduction to `std::chrono`

std::chrono::time_point

Defined in header <chrono>

```
template<
    class Clock,
    class Duration = typename Clock::duration
> class time_point;
```

(since C++11)

`std::chrono` usage 101

- Make conversions happen at interface (or IO) boundaries
- Safe (no information loss) conversion is implicit
- Treat durations homogeneously, don't worry too much about ratio
- Eliminate classes of bugs
  - Never again accidentally pass untyped milliseconds to a function expecting seconds

Most importantly:

- Stop thinking in terms of seconds/milliseconds/microseconds!
  - Maybe define your own clocks/durations

Make your own clock?

```
struct my_clock {
  using rep = ...;
  using period = std::ratio<...>;
  using duration = std::chrono::duration<rep, period>;
  using time_point = std::chrono::time_point<my_clock, duration>;
  constexpr static bool is_steady = ...;
  auto now() /*noexcept?*/ -> time_point;
};
static_assert(std::is_clock_v<my_clock>);
```

`std::chrono` usage examples

When you're writing a networked game, it's important for the client and server to agree when things happen.

When a client connects to a server, it can synchronize a clock.

What is synchronization? It's setting the epoch of the `server_clock` on the client.

Stop thinking in {milli,micro}seconds

You're programming some embedded stuff.

You have a crystal RTC that ticks at 32,768Hz.

```
constexpr auto ticks_to_us(auto ticks) {
  return ticks * 1'000'000 / 32'768;
}
```

Stop thinking in {milli,micro}seconds

You're programming some embedded stuff.

You have a crystal RTC that ticks at 32,768Hz.

```
using rtc_tick = std::chrono::duration<
    std::int64_t, std::ratio<1, 32'768>>;
```

Make your own units

```
struct user {
  system_clock::time_point afk;
};

auto afk_display(user u) -> std::string {
  using five_minutes = duration<system_clock::rep,
                                std::ratio<5 * 60>>;
  const minutes afk_period =
      duration_cast<five_minutes>(system_clock::now() - u.afk);
  return std::format("AFK for {}", afk_period);
}
```

Make your own units

```
using rep_t = std::chrono::system_clock::duration::rep;

using season_of_love = std::chrono::duration<
    rep_t, std::ratio<525'600 * 60>>;

using ntsc_frame = std::chrono::duration<
    rep_t, std::ratio<100, 29'97>>;

using medication_period = std::chrono::duration<
    rep_t, std::ratio<6 * 60 * 60>>;
```

One more `std::chrono` tip

Don't use `high_resolution_clock`.

It sounds great, but it's under-specified.

If you want human "wall" time, use `system_clock`.

If you want a stopwatch, use `steady_clock`.

For guaranteed high resolution, make your own (platform-specific) clock.

Other chrono clocks (C++20)

- `utc_clock`
  - epoch 1970/January/1
  - includes leap seconds
  - `to_sys`/`from_sys` conversions
- `tai_clock`
  - epoch 1958/January/1
  - does *not* include leap seconds
  - currently 37 seconds ahead of UTC
- `gps_clock`
  - epoch 1980/January/6
  - always 19 seconds behind `tai_clock`
- `file_clock`
- `local_t` (pseudo-clock)

Caveat around now()

"[Calling clock.now()] returns a time_point object representing the current point in time." - [time.clock.req]

But it is unclear what "the current point in time" means in the presence of reordering.

See P0342.

Now let's talk about dates

Important date types in `std::chrono`

`day` (`unsigned char`-ish)

`month` (`enum`-ish, `January` = 1, modulo arithmetic)

`year` (`int`-ish)

`weekday` (`enum`-ish, `Sunday` = 0, modulo arithmetic)

`year_month_day` and `year_month_weekday`

`sys_days` (a `system_clock::time_point` with `days` `duration`)

Easy construction

```
constexpr auto today = day{8}/May/2023;
constexpr auto today = May/8/2023;
constexpr auto today = year{2023}/May/8;
```

Constructors of {`year`, `month`, `day`} are all explicit. (There are also UDLs.)

But: `operator/` is overloaded with `int`.

You can do any "sensible" orderings... probably stick to one convention.

Historical Diversion

I must sideline C++ for a while.

Because calendars are strange, and there is some fascinating history and mathematics to cover. Much, even most, mathematical progress before about the 1600s was driven by figuring out when Easter falls.

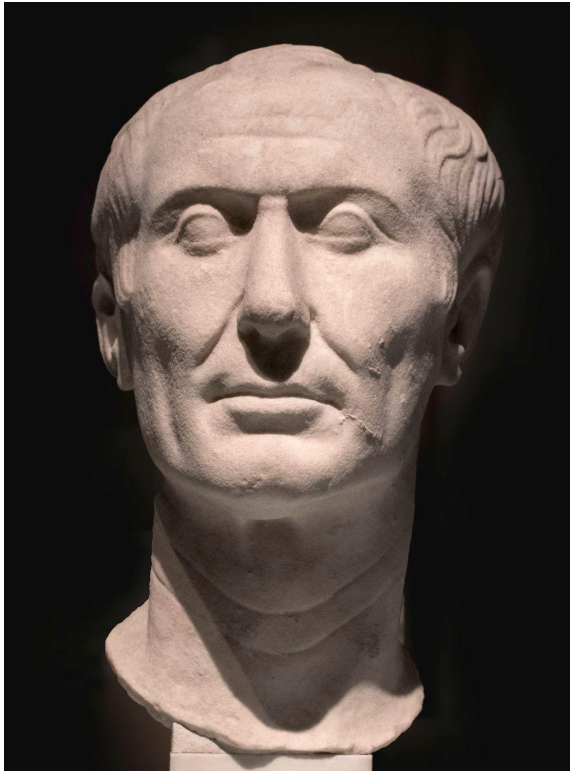
We'll get back to expressing things in C++ later.

46 BCE: the story begins

The Roman republic has been in a civil war with Julius Caesar since 49 BCE (when he crossed the Rubicon).

