Project: A simple clock
My own project, no frills graphics (or lack there of), multi-threaded project
(clock) to display a running time, a possible alarm (and its state) and a possible countdown timer (and its state). Local time is used for the acquistion of the date/time.

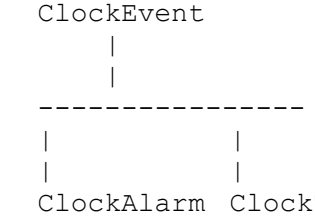
Description:
A simple, no frills "alarm clock" that displays the time, whether an alarm is set
and if the alarm is ringing, and a timer, that when set, will count down and
expire (ringing) when it reaches 0.

Valid Commands when clock (aSimpleClock) is running:
A XX:YY (set Alarm to hour XX (0-23) Minute YY (0-59))
T ttt (set countdown timer to ttt seconds)
O A (turn off Alarm before it expires (rings)
O T (turn off Timer before it counts down to 0)
E (turns off ringing Alarm and/or Timer)
Cntrol-C (terminate clock)
NOTE: All other commands (unknown or incorrectly entered) ignored

Build Instructions:
1. change directory into build directory (cd build)
2. run cmake .. (will use the file CMakeLists.txt file in parent directory
3. If successful, then type make and hit enter key to compile source and build executable aSimpleClock
4. ./aSimpleClock (from build directory) for 'simpliest' invocation or ./aSimpleLink -h for optional arguments to pass in.

Code Structure
4 threads
clockPulseGenerator : Generates 1 seconds 'ticks', monitors for 'time drift'
clockPulseResponse : Responds to the issuance of a new 'tick' from clockPulseGenerator
clockUserInput : Responsible for validating and and implementing user requests, changes ClockSettings
main : validate start condition(s), create threads, join threads, write to file upon termination

Class structure:



ClockSettings
ClockAlarm
ClockTimer

NOTE:
A selectable debug output class (dbgMsg.h) was used in the early development of this project.
It is included in the source code but not referenced in the delivered project.

Rubic

The project demonstrates an understanding of C++ functions and control structures. (all over)

The project reads data from a file and process the data, or the program writes data to a file. (main.cpp, lines 472 - 483)

The project accepts user input and processes the input (main.cpp: clockUserInputThread, function ProcessCommandLineArgs)

The project uses Object Oriented Programming techniques (clockAlarm.[h|cpp], clockTimer.[h|cpp], main.cpp)

Classes use appropriate access specifiers for class members (clockAlarm.[h|cpp], clockTimer.[h|cpp], main.cpp)

Class constructors utilize member initialization lists (clockAlarm.cpp: line 3, clockTimer.cpp: line 3, line 7)

Class abstracts implementation details from their interface (clockAlarm.[h|cpp], clockTimer.[h|cpp], clockSettings.[h|cpp])

Classes abstracts behavior (clockAlarm.[h|cpp], clockTimer.[h|cpp], clockSettings.[h|cpp])

Classes follow an appropriate inheritance hierarchy (clockElement.[h|cpp], clockAlarm.[h|cpp], clockTimer.[h|cpp])

Overloaded functions allow same function to operate on different parameters (clockTimer.cpp: line 3, line 7)

Derived calls functions override virtual base class functions (clockAlarm.[h|cpp], clockTimer.[h|cpp]

Templates generalize functions in the project (main.cpp, lines 46-50, function inRange)

The project makes use of references in function declarations. (clockAlarm.[h|cpp]

```
                                void
setClockAlarmTimeSecondsInDay(const unsigned int & alarmTimeInSeconds);
                                  void checkAlarm( const time_t &
secondsSinceEpoch );
```

The project uses destructors appropriately (clockSettings.h)

```
                           std::unique_ptr<ClockAlarm> clockAlarmPtr;
                           std::unique_ptr<ClockTimer> clockTimerPtr;
```

The project uses scope / RAII where appropriate (clockSettins.[h|cpp]

```
                                 std::unique_ptr<ClockAlarm>
clockAlarmPtr;
                            std::unique_ptr<ClockTimer> clockTimerPtr;
```

The project uses smart pointers instead of raw pointers

```
                                  std::unique_ptr<ClockAlarm>
clockAlarmPtr;
                                     std::unique_ptr<ClockTimer>
clockTimerPtr;
```

The projects uses multithreading (main.cpp, threads clockPulseGenerator, clockPulseResponse, clockUserInput)

A promise and future is used in the project (main.cpp, lines 455-456, 459, 476)

A mutex or lock is used in the file (main.cpp, line 34 (declared ClockSettingsLock, used in threads listed above)

A condition variable is used in the project (main.cpp, lines 246, 275, declared in class ClockSettings)