КОНСТРУИРОВАНИЕ ПРОГРАММ И ЯЗЫКИ ПРОГРАМИРОВАНИЯ

Лекция № 25 — Библиотека времени

Преподаватель: Поденок Леонид Петрович, 505а-5

+375 17 293 8039 (505а-5)
+375 17 320 7402 (ОИПИ НАНБ)
prep@lsi.bas-net.by

ftp://student:2ok*uK2@Rwox@lsi.bas-net.by

Кафедра ЭВМ, 2021

Оглавление

<ratio> — заголовок рационального отношения

В этом заголовке объявляется шаблон класса отношений и несколько вспомогательных типов для работы с ними.

Отношение выражает пропорцию (например, 1:3 или 1000:1) как набор двух констант времени компиляции (числитель и знаменатель).

Для выражения отношения используется только тип (объекты этих типов значения не имеют).

В стандартной библиотеке типы соотношений используются в качестве параметров шаблона для объектов **`std::chrono::duration`** (продолжительность, длительность).

Шаблоны классов

`ratio` — класс отношения

Шаблоны классов арифметики отношений

Генерируют новые типы **`ratio`**, которые являются результатом операции

`ratio_add` — сложение двух отношений

`ratio_subtract` — вычитание двух отношений

`ratio_multiply` — умножение двух отношений

`ratio_divide` — деление двух отношений

Шаблоны классов сравнения отношений

`ratio_equal` — сравнить на равенство
`ratio_not_equal` — сравнить на неравенство
`ratio_less` — сравнить на меньше чем
`ratio_less_equal` — сравнить на меньше чем или равенство
`ratio_greater` — сравнить на больше чем
`ratio_greater_equal` — сравнить на больше чем или равенство

std::ratio — рациональное отношение

```
template <intmax_t N, intmax_t D = 1> class ratio;
```

Этот шаблон используется для создания *экземпляров типов*, которые представляют собой конечное рациональное число, обозначаемое парой — числитель и знаменатель.

Числитель и знаменатель реализованы как константы времени компиляции типа **intmax_t**[1].

Следует обратить внимание, что **ratio** не представляется объектом данного типа, а самим типом, который использует члены-константы времени компиляции для определения **ratio**.

Следовательно, **ratio** может использоваться только для выражения констант **constexpr** и не может содержать никаких значений.

Типы этого шаблона используются в стандартном классе **std::chrono::duration**.

N — числитель.

Его абсолютное значение должно быть в диапазоне представимых значений **intmax_t**.

D — знаменатель.

Его абсолютное значение должно быть в диапазоне представимых значений **intmax_t** и не должно быть нулевым.

intmax_t — самый широкий целочисленный тип со знаком.

---

1 <cstdint> (stdint.h)

Члены-константы

member constexpr	description
num	числитель
den	знаменатель

Значения **num** и **den** представляют уникальное наименьшее сокращение отношения **N:D**.

Это означает, что в некоторых случаях **num** и **den** не совпадают с аргументами шаблона **N** и **D**:

- если наибольший общий делитель (НОД) между **N** и **D** не равен единице, **num** и **den** являются результатом деления **N** и **D** на этот наибольший общий делитель (взаимно простые).

- знак всегда представлен числителем **num** (**den** всегда положительно).

Если **D** отрицательно, знак числа противоположен знаку **N**.

Создание экземпляров шаблонов

Существуют следующие предопределенные стандартные экземпляры **`ratio`**:

