```
// HeartInfra.cpp - HEART RATE MONITOR INFRASTRUCTURE
//
// MODULE INDEX
// NAME                 CONTENTS
// AddTime              Add a period in milliseconds to a timespec
// CmpTime              Compare timespecs
// heart_sleep  Sleep for a specified number of milliseconds
// heart_beat   Determine whether a heartbeat is in progress at the
//              current time
//
// MAINTENANCE HISTORY
// DATE         PROGRAMMER AND DETAILS
// 06-10-15     JS      Original
//
//-----------------------------------------------------------------------------

#include <ctime>                // C-style system time functions
#include <cstring>              // C-style string manipulation functions
#include <cstdlib>              // C-style standard library
#include <iostream>             // C++ I/O stream declarations
using namespace std;    // Expand the standard namespace
#include "heartio.h"    // Heart rate infrastructure declarations

//-----------------------------------------------------------------------------

// DEFINITIONS

const long ON_BOUNCE_TIME = 20;
                                // Switch on bounce time in milliseconds
const long ON_STEADY_TIME = 200;
                                // Switch on steady time in milliseconds
const long OFF_BOUNCE_TIME = 40;
                                // Switch off bounce time in milliseconds
const long OFF_STEADY_TIME = 740;
                                // Switch off steady time in milliseconds

//-----------------------------------------------------------------------------

// STATE ENUMERATOR

enum State_t {
        ON_BOUNCE,              // Switch on bounce
        ON_STEADY,              // Switch on steady
        OFF_BOUNCE,             // Switch off bounce
        OFF_STEADY              // Switch off steady
};

//-----------------------------------------------------------------------------

// INITIAL TIME

struct IniTime_t : public timespec {
        IniTime_t () { clock_gettime (CLOCK_MONOTONIC, this); }
};
IniTime_t               iniTime;

//-----------------------------------------------------------------------------

// GLOBAL DATA

IniTime_t               lastWakeup;     // Last wakeup time
IniTime_t               nextTransition; // Next transition time
State_t                 currentState = OFF_STEADY; // Current state
```

```
//------------------------------------------------------------------------

// ADD A PERIOD IN MILLISECONDS TO A TIMESPEC

void
AddTime (
        struct timespec *ts,            // Time specification
        long            ms)             // Period in milliseconds
{
        ts->tv_sec += ms/1000;
        ts->tv_nsec += (ms%1000) * 1000000;
        if (ts->tv_nsec > 1000000000) {
                ts->tv_sec ++ ;
                ts->tv_nsec -= 1000000000;
        }
}

//------------------------------------------------------------------------

// COMPARE TIMESPECS

int
CmpTime (
        const struct timespec *ts1,     // First time specification
        const struct timespec *ts2)     // Second time specification
{
        if (ts1->tv_sec > ts2->tv_sec) return 1;
        if (ts1->tv_sec < ts2->tv_sec) return -1;
        if (ts1->tv_nsec > ts2->tv_nsec) return 1;
        if (ts1->tv_nsec < ts2->tv_nsec) return -1;
        return 0;
}

//------------------------------------------------------------------------

// SLEEP FOR A SPECIFIED NUMBER OF MILLISECONDS

void
heart_sleep (
        int             ms)             // Number of milliseconds to sleep
{
        struct timespec remainder;      // Remaining interval

        AddTime (&lastWakeup, ms);
        clock_nanosleep (CLOCK_MONOTONIC, TIMER_ABSTIME, &lastWakeup,
                &remainder);
}

//------------------------------------------------------------------------

// DETERMINE WHETHER A HEARTBEAT IS IN PROGRESS AT THE CURRENT TIME

bool
heart_beat ()
{
        IniTime_t       currentTime;    // Current time
        bool            beatInProg;     // Beat in progress flag

        // Advance the next transition time until it is greater than
        // the current time

        while (CmpTime (&nextTransition, &currentTime) <= 0) {
                switch (currentState) {
                case ON_BOUNCE:
                        AddTime (&nextTransition, ON_STEADY_TIME);
```

```
                        currentState = ON_STEADY;
                        break;
                case ON_STEADY:
                        AddTime (&nextTransition, OFF_BOUNCE_TIME);
                        currentState = OFF_BOUNCE;
                        break;
                case OFF_BOUNCE:
                        AddTime (&nextTransition, OFF_STEADY_TIME);
                        currentState = OFF_STEADY;
                        break;
                case OFF_STEADY:
                        AddTime (&nextTransition, ON_BOUNCE_TIME);
                        currentState = ON_BOUNCE;
                        break;
                default:
                        cerr << "Error: bad state\n";
                        exit (1);
                }
        }

        // Return a value consistent with the current state

        switch (currentState) {
        case ON_BOUNCE:
                beatInProg = rand() % 2;
                break;
        case ON_STEADY:
                beatInProg = 1;
                break;
        case OFF_BOUNCE:
                beatInProg = rand() % 2;
                break;
        case OFF_STEADY:
                beatInProg = 0;
                break;
        default:
                cerr << "Error: bad state\n";
                exit (1);
        }
        return beatInProg;
}
```