JC returns to Rome claiming victory from campaigns in the wider Roman pre-empire (Gaul, Egypt, Asia Minor, North Africa).

The Roman senate makes a gamble: they give him dictatorship for 10 years and consulship for 5.



The Roman Calendar in a shambles

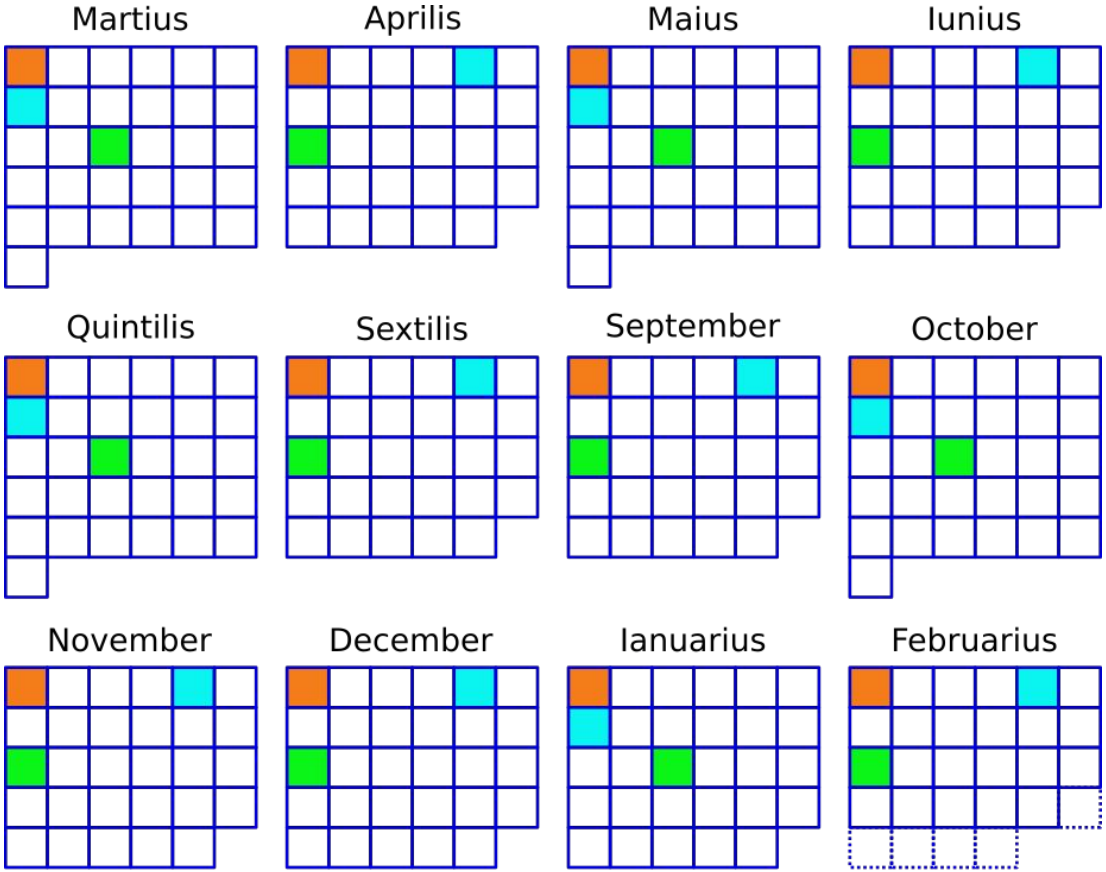
46 BCE was the year 708 AUC in the Roman Republic calendar.

Like many prehistorical calendars, it had evolved from a lunar calendar.

Basically it had alternating* short (29) and long (31) months, starting in March and ending in February.

A year was 355 days long and it was the job of the Pontifex Maximus to decree the insertion of intercalary days and/or months. Who was PM? Julius Caesar. He'd been busy... the calendar was about 3 months adrift.

Roman Republic (pre-Julian) Calendar



A few quick facts about the Roman Calendar

Yes they had 12 months: Numa Pompilius added January & February around 700 BCE ("legend": Rome was sacked in 390 BCE and all records destroyed).

The year began in March and ended in February (355 days).

When required, February would be shortened and an intercalary month added for 377/378 days.

Quintilis became Iulius in 44 BCE (after JC's assassination).

Sextilis became Augustus in 8 BCE (in the reign of Augustus).

