# GTU Department of Computer Engineering
# CSE 437 – Spring 2023
# Homework #1 Report

**Hasan Mutlu**
**1801042673**

## Requirements

**Thread Safety:** The timer class should be thread-safe. The timer thread and the main program the timer class runs on should work in synchronized way. No race conditions or deadlocks should happen.

**Timer Registration Types:** The timer class should allow registering four diffferent kinds of timers that work in different ways, as described in the homework PDF.

**Timer Precisions:** The registered timers should generate their events accurately, at the required times. By observing the example output in the homework PDF, it is decided that the timer error margin should be +-20 milliseconds.

### Assumptions

- ➔ The class relies on C++11 features such as high resolution clock. It is assumed that these features are working accurately.

## Design of the Timer Class

The name of the class is Timer and it implements the Itimer interface. It has the following fields:

- ➔ std::atomic<bool> isRunning: A boolean value to indicate the running status of the timer.
- ➔ std::vector<TimerTask> taskVec: A vector to save record of timer tasks.
- ➔ std::mutex mutx: A mutex to to protect access to task vector.
- ➔ std::condition_variable cond: A condition variable to provide synchronization between the timer thread and the main program.
- ➔ std::thread timerThread: Timer thread.

It has two functions other than ITimer interface functions:

- ➔ timerThreadFunc: Timer thread function.
- ➔ getNextTaskIndex: A function that fetches the index of the next task to be executed.

### TimerTask Struct

The class also has a struct called **TimerTask** to keep data about each timer. It has following fields and methods:

- ➔ Timepoint tp: The timepoint of the next event.

- ➔ Timepoint endTp: The ending point of a periodic timer, if it is type 3.
- ➔ Millisecs period: Period of the timer.
- ➔ TPredicate pred: The predicate of the timer, if it is type 4.
- ➔ TTimerCallback callback: Callback function.
- ➔ timerType type: Type of the timer, defined with the timerType enum. It has four possible values for each timer type: ONCE, PERIODIC_FRVR, PERIODIC_UNTIL, PERIODIC_PRED.

The TimerTask has four constructors for each type of timer, works in parallel with the ITimer interface methods and does required assignments to its fields.

### Work Flow of the Timer Class

When a Timer instance is created, it creates and starts the timer thread. When an overloaded registerTimer() function is called, it assigns this timer using the respective TimerTask constructor, adds the task to the task vector and notifies the timer thread about the new task. Adding new task to vector is protected using mutex.

The timer thread continuously checks the task vector and waits on a notification if it is empty. If not, it fetches the next task to be executed. If the next task's time hasn't arrived yet, it goes on to wait again until the time point. When the time is arrived for the task to be executed, its callback function is executed. After that, the next execution time is decided according to the type of the timer. If it is supposed to run once, it is directly removed from the task vector. If it is periodic, according to its type, it is checked whether the ending time is reached for third type of timer or the termination criteria is achieved for the fourth type of timer. They are removed from the task list if these conditions are satisfied. If not, the next time point of the timer is calculated and prepared for the next execution.

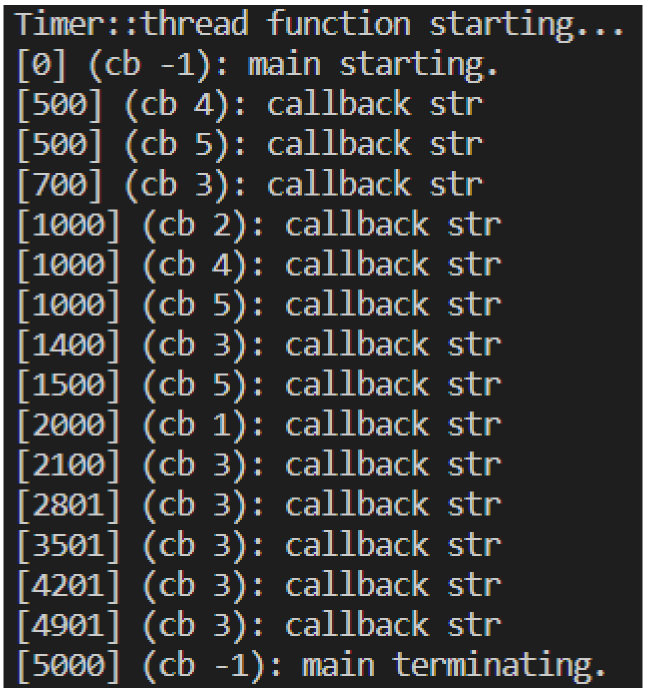
## How to Build and Test

In order to build and test the Timer class, the Timer.h file should be included and the ITimer.h should also be present. In the main program, an instance should be created and the desired type of timer should be created by using one of the four overloaded registerTimer functions.

In the homework submission, the test program is build with the name “test”.

## Test Results

Here is the test results of the provided test code on the homework PDF:



```
Timer::thread function starting...
[0] (cb -1): main starting.
[500] (cb 4): callback str
[500] (cb 5): callback str
[700] (cb 3): callback str
[1000] (cb 2): callback str
[1000] (cb 4): callback str
[1000] (cb 5): callback str
[1400] (cb 3): callback str
[1500] (cb 5): callback str
[2000] (cb 1): callback str
[2100] (cb 3): callback str
[2801] (cb 3): callback str
[3501] (cb 3): callback str
[4201] (cb 3): callback str
[4901] (cb 3): callback str
[5000] (cb -1): main terminating.
```