type	definition	description
yocto	`ratio<1, 1000000000000000000000000>`	10^{-24} *
zepto	`ratio<1, 1000000000000000000000>`	10^{-21} *
atto	`ratio<1, 1000000000000000000>`	10^{-18}
femto	`ratio<1, 1000000000000000>`	10^{-15}
pico	`ratio<1, 1000000000000>`	10^{-12}
nano	`ratio<1, 1000000000>`	10^{-9}
micro	`ratio<1, 1000000>`	10^{-6}
milli	`ratio<1, 1000>`	10^{-3}
centi	`ratio<1, 100>`	10^{-2}
deci	`ratio<1, 10>`	10^{-1}
deca	`ratio<10, 1>`	10^{1}
hecto	`ratio<100, 1>`	10^{2}
kilo	`ratio<1000, 1>`	10^{3}
mega	`ratio<1000000, 1>`	10^{6}
giga	`ratio<1000000000, 1>`	10^{9}
tera	`ratio<1000000000000, 1>`	10^{12}
peta	`ratio<1000000000000000, 1>`	10^{15}
exa	`ratio<1000000000000000000, 1>`	10^{18}
zetta	`ratio<1000000000000000000000, 1>`	10^{21} *
yotta	`ratio<1000000000000000000000000, 1>`	10^{24} *

Эти имена совпадают с префиксами, используемыми в стандартных единицах Международной системы единиц (СИ). Обе константы (*), должны быть представимы с помощью **`intmax_t`**.

Шаблоны классов арифметики отношений

Генерируют новые типы **ratio**, которые являются результатом операции

ratio_add – сложение двух отношений

ratio_subtract – вычитание двух отношений

ratio_multiply – умножение двух отношений

ratio_divide – деление двух отношений

```
#include <iostream>
#include <ratio>

int main () {

  typedef std::ratio<1, 2> one_half;
  typedef std::ratio<2, 3> two_thirds;

  typedef std::ratio_add<one_half, two_thirds> sum;

  std::cout << "sum = " << sum::num << "/" << sum::den;
  std::cout << " (which is: " << (double(sum::num)/sum::den) << ")" << "\n";

  return 0;
}
```

Вывод

```
sum = 7/6 (which is: 1.16667)
```

Шаблоны классов сравнения отношений

Сравниваются два типа отношений

ratio_equal – сравнить на равенство

ratio_not_equal – сравнить на неравенство

ratio_less – сравнить на меньше чем

ratio_less_equal – сравнить на меньше чем или равенство

ratio_greater – сравнить на больше чем

ratio_greater_equal – сравнить на больше чем или равенство

```
#include <iostream>
#include <ratio>

int main () {
  typedef std::ratio<1, 2> one_half;
  typedef std::ratio<2, 4> two_fourths;

  std::cout << "1/2 == 2/4 ? " << std::boolalpha; // true/false вместо 1/0
  std::cout << std::ratio_equal<one_half, two_fourths>::value << std::endl;

  return 0;
}
```

Вывод

```
1/2 == 2/4 ? true
```

Пример ratio

```
#include <iostream>
#include <ratio>

int main () {

  typedef std::ratio<1, 3> one_third;
  typedef std::ratio<2, 4> two_fourths;

  std::cout << "one_third = " << one_third::num << "/" << one_third::den
            << std::endl;
  std::cout << "two_fourths = " << two_fourths::num << "/" << two_fourths::den
            << std::endl;

  typedef std::ratio_add<one_third, two_fourths> sum;

  std::cout << "sum = " << sum::num << "/" << sum::den;
  std::cout << " (which is: " << (double(sum::num)/sum::den) << ")"
            << std::endl;

  std::cout << "1 килограмм = " << (std::kilo::num/std::kilo::den) << "грамм";
  std::cout << std::endl;

  return 0;
}
```

Вывод

one_third= 1/3

two_fourths= 1/2

sum = 5/6 (which is 0.833333)

1 килограмм = 1000 грамм

<chrono> — библиотека времени

Элементы в этом заголовке связаны со временем.

chrono — это имя и заголовка, и подпространства имен — все элементы в этом заголовке (за исключением специализации шаблона **common_type**) не определены непосредственно в пространстве имен **std** (как и большая часть стандартной библиотеки), а в пространстве имен **std::chrono**.

В заголовке определены три концепции:

Durations — длительность

Измеряет промежутки времени, например, одну минуту, два часа или десять миллисекунд.

Длительность представлена объектами шаблона класса длительности, который объединяет значение счетчика и точности периода (например, десять миллисекунд имеют десять как значение счетчика и миллисекунды как точность периода).

Time points — Моменты времени

Ссылка на конкретный момент времени, например, день рождения, сегодняшний рассвет или время прохождения следующего поезда.

