**“Timer.h”**

**(Because of different format code may appear different)**

#ifndef TIMER\_H

#define TIMER\_H

#include <thread>

#include <chrono>

class Timer

{

std::thread Thread; // used for asynchronous code execution without blocking main thread

bool Alive = false; // check if timer is running

long CallNumber = -1L; // how many times we would like to call a certain function

long repeat\_count = -1L; // count amount of times a certain function has been called

std::chrono::milliseconds interval = std::chrono::milliseconds(0); //interval between function calls, default is 0

std::function<void(void)> funct = nullptr; // function that takes nothing and returns nothing

void SleepAndRun()

{

std::this\_thread::sleep\_for(interval); //pause thread for certain time interval

if (Alive)

Function()(); // double parenthesis - first calls Function and second calls function that Function returns

}

void ThreadFunc()

{

if (CallNumber == Infinite)

while (Alive)

SleepAndRun();

else

while (repeat\_count--)

SleepAndRun();

}

public:

static const long Infinite = -1L;

Timer() {};

Timer(const std::function<void(void)> &f) : funct(f) {};

Timer(const std::function<void(void)> &f, const unsigned long &i, const long repeat = Timer::Infinite) : funct(f), interval(std::chrono::milliseconds(i)), CallNumber(repeat) {};

void Start(bool Async = true)

{

if (IsAlive()) // check if timer is running, if not set to run

return;

Alive = true;

repeat\_count = CallNumber; // set repeat to how many times we need to run

if (Async) // if thread is not being blocked

Thread = std::thread(&Timer::ThreadFunc, this);

else

this->ThreadFunc();

}

void Stop()

{

Alive = false; // set timer to stop running

Thread.join();

}

void SetFunction(const std::function<void(void)> &f) // sets the func to be executed

{

funct = f;

}

bool IsAlive() const { return Alive; } // check if timer is running

void RepeatCount(const long r) // sets number of calls

{

if (Alive)

return;

CallNumber = r;

}

long GetLeftCount() const { return repeat\_count; } // see how many iterations are left

long RepeatCount() const { return CallNumber; } // total number of occurences to be done

void SetInterval(const unsigned long &i)

{

if (Alive)

return;

interval = std::chrono::milliseconds(i);

}

unsigned long Interval() const { return (unsigned long)interval.count(); } // fetching interval to long type

const std::function<void(void)> &Function() const

{

return funct; // returns a function to be called right after

}

};

#endif // TIMER\_H