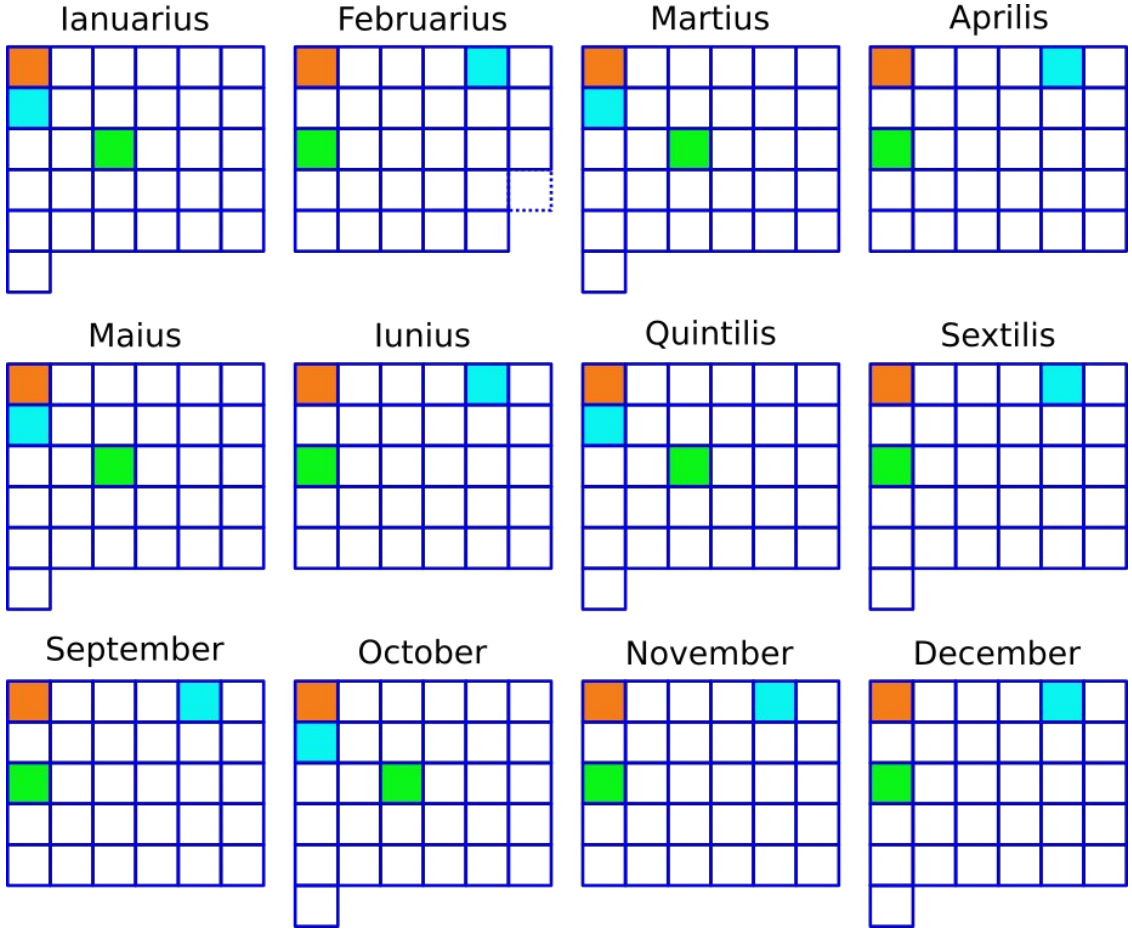
Julian Calendar reform

46 BCE: the longest year in human history at 445 days (trivia question) - JC added 90 intercalary days to do a reset.

Sosigenes of Alexandria helped with the new calendar, which was to do away with human foibles and be the first fixed calendar, amenable to pure calculation. (Based on the Egyptian calendar of 30-day months.)

We basically still use it today, with a minor correction from the middle ages to make the Gregorian calendar.

The Julian Calendar



Kalends, Nones, Ides

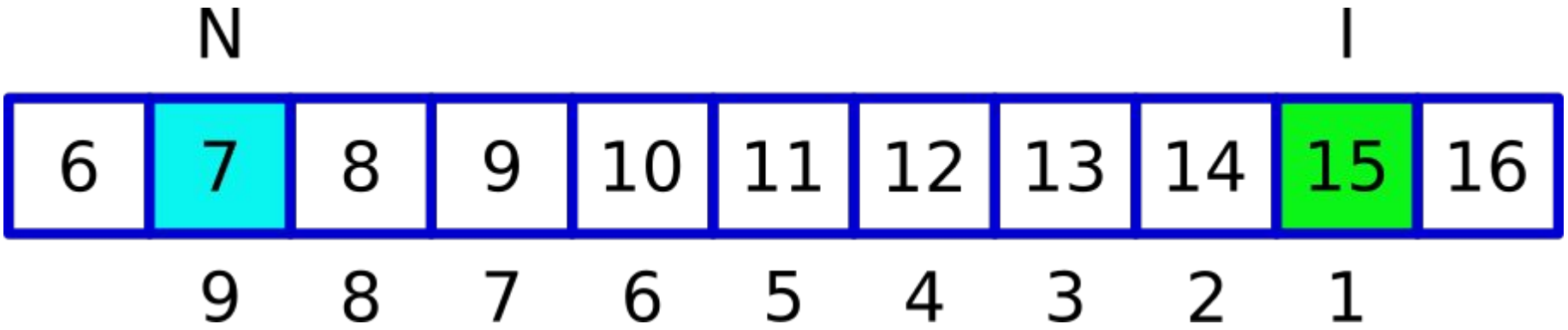
The Romans specified dates as the number of days before (counting inclusively!) the next "important day" (Kalends, Nones or Ides).

e.g. the Ides of March = March 15th; the Nones of March = March 7th.

Nones are nine days (remember count inclusively) before Ides.

"In March, July, October, May, the Ides fall on the 15th day, the Nones the 7th. The rest besides, take two days from Nones and Ides."

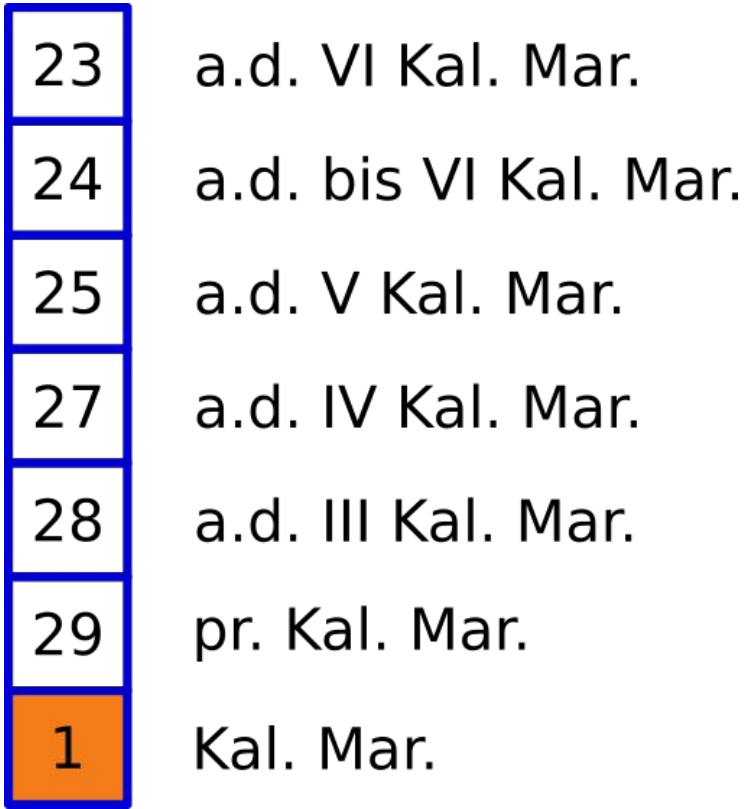
Figuring the date in Roman



Saying the date in Roman

11	a.d. V Id. Mar.
12	a.d. IV Id. Mar.
13	a.d. III Id. Mar.
14	pr. Id. Mar.
15	Id. Mar.
16	a.d. XVII Kal. Apr.