Моменты времени представлены объектами шаблона класса **time_point**. Они выражаются с помощью длительности относительно *эпохи* (которая является фиксированной точкой времени, общей для всех объектов **time_point**, которые используют одни и те же часы.

Clocks — часы

Структура, которая связывает точку времени с реальным физическим временем.

Библиотека предоставляет по крайней мере три тактовых генератора, которые предоставляют средства для выражения текущего времени, как **`time_point`**:

`system_clock`
`stable_clock`
`high_resolution_clock`

В заголовке определено несколько классов и функций.

Классы длительности и моментов времени:

`duration` — длительность (шаблон)
`time_point` — момент времени (шаблон)

Классы часов — обеспечивают доступ к текущей **`time_point`**.

`system_clock` — системные часы (класс)
`steady_clock` — стабильно идущие часы (класс)
`high_resolution_clock` — часы высокого разрешения (класс)

Классы характеристик и типов

`treat_as_floating_point` — трактовать, как плавающую запятую
`duration_values` — значения длительности
`common_type(duration)` — специализация стандартного класса характеристик **`common_type`**

Шаблоны функций

`duration_cast` — приведение к длительности
`time_point_cast` — приведение к моменту времени

Типы для создания экземпляров класса

В пространстве имен **`std::chrono`** также определены некоторые типы экземпляров длительности:

`hours` — длительность в часах
`minutes` — длительность в минутах
`seconds` — длительность в секундах
`milliseconds` — длительность в милисекундах
`microseconds` — длительность в микросекундах
`nanoseconds` — длительность в наносекундах

std::chrono::duration — шаблон класса длительности

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

Объект длительности выражает промежуток времени с помощью членов **count** и **period**. Внутри объект хранит **count** как объект типа члена **rep** (псевдоним первого параметра шаблона, **Rep**), значение которого можно получить, вызвав функцию-член **count**.

Значение **count** выражается в **period**.

Длина периода интегрируется в тип (во время компиляции) его вторым параметром шаблона (**Period**), который представляет собой тип **ratio**, выражающий количество (или долю) секунд в каждом периоде.

Rep — (тип члена **rep**) арифметический тип или класс, имитирующий арифметический тип, который будет использоваться в качестве типа для внутреннего значения **count**.

Period — (тип члена **period**) тип отношения, представляющий период в секундах.

type	Представление	Период
hours	знаковый целочисленный тип размером не менее 23 бит	ratio<3600, 1>
minutes	знаковый целочисленный тип размером не менее 29 бит	ratio<60, 1>
seconds	знаковый целочисленный тип размером не менее 35 бит	ratio<1, 1>
milliseconds	знаковый целочисленный тип размером не менее 45 бит	ratio<1, 1000>
microseconds	знаковый целочисленный тип размером не менее 55 бит	ratio<1, 1000000>
nanoseconds	знаковый целочисленный тип размером не менее 64 бит	ratio<1, 1000000000>

Открытые функции члены

`(constructor)` — создает объект типа **`duration`**
`(destructor)` — разрушает объект типа **`duration`**
`count()` — вернуть значение счетчика

Статические функции-члены

`zero()` — нулевое значение
`min()` — минимальное значение длительности
`max()` — максимальное значение длительности

Функции-не-члены
`operators` — операторы длительности

std::chrono::duration::duration — конструкторы

(1) по умолчанию

```
duration( ) = default;
```

Создает объект со значением счетчика, инициализированным по умолчанию.

(2) с инициализацией 1

```
duration(const duration& dtn);
```

Инициализирует объект длительностью **dtn**.

dtn — другой объект длительности.

(3) **с инициализацией 2**

```
template <class Rep2, class Period2>
constexpr duration(const duration <Rep2, Period2>& dtn);
```

Инициализирует объект длительностью **dtn**.

(4) с инициализацией 3

```
template<class Rep2>
constexpr explicit duration(const Rep2& n);
```

Инициализирует объект длительностью, **count** которой равен **n**.

duration <Rep2, Period2> — это тип, который не может вызывать неявную ошибку усечения при преобразовании.

