CSE 437

REAL TIME SYSTEM ARCHITECTURE

HOMEWORK 1

REPORT 1

- ## Requirements

The program must be thread-safe, allowing for multiple threads to register and cancel timer events simultaneously without causing race conditions or deadlocks. The program should use a minimum amount of system resources, avoiding unnecessary CPU cycles and memory usage when waiting for timer events to fire.

In this function, std::chrono::high_resolution_clock::time_point type is required, takes this type argument and converts to long type.

```
long timepoint_to_long(const Timepoint& tp)
```

In this function, a variable in type Timepoint and a variable in type TTimerCallback, takes these arguments and run the callback once at time point tp.

```
void registerTimer(const Timepoint& tp, const TTimerCallback& cb)
```

In this function, a variable in type Millisecs and a variable in type TTimerCallback, takes these arguments and run the callback periodically forever.

```
void registerTimer(const Millisecs& period, const TTimerCallback& cb)
```

In this function, a variable in type Millisecs, Timepoint and TTimerCallback, takes these arguments and run the callback periodically until time point tp.

```
void registerTimer(const Timepoint& tp, const Millisecs& period, const TTimerCallback& cb)
```

In this function, a variable in type Millisecs, TPredicate and TTimerCallback, takes these arguments and run the callback periodically. After calling the callback every time, call the predicate to check if the termination criterion is satisfied. If the predicate returns false, stop calling the callback.

```
void registerTimer(const TPredicate& pred, const Millisecs& period, const TTimerCallback& cb)
```

- Constraints

The Timer class is implemented as a single-threaded loop that runs continuously in the background, checking for registered timer events and executing their callback functions when they are due. As a result, this implementation may not be suitable for high-precision timing applications that require sub-millisecond accuracy or low-latency response times.

The Timer class uses a hash map to store registered timer events, which may incur some memory overhead for large numbers of events or long-running periods.

- Assumptions

It assumes valid, non-null function pointers, non-negative time point, non-null predicate function pointers for all registered timer events. And callback functions and predicates are thread-safe and do not block or take a long time to execute.

- Design of Timer

Used unordered_map to store time_point as key, events as value (a thread-safe data structure that maps time points to lists of registered events). My events struct includes time_point, callback, is_periodic, period, predicate function, until time, is_until_set variables. For each registerTimer function, initialized these variables and pushed them to map. And uses a dedicated timer loop thread to execute events at the appropriate times. The timer loop thread checks the current time against the registered events, executes the callbacks of any events that are due, and updates the data structure with any new events that have been registered.

- Build and Test

1. make (Compiles the program)
2. ./a.out (Runs the program)
3. make clean (removes the binary files)