Roman leap years



Romans counted inclusively

So at first they screwed up leap years!

- JC said every 4th year
- They understood a 3-year cycle (inclusive counting, remember?)

Augustus fixed it by omitting leap years after 9 BCE (a leap year) and before 8 CE (the next).

All this is… cumbersome

`std::chrono` is Gregorian-calendar-centric.

Arithmetic is possible on `year_month_day` (with appropriate conversion to/from `sys_days`).

To work up a full Julian calendar system requires computation with `sys_days` and different leap year calculations.

Not *difficult* per se, but very fiddly… and we haven't even talked (and we won't) about the nundinal cycle.

The Computus

or, how to calculate Easter

Time to talk about Easter

You're a (very) early Christian. How do you know when to celebrate Easter?

You know that Jesus rose on Sunday after the first night of Passover. And maybe you can ask your local rabbi when Passover is.

But after a few hundred years, the Roman empire is large, and reaches far-flung parts of Europe...

Oh noes, people might be celebrating Easter on the wrong day!

The correct date for Easter is important

In the outer reaches and dying days of the western Roman empire, life is threatened (perhaps daily) by the likes of:

- Pale people with red hair who will burn your village
- Pale people with big sticks who will smash up your stuff
- Pale people with black lipstick who will read you depressing poems

To say nothing of failed crops, famine, pestilence in a pre-medical-science society, and the occupational injuries of a life of manual labour.

It's important to safeguard your immortal soul!

"O, swear not by the moon, th'inconstant moon,
That monthly changes in her circled orb,
Lest that thy love prove likewise variable."

Romeo and Juliet, Act 2 Scene 2

(At this point, we acknowledge the existence of the moon)

Problems Any Calendar Must Solve

You can count suns, and/or you can count moons. Counting moons makes sense for farming/fishing societies, so many ancient calendars are lunar (or lunisolar).

But really, you need to count days... and the essential problem (of course) is that the times taken by:

- the earth to revolve around its axis
- the earth to make a trip around the sun
- the moon to make a trip around the earth

...are all basically incommensurate.

The (First) Council of Nicaea, 325 CE

We're going to *define* Easter now, and stop relying on Passover. Easter is:

- the first Sunday after*
- the first full moon on or after
- the vernal equinox

Oh, and the vernal equinox is on March 21st. (We don't care about astronomy so much.)

*or the Sunday after the first Saturday on or after

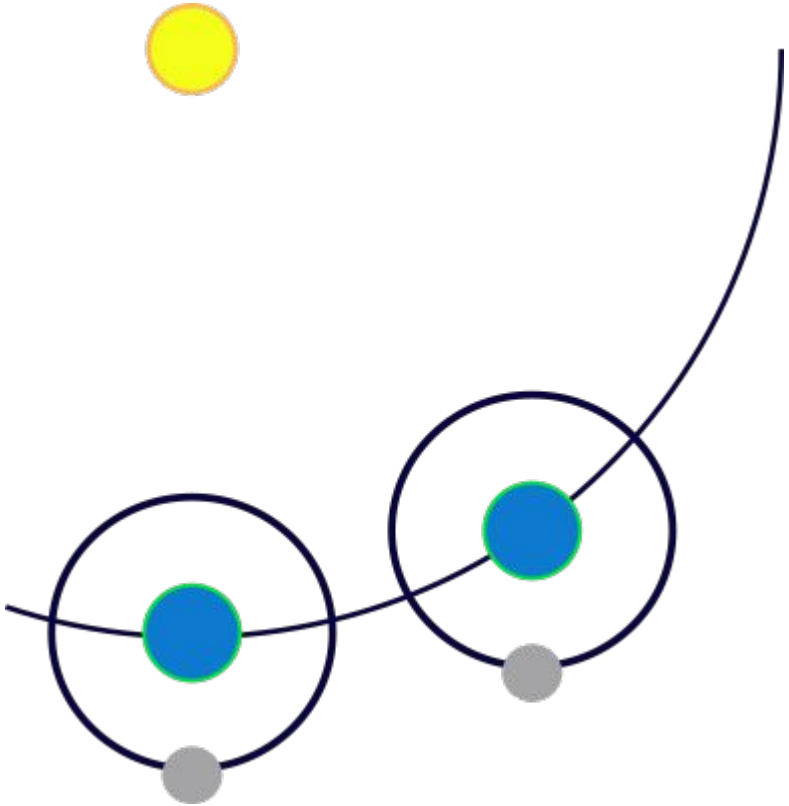
Period of the moon

Sidereal period ≈ 27.3 days

(Latin *sidus* = star)

Synodic period ≈ 29.5 days

(Greek *sunodos* = conjunction)



The Early Christian Church has a go

Theophilus of Alexandria produces Paschal tables around 390 CE.

Cyril of Alexandria extrapolates from this and produces Paschal tables around 440 CE.

Dionysius Exiguus takes it further, producing Paschal tables from 525 CE. He also fixes year 1 CE (753 AUC). (Wrongly, as it happens.)

