From Time Point to Calendar Time

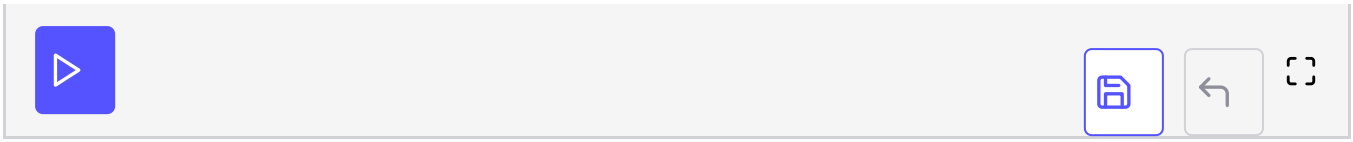
This lesson gives a brief introduction to calendar time and its usage in C++ with the help of interactive examples.

WE'LL COVER THE FOLLOWING

Thanks to `std::chrono::system_clock::to_time_t`, you can convert a time point that internally uses `std::chrono::system_clock` to an object of type `std::time_t`. Further conversion of the `std::time_t` object with the function `std::gmtime` gives you the calendar time, expressed in Coordinated Universal Time (UTC). In the end, this calendar time can be used as the input for the function `std::asctime` to get a textual representation of the calendar time.

```
// timepoint.cpp

#include <chrono>
#include <ctime>
#include <iostream>
#include <string>

int main(){

    std::cout << std::endl;

    std::chrono::time_point<std::chrono::system_clock> sysTimePoint;
    std::time_t tp= std::chrono::system_clock::to_time_t(sysTimePoint);
    std::string sTp= std::asctime(std::gmtime(&tp));
    std::cout << "Epoch: " << sTp << std::endl;

    tp= std::chrono::system_clock::to_time_t(sysTimePoint.min());
    sTp= std::asctime(std::gmtime(&tp));
    std::cout << "Time min: " << sTp << std::endl;

    tp= std::chrono::system_clock::to_time_t(sysTimePoint.max());
    sTp= std::asctime(std::gmtime(&tp));
    std::cout << "Time max: " << sTp << std::endl;

    sysTimePoint= std::chrono::system_clock::now();
    tp= std::chrono::system_clock::to_time_t(sysTimePoint);
    sTp= std::asctime(std::gmtime(&tp));
    std::cout << "Time now: " << sTp << std::endl;
```

```
}
```

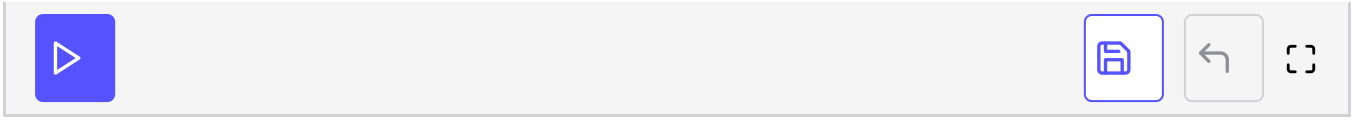


The output of the program shows the valid range of `std::chrono::system_clock`. On my Linux PC, `std::chrono::system_clock` has the UNIX-epoch as the starting point, and can have time points between the years 1677 and 2262. You can add time durations to time points to get new time points; However, note that adding time durations beyond the valid time range is undefined behavior.

Cross the valid Time Range

The following example uses the current time and adds or subtracts 1000 years. For the sake of simplicity, I ignore leap years and assume that a year has 365 days.

```
// timepointAddition.cpp

#include <chrono>
#include <ctime>
#include <iostream>
#include <string>

using namespace std::chrono;
using namespace std;

string timePointAsString(const time_point<system_clock>& timePoint){
  time_t tp= system_clock::to_time_t(timePoint);
  return asctime(gmtime(&tp));
}

int main(){

  cout << endl;

  time_point<system_clock> nowTimePoint= system_clock::now();
  cout << "Now:              " << timePointAsString(nowTimePoint) << endl;

  const auto thousandYears=  hours(24*365*1000);
  time_point<system_clock>  historyTimePoint= nowTimePoint - thousandYears;
  cout << "Now - 1000 years: " << timePointAsString(historyTimePoint) << endl;

  time_point<system_clock>  futureTimePoint= nowTimePoint + thousandYears;
  cout << "Now + 1000 years: " << timePointAsString(futureTimePoint) << endl;

}
```



For readability, I introduced the namespace `std::chrono`. The output of the program shows that an overflow of the time points in lines 25 and 28 causes incorrect results. Subtracting 1000 years from the current time point gives a time point in the future; adding 1000 years to the current time point gives a time point in the past.

The difference between two time points is a time duration. Time durations support the basic arithmetic and can be displayed in different time ticks.