Rep2 — это арифметический тип (или класс, имитирующий арифметический тип).

Пример конструкторов duration

```
#include <iostream>
#include <ratio>
#include <chrono>

int main () {

  typedef std::chrono::duration<int>                    seconds_type;
  typedef std::chrono::duration<int, std::milli>        milliseconds_type;
  typedef std::chrono::duration<int, std::ratio<60*60>> hours_type;

  hours_type         h_oneday(24);          // 24h
  seconds_type       s_oneday(60*60*24);    // 86400s
  milliseconds_type  ms_oneday(s_oneday);   // 86400000ms

  seconds_type s_onehour(60*60);            // 3600s
//hours_type   h_onehour(s_onehour);        // NOT VALID (усечение), надо:
  hours_type   h_onehour(std::chrono::duration_cast<hours_type>(s_onehour));
  milliseconds_type ms_onehour(s_onehour);  // 3600000ms (нет усечения)

  std::cout << ms_onehour.count() << "ms in 1h" << std::endl;

  return 0;
}
```

Вывод

```
3600000ms in 1h
```

std::chrono::duration::count — вернуть значение длительности

```
constexpr rep count() const;
```

Возвращает значение внутреннего счетчика объекта длительности.

Возвращаемое значение — это текущее значение внутреннего представления, выраженное в терминах интервала периода класса, который не обязательно равен секундам.

Пример

```
// duration::count
#include <iostream>     // std::cout
#include <chrono>       // std::chrono::seconds, std::chrono::milliseconds
                        // std::chrono::duration_cast

int main () {

  using namespace std::chrono;
  // std::chrono::milliseconds является воплощением std::chrono::duration:
  milliseconds foo(1000); // 1 second
  foo *= 60;

  std::cout << "duration (in periods): ";
  std::cout << foo.count() << " milliseconds.\n";

  std::cout << "duration (in seconds): ";
  std::cout << foo.count()*milliseconds::period::num/milliseconds::period::den;
  std::cout << " seconds.\n";

  return 0;
}
```

std::chrono::duration operators — операторы длительности

Функции-члены

```
          duration& operator=  (const duration& rhs) = default;
constexpr duration  operator+  () const;
constexpr duration  operator-  () const;
          duration& operator++ ();
          duration  operator++ (int);
          duration& operator-- ();
          duration  operator-- (int);
          duration& operator+= (const duration& rhs);
          duration& operator-= (const duration& rhs);
          duration& operator*= (const rep& r);
          duration& operator/= (const rep& r);
          duration& operator%= (const rep& r);
          duration& operator%= (const duration& rhs);
```

Функции-не-члены

Перегруженные арифметические и реляционные операторы
+, **-**, *****, **/**, **%**, **==**, **!=**, **<**, **<=**, **>=**, **>**.

Все они **constexpr**

Выполняют соответствующую операцию над задействованными объектами длительности, как если бы она была применена прямо к своему внутреннему объекту **count**.

Пример операторов над длительностями

```
#include <iostream>
#include <ratio>
#include <chrono>

int main () {

  std::chrono::duration<int> foo;     //
  std::chrono::duration<int> bar(10); //

                    // counts: foo bar
                    //         --- ---
  foo = bar;                   // 10  10
  foo = foo + bar;             // 20  10
  ++foo;                       // 21  10
  --bar;                       // 21   9
  foo *= 2;                    // 42   9
  foo /= 3;                    // 14   9
  bar += (foo % bar);          // 14  14

  std::cout << std::boolalpha;
  std::cout << "foo==bar: " << (foo==bar) << std::endl;
  std::cout << "foo: " << foo.count() << std::endl;
  std::cout << "bar: " << bar.count() << std::endl;

  return 0;
}
```

std::chrono::system_clock — класс системных часов

```
class system_clock;
```

Классы часов обеспечивают доступ к текущему моменту времени **time_point**.
В частности, **system_clock** — это общесистемные часы реального времени.