Aside: the Synod of Whitby, 664 CE

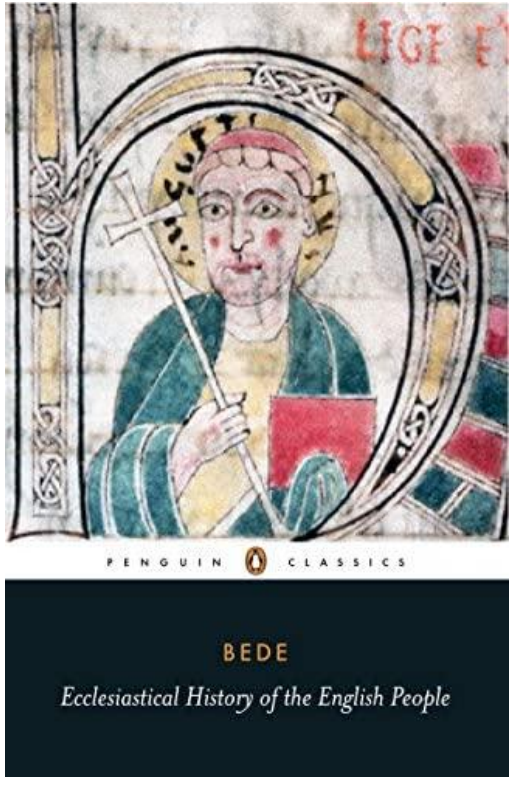
Problem: King Oswig and Queen Eanflæd were celebrating Easter a week apart! Which was right?

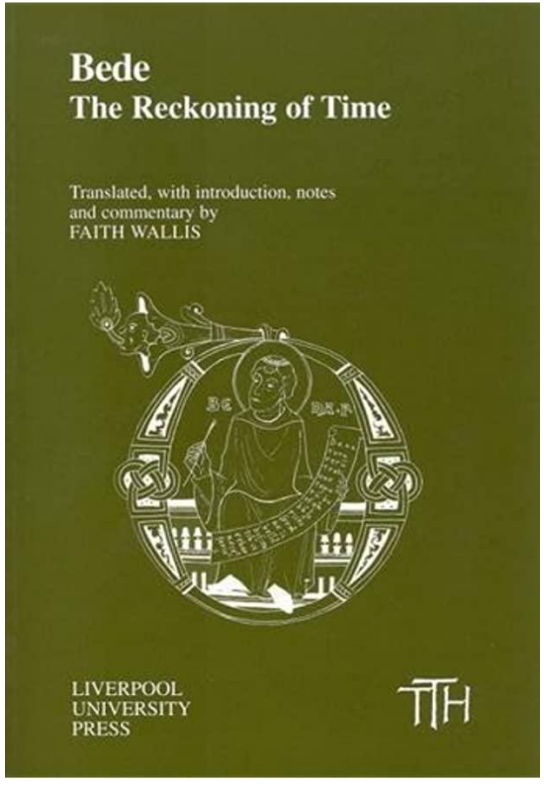
The Ionian way (Bishop Colmán & St Columba)?

Or the Roman/Alexandrian way (Wilfrið)?

Oswig ruled Wilfrið the winner (despite a ludicrous argument) and the Roman way was settled.

Bede


PENGUIN CLASSICS
BEDE
Ecclesiastical History of the English People


Bede
The Reckoning of Time
Translated, with introduction, notes
and commentary by
FAITH WALLIS
LIVERPOOL
UNIVERSITY
PRESS
TTH

Bede (725 CE) Understood Things

Although pious, he was a scientist. He observed things accurately. And he was quite snarky to a) non-Christians and b) his contemporaries & predecessors who claimed things that were manifestly untrue.

He produced a 532-year cycle (19 * 28) of tables, and finally an accurate method that was adopted by the whole church, and stood until the Gregorian calendar reforms...

He also noticed that the Julian calendar was already slipping.

The Metonic Cycle

19 tropical years ≈ 235 synodic months

12 synodic months ≈ 354 days

Every year, the moon falls short by about 11 days, or (lunar) ***epacts***.

Year	1	2	3	4	5	6	7	8	9	10	11
Epacts	11	22	3	14	25	6	17	28	9	20	1
Year	12	13	14	15	16	17	18	19	20	21	22
Epacts	12	23	4	15	26	7	18	30	11	22	3

Thus the age of the moon on 1st Jan is the previous year's epacts.

Metonic Cycle Terms

In the 19 year cycle, some years are ***common*** (12 lunations) and some are ***embolismic*** (13 lunations). The extra intercalated month is the ***embolism***.

Some (lunar) months are ***hollow*** (29 days) and some are ***full*** (30 days).

Year 19 has an extra ***epact***: the ***saltus lunae*** or leap of the moon to resynchronise the solar calendar. Added to either the July lunation or the final lunation (Bede) of year 19.

The first 8 years of the M.C. are the ***ogdoas*** (pattern CCECCECE).

The final 11 years of the M.C. are the ***hendecas*** (pattern CCECCECCECE).

The current cycle year is often called the ***Golden number***.

Fitting Weeks to Years

The Metonic cycle tells us about the moon.

A year is not an integral number of weeks, so the other part of the calculation is to figure out the equivalent solar epacts or ***concurrents*** for the current year.

This is fairly easy: each non-leap year has 1 concurrent, and leap years have 2 (i.e. 365 % 7 = 1 and 366 % 7 = 2).

And once again we accumulate concurrents over the 28-year solar cycle.

Relatedly, some calculations use the ***Dominical letter*** (A-G).

To Calculate Easter

1. Figure out which year of the Metonic cycle you're in. This tells you where you are in the lunar calendar.
2. In certain cycle years, the 14th Paschal moon occurs in March, in other years it's in April. You can figure this out, but it's easy to just remember a small table.
3. Find the number of epacts for the current year. Subtract this from 36 (March) or 35 (April) and to give the date of the 14th Paschal moon.*
4. Find the number of concurrents for the current year. Add this to the date, add 4 (if in March) and mod by 7. This is the weekday of the 14th Paschal moon. (1 = Sunday)

A bit of C++!

```
constexpr auto metonic_year(int year) -> int {
  return (year + 1) % 19;
}

constexpr auto in_march(int metonic_year) -> bool {
  return std::string{2, 5, 7, 10, 13, 16, 18}.contains(metonic_year);
}

constexpr auto epacts(int year) -> int {
  return ((year % 19) * 11) % 30;
}

constexpr auto concurrents(int year) -> int {
  return (year + (year / 4) + 4) % 7;
}
```

