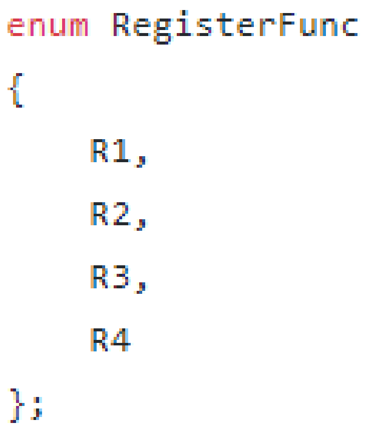
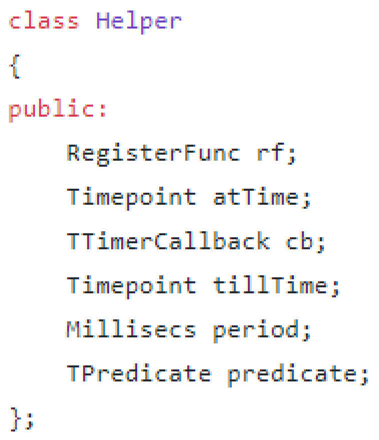
CSE 437 REAL TIME SYSTEM ARCHITECTURES
HOMEWORK 1 REPORT

- **The requirements of your timer, containing constraints and assumptions.**
  - First of all, to implement this timer class, some necessary **C++11** features are used for timing, synchronous and thread creating.
  - In order to implement this Timer, I needed to create a class that is called **Helper** for holding necessary parameters of registerTimer functions and an enum type that is called **RegisterFunc** to verify the function type. These class and enum type are for help to provide queuing system for the all timers function.
  - Also I need to use **a mutex**, **a condition variable**, **an atomic boolean variable** and **lock/notify (lock_guard, lock_unique and notify_all)** system to avoid from race condition and provide synchronous timer system.
- **The design of the timer.**
  - First of all, I create a class and an enum type for hold parameters of the functions that I had to implement.

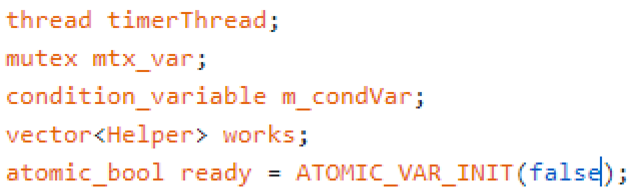


```
enum RegisterFunc
{
    R1,
    R2,
    R3,
    R4
};
```



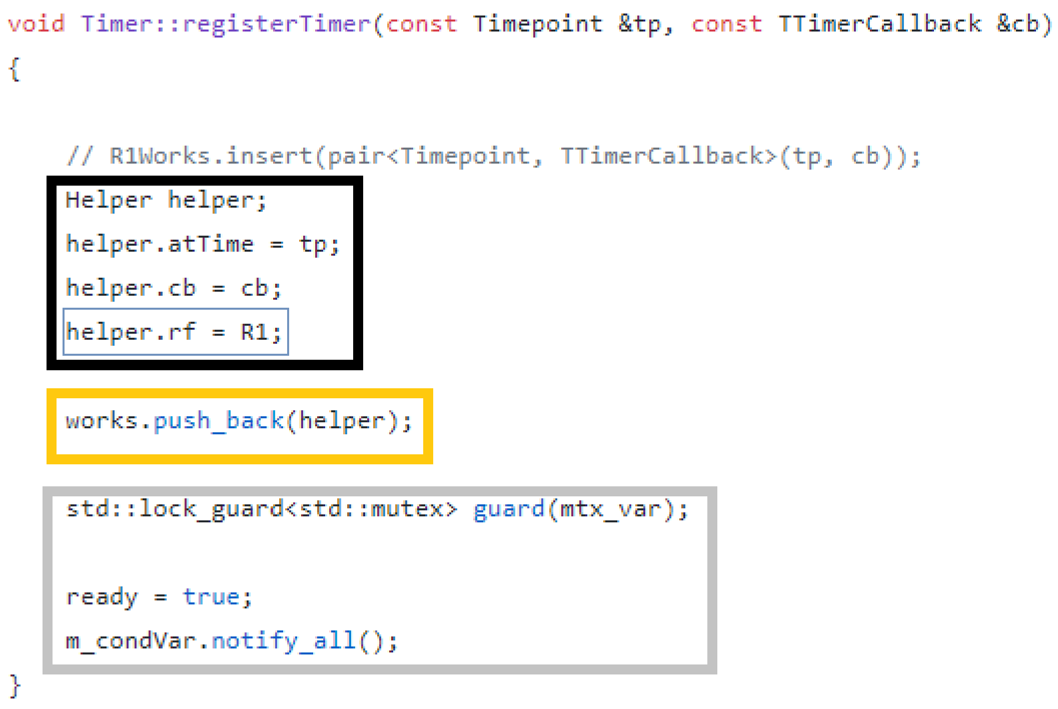
```
class Helper
{
public:
    RegisterFunc rf;
    Timepoint atTime;
    TTimerCallback cb;
    Timepoint tillTime;
    Millisecs period;
    TPredicate predicate;
};
```

  - The enum type is for verifying which RegisterTimer is that.
  - The Helper class tells the program which registerTimer function will run and with what parameters.
    - For example; if the first registerTimer will be run then **rf**, **atTime** and **cb** variable will be initialized according to given parameters of registerTimer function.
  - Whenever the called the registerTimer function, a Helper object created/initialized then added into a vector. After even one object added into the vector, the thread that in the Timer class starts working.
  - To avoid race condition and provide synchronous, I needed to use these in Timer class:



```
thread timerThread;
mutex mtx_var;
condition_variable m_condVar;
vector<Helper> works;
atomic_bool ready = ATOMIC_VAR_INIT(false);
```

- Firstly, I initialize the timerThread in constructor. The thread waits for the atomic boolean variable (**ready**) to be become true in order to make progress on **works** data.
- That atomic variable become true when registerTimer function is called and the works object is filled. After that variable become true then I called the notify_all function on that atomic variable in order to timerThread can start to work.
- You can see the implementation of the first registerTimer function as example;



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

    // R1Works.insert(pair<Timepoint, TTimerCallback>(tp, cb));
    Helper helper;
    helper.atTime = tp;
    helper.cb = cb;
    helper.rf = R1;

    works.push_back(helper);

    std::lock_guard<std::mutex> guard(mtx_var);

    ready = true;
    m_condVar.notify_all();
}
```

- **How to build and test the timer project?**
  - To build the project with, go to the directory that **CmakeList.txt** file exist, then run the commands:
    - **$cmake .**
    - **$cmake –build .**
  - Then an executable file that is called **timer** is created, you can run this file with this command:
    - **$./timer**
  - To test the project with your test functions, you have to **replace** the file that **Test_main.cpp** with yours or add your one into the **CmakeList.txt** file if you want to change the main file. Otherwise you can just change the context of the **Test_main.cpp** file with your implementation.