Свойства

realtime — предназначены для представления реального времени, и, следовательно, его можно образом преобразовывать в представления календаря и обратно (**to_time_t( )** и **from_time_t( )**).

signed count — значения **time_point** могут относиться к временам до эпохи (отрицательные значения).

system-wide — все процессы, запущенные в системе, с помощью этих часов получают одни и те же значения **time_point**.

Константный член

`is_steady` — значение типа **`bool`**, определяющее, идут ли часы все время вперед и находятся ли они в устойчивом состоянии относительно физического времени. Если возвращена истина, это означает, что системные часы не могут регулироваться.

`steady_clock` — монотонное время, идущее только вперед. Время между тиками всегда постоянно и это позволяет лучше измерять интервалы, чем в случае **`system_clock`**.

Проблема использования **`system_clock`** для измерения прошедшего времени заключается в том, что во время работы при измерении интервала часы могут быть перенастроены, например, при синхронизации с другими часами по сети, может произойти пеерход на на летнее время и т.п.. steady_clock не подлежат корректировке, поэтому они предпочтительнее при наблюдении за прошедшим временем.

Статические функции-члены

`now()` — получить текущее время
`to_time_t()` — преобразовать в **`time_t`**
`from_time_t()` — преобразовать из **`time_t`**

std::chrono::system_clock::now — получить текущее время

```
static time_point now( ) noexcept;
```

Возвращает текущий момент времени в рамках **system_clock**.
time_point — это тип члена, определенный как псевдоним **time_point <system_clock>**.

std::chrono::system_clock::to_time_t — преобразовать в формат time_t

```
static time_t to_time_t (const time_point& tp) noexcept;
```

std::chrono::system_clock::from_time_t — преобразовать из формата time_t

```
static time_point from_time_t (time_t t) noexcept;
```

time_t — тип времени из <ctime> (<time.h>)

Псевдоним основного арифметического типа, способного представлять время, возвращаемое функцией **time( )**. По историческим причинам это обычно реализуется как целое значение, представляющее количество секунд, прошедших с 00:00 часов 1 января 1970 года по всемирному координированному времени (т.е. временная метка unix). Хотя библиотеки могут реализовать этот тип с использованием альтернативных представлений времени.

Пример system_clock::now()

```
#include <iostream>
#include <ctime>
#include <ratio>
#include <chrono>

int main () {

  using namespace std::chrono;

  duration<int, std::ratio<60*60*24> > one_day(1); //  60*60*24 с в периоде

  system_clock::time_point today    = system_clock::now();
  system_clock::time_point tomorrow = today + one_day;

  time_t tt = system_clock::to_time_t(today);
  std::cout << "today is: " << ctime(&tt);  tt =
system_clock::to_time_t(tomorrow);
  std::cout << "tomorrow will be: " << ctime(&tt);

  return 0;
}
```

Вывод

```
today is: Thu Dec 23 11:34:44 2021
tomorrow will be: Fri Dec 24 11:34:44 2021
```

std::chrono::steady_clock — стабильные часы

```
class steady_clock;
```

Класс **stable_clock** специально разработан для расчета временных интервалов.

Свойства

монотонные (monotonic) — его функция-член **now( )** никогда не возвращает более низкое значение, чем в предыдущем вызове.

стабильные (steady) — на каждый тик, на который часы смещаются, уходит одинаковое количество физического времени.

Константный член

is_steady( ) — всегда **true**

std::chrono::steady_clock::now() — вернуть текущее время

```
static time_point now( ) noexcept;
```

Пример steady_clock

```
#include <iostream>
#include <ctime>
#include <ratio>
#include <chrono>

int main () {

  using namespace std::chrono;
  typedef  steady_clock::period sc_tick;
  steady_clock::time_point clock_begin = steady_clock::now();

  std::cout << "печатаем 1000 звездочек...\n";
  for (int i = 0; i < 1000; ++i)
     std::cout << "*";
  std::cout << std::endl;

  steady_clock::time_point clock_end = steady_clock::now();
  steady_clock::duration time_span = clock_end - clock_begin;
  double nseconds = double(time_span.count()) * sc_tick::num/sc_tick::den;

  std::cout << "It took me " << nseconds << " seconds.";
  std::cout << std::endl;

  return 0;
}
```