More C++

```
auto paschal_moon_date(int year)
    -> std::pair<std::chrono::year_month_day, std::chrono::weekday> {
  const auto my = metonic_year(year);
  const auto month = in_march(my) ? std::chrono::March : std::chrono::April;

  const auto adjustment = in_march(my) ? 1 : 0;
  const auto day = (35 + adjustment - epacts(year)) % 30;

  const auto weekday_adjustment = in_march(my) ? 4 : 0;
  const unsigned weekday = (day + concurrents(year) + weekday_adjustment) % 7;

  return {month / day / year, --std::chrono::weekday{weekday}};
}
```

Easter Sunday

```
constexpr auto easter_sunday(int year) -> std::chrono::year_month_day {
  const auto [moon_date, weekday] = paschal_moon_date(year);
  const auto to_next_sunday = ++(std::chrono::Saturday - weekday);
  return static_cast<std::chrono::sys_days>(moon_date) + to_next_sunday;
}
```

Easter Sunday 2023 is on 3rd April? What?

Google says Easter is April 9th...

Before we leave Bede… miscellany

The Julian day starts at noon.

Leap day is Feb 24th, not Feb 29th? Romans: festival of Terminalia on the last day of the year (a.d. VII Kal. Mar.)

When is New Year's Day?

- 1st January (JC & popular)
- 25th March? (Lady Day - from 1155 [Bond] to 1752 in England)
- 25th December?
- Start of Advent (liturgical)?

Genesis Chapter 1: God's diary

Day 1: Made light. Divided dark from light. [equinox: Mar 21]

Day 2: Separated heaven from earth.

Day 3: Made dry land & plants.

Day 4: Made sun and moon to keep day and night. Days, seasons, years. [1st dawn: Mar 25]

Day 5: Made birds & sea creatures.

Day 6: Made land animals & people.

And at this point, like everyone, God got bored of keeping a diary.

Bede Tells us the Old English Months

Æfterra Geola (after yule, Jan)

Sol-monaþ (hearthcakes, Feb)

Hreð-monaþ (Hreða, Mar)

Eostre-monaþ (Eostre, Apr)

Þrimilce (Milk cows 3x day, May)

Ærra Liþa (smooth sailing, Jun)

Æfterra Liþa (Jul)

Þeod-monaþ (tares [weeds], Aug)

Halig-monaþ (holy rites, Sep)

Þintirfylleþ (winter full [moon], Oct)

Blod-monaþ (sacrifice, Nov)

Ærra Geola (before yule, Dec)

A wild Pope Gregory XIII appears!

1582: The World* listens to Bede

*Catholic countries in Europe anyway

The Julian calendar is now about 10 days adrift.

Pope Gregory XIII institutes reform: century-years are only leap years if they are divisible by 400. (Accurate enough and expedient: the first deviation from the Julian calendar is comfortably distant.)

Calendar adjusted by 10 days. Note: weekdays *not* affected - Thu 4th Oct followed by Fri 15th Oct.

Aloysius Lilius & Christopher Clavius devise the new Easter algorithm.

The Lilius/Clavius Algorithm

```
constexpr auto easter_sunday(int year) -> std::chrono::year_month_day {
  const auto A = year % 19 + 1;            // metonic cycle year
  const auto B = year / 100 + 1;           // century
  const auto C = (3 * B) / 4 - 12;         // J-G leap year correction
  const auto D = (8 * B + 5) / 25 - 5;     // metonic cycle correction
  const auto E = (year * 5) / 4 - 10 - C; // March 21 weekday
  auto F = ((11 * A + 20 + D - C) % 30 + 30) % 30;  // epact
  if (F == 24 or (F == 25 and A > 11)) { ++F; }
  auto G = 44 - F;                         // full moon after March 21
  if (G < 21) { G += 30; }

  const auto base = std::chrono::March / 1 / year;
  const auto offset = std::chrono::days{G + 7 - (E + G) % 7 - 1};
  return static_cast<std::chrono::sys_days>(base) + offset;
}
```

16th-18th Century: a Messy Time

Various countries adopt the Gregorian calendar at various times.

Astronomy is now a Real Science and weighs in on astronomical Easter.

Protestants, Lutherans etc have a problem with papal authority.

Some Christians erroneously think Easter *must not* coincide with Passover.

In Britain, Elizabeth I is progressive enough to see merits in the reform, but her bishops (newly minted C of E) want no part of it.

Switching to the New Calendar

In Britain this happened in the early 1750s.

1751 started on March 25th and ended on Dec 31st*. It was therefore the shortest year, at 282 days.

(1752 was the second shortest year: 355 days.)

Reports of rioting (over the calendar change) are overstated; however there were legitimate concerns. The act therefore distinguishes "nominal days" from "natural days".

Switching to the New Calendar

In some countries (Eastern/Southern Europe) this didn't happen until the 20th century!

In May 1923, some Eastern churches agreed reforms:

1. Omit first 13 days of October 1923 (spottily implemented).
2. Centennial leap years have a remainder of 200 or 600 when divided by 900 (more accurate than the Gregorian method, but won't diverge until 2800).
3. Change the way Easter is calculated (rejected).

Some Orthodox churches still use the Julian calendar today, e.g. Christmas in Russia is commonly celebrated on 7th Jan.

Dates that do not exist

Because of the British Calendar act of 1751, the following dates do not exist in England:

1st Jan 1751 - 24th March 1751 (elided by New Year's Day moving).

3rd September 1752 - 13th September 1752 (elided by Julian-Gregorian calendar correction - now 11 days).

Dates that exist multiple times

In Scotland: James VI decreed in 1600 that New Year's Day was January 1.

Union of the crowns: 1603 (James IV of Scotland = James I of England)

Acts of union: 1707

But until 1752, the same day could be different dates in England & Scotland even though they were "the same country".

If you have to deal with 17th-18th calendar calculations…

Good luck!

Aside: The UK Tax Year

Why does the UK tax year start on 6th April?

Recall: New Year's Day used to be 25th March.