Вывод

```
печатаем 1000 звездочек...
**************************************************
**************************************************
**************************************************
**************************************************
**************************************************
**************************************************
**************************************************
**************************************************
**************************************************
**************************************************
**************************************************
**************************************************
**********
It took me 9.2979e-05 seconds.
```

Пример steady_clock 2

```
#include <iostream>
#include <ctime>
#include <ratio>
#include <chrono>

int main () {

  using namespace std::chrono;

  steady_clock::time_point t1 = steady_clock::now();

  std::cout << "печатаем 1000 звездочек...\n";
  for (int i = 0; i < 1000; ++i)
     std::cout << "*";
  std::cout << std::endl;
  steady_clock::time_point t2 = steady_clock::now();

  duration<double> time_span = duration_cast<duration<double>>(t2 - t1);

  std::cout << "It took me " << time_span.count() << " seconds.";
  std::cout << std::endl;

  return 0;
}
```

std::chrono::high_resolution_clock — часы высокого разрешения

```
class high_resolution_clock;
```

high_resolution_clock — часы с самым коротким периодом тика.
Они могут быть синоним **system_clock** или **stable_clock**.
Могут быть **steady**.
Имеют одну функцию-член **now( )**

std::chrono::time_point — шаблон класса момент времени

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

Объект типа **time_point** выражает момент времени относительно эпохи часов.

Внутри объект хранит объект типа **duration** и использует тип **Clock** в качестве ссылки для своей эпохи.

Параметры шаблона

Clock — класс часов, например **system_clock**, **stable_clock**, **high_resolution_clock** или пользовательский класс часов.

Duration — тип длительности

Следующие псевдонимы являются типами членов **time_point**. Они используются функциями-членами в качестве типов параметров и возвращаемых значений:

Тип члена	Определение	Примечание
clock	первый параметр шаблона (**Clock**)	Класс часов (**system_clock**, **high_resolution_clock**, **stable_clock**, или пользовательский класс часов).
duration	Второй параметр шаблона (**Duration**)	Тип **duration**, используемый для представления **time point**
rep	duration::rep	Тип, возвращаемый функц. **duration::count()**
period	duration::period	Тип **ratio**, представляющий длину периода в секундах

std::chrono::time_point::time_point — конструкторы

(1) По умолчанию

```
time_point();
```

Создает объект с эпохой в качестве значения.

(2) копирования из объекта типа time_point

```
template <class Duration2> time_point(const time_point<clock, Duration2>& tp);
```

Создает объект, представляющий тот же момент времени, что и **tp**. Вызывается только в том случае, если **Duration2** неявно преобразуется в тип длительности вновь созданного объекта.

(3) Из объекта типа duration

```
explicit time_point (const duration& dtn);
```

Создает объект, представляющий момент времени, когда с начала эпохи истекает время **dtn**.

tp — другой объект time_point.

time_point <clock, Duration2> — это тип **time_point**, который использует те же часы и имеет тип длительности, неявно конвертируемый в тип во вновь созданном объекте.

dtn — объект типа **duration.**

duration - это тип члена, определяемый как тип длительности, используемый объектом.

Пример конструктора `time_point`

```
#include <iostream>
#include <chrono>
#include <ctime>

int main () {

  using namespace std::chrono;

  system_clock::time_point tp_epoch;  // epoch value

  time_point <system_clock, duration<int>> tp_seconds(duration<int>(1));

  system_clock::time_point tp(tp_seconds);

  std::cout << "1 second since system_clock epoch = ";
  std::cout << tp.time_since_epoch().count();
  std::cout << " system_clock periods." << std::endl;

  // display time_point:
  std::time_t tt = system_clock::to_time_t(tp);
  std::cout << "time_point tp is: " << ctime(&tt);
}
```

Вывод

```
1 second since system_clock epoch = 1000000000 system_clock periods.
time_point tp is: Thu Jan  1 03:00:01 1970
```

std::chrono::time_point operators — операторы для момента времени

Выполняет соответствующую операцию над задействованными объектами **time_point**, как если бы она была применена непосредственно к его внутреннему объекту **duration**.