When the calendar switch happened, the treasury didn't want to lose revenue, and the people didn't want to be charged extra. So the tax year start was adjusted (by the 11 days) to 5th April.

In 1800, the Julian calendar slipped another day with respect to the Gregorian calendar, and the tax year start was moved a day to 6th April.

And it's been there ever since...

A Maths Problem? Gauss Has a Go

In 1800, Gauss published an algorithm for calculating Gregorian Easter.

Amended in 1807, 1811, 1816.

And still contains an edge case that is dealt with separately.

It turns out that calculating Easter is nontrivial, even for the best minds.

Gauss' Algorithm

```
constexpr auto easter_sunday(int year) -> std::chrono::year_month_day {
  const auto a = year % 19;
  const auto b = year % 4;
  const auto c = year % 7;

  const auto k = year / 100;
  const auto p = (13 + 8 * k) / 25;
  const auto q = k / 4;
  const auto M = (15 - p + k - q) % 30;
  const auto N = (4 + k - q) % 7;

  auto d = (19 * a + M) % 30;
  const auto e = (2 * b + 4 * c + 6 * d + N) % 7;
  if (e == 6 and (d == 28 or d == 29)) { d -= 7; }

  const auto base = std::chrono::March / 22 / year;
  const auto offset = std::chrono::days{d + e};
  return static_cast<std::chrono::sys_days>(base) + offset;
}
```

The Algorithm You Should Actually Use

Now known as the Meeus/Jones/Butcher algorithm.

Published originally in 1876 in Nature from "a New York correspondent".

Most recently in *Astronomical Algorithms* (1991) by Jean Meeus.

A tidier variant was published in *New Scientist* on 30th March 1961 (Puzzles and paradoxes by T.H. O'Beirne: *How ten divisions lead to Easter*).

Meeus/Jones/Butcher (NS variant)

```
constexpr auto easter_sunday(int year) -> std::chrono::year_month_day {
  const auto a = year % 19;
  const auto [b, c] = std::div(year, 100);
  const auto [d, e] = std::div(b, 4);
  const auto g = (8 * b + 13) / 25;
  const auto h = (19 * a + b - d - g + 15) % 30;
  const auto [i, k] = std::div(c, 4);
  const auto l = (32 + 2 * e + 2 * i - h - k) % 7;
  const auto m = (a + 11 * h + 19 * l) / 433;
  const unsigned n = (h + l - 7 * m + 90) / 25;
  const auto p = (h + l - 7 * m + 33 * n + 19) % 32;
  return std::chrono::month{n} / p / year;
}
```

Unit Testing for Easter

The *earliest* it can ever occur is *22nd March*, i.e. full moon on Saturday 21st March. This occurred in 1818 and will occur again in 2285.

Early Easters in our lifetime: 23rd March 2008, 25th March 2035, 27th March 2016.

The *latest* it can ever occur is *25th April*, i.e. full moon on Saturday 20th March puts the first full moon after the vernal equinox on Monday 19th April. This occurred in 1943 and will occur again in 2038 and 2190.

Other late Easters in our lifetime: 23rd April 2000, 24th April 2011.

Easter 2023 is on 9th April, more or less in the middle of where it can be.

The Easter Act 1928 (UK)

Passed, but (unfortunately?) never brought into force.

```
constexpr auto easter_sunday(int year) -> std::chrono::year_month_day {
  const auto second_sat = std::chrono::year_month_weekday{
      std::chrono::year{year}, std::chrono::April,
      std::chrono::weekday_indexed{std::chrono::Saturday, 2}};
  return ++static_cast<std::chrono::sys_days>(second_sat);
}
```

The Easter Act 1928 (UK)

Passed, but (unfortunately?) never brought into force.

```
constexpr auto easter_sunday(int year) -> std::chrono::year_month_day {
  const auto second_sat =
      std::chrono::April / std::chrono::Saturday[2] / year;
  return ++static_cast<std::chrono::sys_days>(second_sat);
}
```

Alternative calculation

Alternatively, maybe we can calculate directly using a known full moon `time_point` and the synodic month `duration`.

```
constexpr auto first_full_moon_of_2000 =
    static_cast<sys_days>(January/21/2000) + 4h + 40min;
constexpr auto synodic_month =
    duration<std::int64_t, std::ratio<2'551'443>>{1};
```

Age of the moon on March 21st

This is the "age of the moon" measured from full moon.

```
constexpr auto age_of_the_moon_on_march_21(int year) -> seconds {
  const auto march_21 = static_cast<sys_days>(March / 21 / year);
  return (march_21 - first_full_moon_of_2000) % synodic_month;
}
```

The Paschal full moon

```
constexpr auto paschal_full_moon(int year) -> year_month_day {
  auto age = age_of_moon_on_march_21(year);
  if (age < 0s) { age += synodic_month; }
  const auto time_to_full = synodic_month - age;
  return time_point_cast<days>(
      static_cast<sys_days>(March/21/year) + time_to_full);
}
```

Easter Sunday

```
constexpr auto easter_sunday(int year) -> year_month_day {
  const auto moon = static_cast<sys_days>(paschal_full_moon(year));
  const auto weekday = year_month_weekday{moon}.weekday();
  return ++(moon + (Saturday - weekday));
}
```

Cave Computum!

The direct calculation method is astronomically accurate*, but the devil is in the details...

The first full moon of 2000 was at approx 4:40 *UTC* on January 21st.

You can tell yourself this method is "right", but it *isn't what the church does*. And if this is wrong sometimes, it's probably wrong by a lot (at least a week, maybe a full lunar month).

For example this code gives the wrong date for Easter 2025.

Conclusion

Calendars in general are messy, human things.

Even when they try to match astronomy, it's still an approximation.

Easter is the most complex calculation in the Western calendar.

`std::chrono` can still help us out.

Or you could use a table lookup or two (for short date ranges).

A Few More Examples

What day of the week is that?