Поддерживаются следующие операции:

	Операция	Возвращается
составное присваивание (функции-члены)	`tp += dtn`	`*this`
	`tp -= dtn`	`*this`
арифметические операторы (функции-не-члены)	tp + dtn	значение типа `time_point`
	dtn + tp	значение типа `time_point`
	tp - dtn	значение типа `time_point`
	tp - tp2	значение типа `duration`
реляционные операторы (функции-не-члены)	`tp == tp2`	значение типа `bool`
	`tp != tp2`	значение типа `bool`
	`tp < tp2`	значение типа `bool`
	`tp > tp2`	значение типа `bool`
	`tp >= tp2`	значение типа `bool`
	`tp <= tp2`	значение типа `bool`

tp и **tp2** — объекты **time_point**, а **dtn** — объект длительности.

Пример time_point операторов

```
#include <iostream>
#include <chrono>

int main () {

  using namespace std::chrono;

  system_clock::time_point tp, tp2;                 // epoch value
  system_clock::duration   dtn(duration<int>(1));  // 1 second

                       //  tp     tp2    dtn
                       //  ---    ---    ---
  tp  += dtn;          //  e+1s   e      1s
  tp2 -= dtn;          //  e+1s   e-1s   1s
  tp2  = tp  + dtn;    //  e+1s   e+2s   1s
  tp   = dtn + tp2;    //  e+3s   e+2s   1s
  tp2  = tp2 - dtn;    //  e+3s   e+1s   1s
  dtn  = tp  - tp2;    //  e+3s   e+1s   2s

  std::cout << std::boolalpha;
  std::cout << "tp == tp2: " << (tp == tp2) << std::endl;
  std::cout << "tp > tp2: " << (tp > tp2) << std::endl;
  std::cout << "dtn: " << dtn.count() << std::endl;

  return 0;
}
```

Вывод

```
tp == tp2: false
tp > tp2: true
dtn: 2000000
```

std::chrono::time_point::time_since_epoch() — продолжительность

```
duration time_since_epoch( ) const;
```

Возвращает объект длительности со значением промежутка времени между эпохой и некоторым моментом времени.

Возвращаемое значение является текущим значением объекта внутренней длительности.

Пример time_point::time_since_epoch()

```
#include <iostream>
#include <chrono>
int main ( ) {
  using namespace std::chrono;
  typedef  system_clock::period scp;
  system_clock::time_point tp = system_clock::now( );
  system_clock::duration dtn = tp.time_since_epoch( );
  std::cout << "current time since epoch, expressed in:" << std::endl;
  std::cout << "periods: " << dtn.count( ) << std::endl;
  std::cout << "seconds: " << dtn.count( ) *scp::num /scp::den;
  std::cout << std::endl;
  return 0;
}
```

Вывод

```
current time since epoch, expressed in:
periods: 1338280396212871
seconds: 1338280396
```

std::chrono::duration_cast — шаблон функции преобразования

```
template <class ToDuration, class Rep, class Period>
constexpr ToDuration duration_cast(const duration<Rep, Period>& dtn);
```

Преобразует значение длительности **dtn** в какой-либо другой тип длительности с учетом различий в их периодах.

Функция не использует неявных преобразований. Вместо этого все значения счетчика внутренне преобразуются в самое широкое представление (**common_type** для внутренних типов счетчика), а затем приводятся к целевому типу, причем все преобразования выполняются явно с помощью **static_cast**.

Если целевой тип имеет меньшую точность, значение усекается.

std::chrono::time_point_cast — шаблон функции преобразования

```
template <class ToDuration, class Clock, class Duration>
time_point<Clock, ToDuration>                                  // тип возврата
time_point_cast(const time_point<Clock,Duration>& tp);
```

Преобразует значение **tp** в тип **time_point** с другим внутренним объектом длительности с учетом различий в периодах их длительности. Использует **duration_cast** для преобразования внутренних объектов длительности.

Следует обратить внимание, что первым параметром шаблона функции является не возвращаемый тип, а его компонент длительности.