Convert through `sys_days`.

```
constexpr auto day_of_the_week(year_month_day ymd) -> weekday {
  return year_month_weekday{
      static_cast<sys_days>(ymd)}.weekday();
}
```

When is Thanksgiving this year?

```
constexpr auto thanksgiving_us(int year) -> year_month_day {
  return static_cast<sys_days>(November/Thursday[4]/year);
}

constexpr auto thanksgiving_ca(int year) -> year_month_day {
  return static_cast<sys_days>(October/Monday[2]/year);
}
```

When do I need a prescription refill?

```
constexpr auto prescription_refill_date() -> year_month_day {
  const auto surgery_date = March/15/2023;
  using medication_period = duration<std::int64_t,
                                     std::ratio<6 * 60 * 60>>;
  const auto number_of_pills = 30;
  const auto last_pill_at =
      sys_days{surgery_date} + medication_period{number_of_pills};
  return time_point_cast<days>(last_pill_at);
}
```

The *South* Denver Metro Area C++ Meetup

Our mortal enemies meet on the *last* Thursday of the month.

```
constexpr auto enemy_meetup(int year, auto month)
    -> std::chrono::year_month_day {
  const auto last_thurs =
      month / std::chrono::Thursday[std::chrono::last] / year;
  return static_cast<std::chrono::sys_days>(last_thurs);
}
```

When does Advent start?

```
constexpr auto advent(int year) -> year_month_day {
  const auto xmas_eve =
      --static_cast<sys_days>(December/25/year);
  const auto sunday = xmas_eve
      - (year_month_weekday{xmas_eve}.weekday() - Sunday);
  return sunday - weeks{3};
}
```

How many Friday 13ths are in 2023?

```
constexpr auto unlucky_fridays(int year) {
  return std::ranges::count_if(
      std::ranges::views::iota(1u, 13u),
      [&] (auto m) {
          const auto d = static_cast<sys_days>(month{m}/13/year);
          return year_month_weekday{d}.weekday() == Friday;
      });
}
```

Month arithmetic

Adding months to a `year_month_day` alters the month only and leaves the days alone. So you can end up with nonsense dates (e.g. Feb 30th*).

`std::chrono` deliberately does this because it's unclear how to interpret the programmer's intent.

```
auto ymd = January/30/2023;
ymd += months{1};
assert(not ymd.ok());
```

How to fixup after month arithmetic

```
auto date = January/31/2023;
date += months{1};

// fix 1: roll over into March
date = static_cast<sys_days>(date);

// fix 2: snap to end of February
date = date.year()/date.month()/last;
```

Aside: February 30th?

Sweden, 1699: the Government decides to adopt the Gregorian calendar gradually by skipping leap years from 1700-1740.

1700: 365 days (leap day skipped), good so far. Then: the Great Northern War broke out. Leap years were not skipped in 1704 or 1708.

1712: Charles XII decides to return to the Julian calendar *and* adds the skipped day back in. February 1712 has 30 days!

1753: Sweden properly changes to the Gregorian calendar, skipping February 18-28 in one go.

Working With std::chrono Dates

Use `year_month_day` for default (*proleptic* Gregorian calendar) representation.

Cast to `sys_days` (a `time_point`!) for arithmetic.

Use `weekday` for modulo arithmetic on days of the week.

Use `year_month_weekday` and `last` variants for those specific use cases.

Use `operator/` with month/day/year ordering to avoid verbosity.

Be *really* careful with "nth <weekday> before/after..." specs.

If you need Julian dates, you're off-standard (but you can still use the mod-7 weekday arithmetic).

References

Numerical Recipes [in C 2e]; Press/Teukolsky/Vetterling/Flannery; **978-0521880688**

Calendarium Perpetuum; L. Vitellius Triarius; **978-1494203696**

The Roman Calendar, Origins and Festivals; Marion Pearce; **978-1913768249**

Ecclesiastical History of the English People; Bede; **978-0140445657**

The Reckoning of Time; Bede (tr. Faith Wallis); **978-0853236931**

Calendrical Calculations, The Ultimate Edition; Rheingold/Dershowitz; **978-1107683167**

Astronomical Algorithms 2e; Jean Meeus; **978-0943396613**

Celestial Calculations: A Gentle Introduction to Computational Astronomy; J.L. Lawrence; **978-0262536639**

Puzzles & Paradoxes: Fascinating Excursions in Recreational Mathematics; T.H. O'Beirne; **978-0486246130**

The Easter Computus and the Origins of the Christian Era; Alden Mosshammer; **978-0199543120**

The History of Time: A Very Short Introduction; Leofranc Holford-Strevens; **978-0192804990**

The Mathematics of the Calendar; Marc Cohn; **978-0192804990**

The Oxford Companion to the Year; Bonnie Blackburn & Leofranc Holford-Strevens; **978-0192142313**

The Book of Common Prayer

Handy-Book of Rules and Tables

For Verifying Dates with the

Chriſtian Era;

Giving an Account of the Chief Eras, and Syſtems
uſed by various Nations;
with eaſy methods for determining the
correſponding dates;

With Regnal Years of Engliſh Sovereigns

From the Norman Conqueſt to the preſent time.

A.D. 1066 TO 1874.

By John J. Bond,

Aſſiſtant Keeper in Her Majeſty's Record Office.

References

https://a.co/fCN8eYD

https://www.youtube.com/playlist?list=PL3K4pawuL8R41P1PstWYbU_dgM97-DtFa

Puzzles and Paradoxes column, *New Scientist* 30th March 1961 (via Google Books)

https://digital.library.mcgill.ca/ms-17/fetchfoliodoc.php?target=Glossary

And too many Wikipedia articles to list

There's a *deep* rabbit hole here if you have time (and possibly money) to invest!

Final Thought

If you are a Duran Duran fan, you can light your torch and wave it for the *New Moon on Monday* accurately between the following days in 2023:

- 14th Feb - 19th Feb (new moon on Monday 20th Feb)
- 11th July - 16th July (new moon on Monday 17th July)
- 7th Nov - 12th Nov (new moon on Monday 13th Nov)

Calculation for other years left as an exercise...