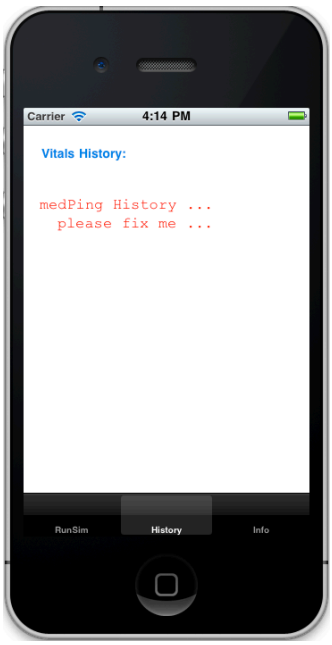
Keeping a HISTORY of a Patient's VITAL SIGNS *or*
"Uncle Foo is counting on YOU!"

The amazing medPing© is not quite ready for prime time. One serious limitation is that it is unable to keep track of a patient's *history* of vital signs. This assignment is a quick mock-up of how one might save a history of a patient's vital signs in an **array of structs**. You know your overweight Uncle Foo? … medPing can help monitor his health.



Work on this assignment in small byte-size pieces.

(0) Download my "starting kit" file from onCourse and study the file: `History_medPing_Main.cpp`

(1) **Study the code** in **`History_medPing_Main.cpp`**. Run it to get a feel of what it can do so far (not much in the way of keeping a history of a patient's vital signs … but hey, that's why you are paid the big bucks, right?) ***Uncle Foo is counting on YOU!***

(2) Study the way the simulation handles time. Read the API directions for the time.h functions:
http://www.cplusplus.com/reference/clibrary/ctime/

NOTE to Windows OS users! You must make the follow changes:
(i) *of course, each .cpp file needs:* **`#include "stdafx.h"`**
(ii) **`#include <windows.h>`**
then later, when you want to "sleep", use the Windows-specific `Sleep()` *function:*
(iii) **`Sleep(seconds*1000);`** `// see code`
(iv) *of course where needed,* **`system("PAUSE");`**

(3) Get comfortable with the DATA STRUCTURE that we'll use: an **array of struct**s. Draw pictures!

```
const long MAX_HISTORY = 5;   // can store upto (last) 5 sets of vital signs

struct oneVitalHistoryRecord
{
        double      bodyTemp_F;
        short       pulseRate;
        // :
        // you need more here ...
        // :

};

//======= DATA STRUCTURE ====================================
// to hold patient's history of vital signs
oneVitalHistoryRecord     vitalHistory[MAX_HISTORY];
// hmr (how many really) vital signs  0 <= hmr < MAX_HISTORY
long hmr = 0;
//============================================================
```

(4) Practice "pinging" the medPing chip and retrieving some vital signs. Don't worry about storing them in your history array yet … just have your code request values from the medical chip and print lil' messages to your "cell" screen.
(5) Study the `printAllVitalRecords()` function. Once you understand how this function works, that will help you write `AddHistoryRecord()`. Implement the function to add each new set of vital signs to your data structure.
(6) As you add more functionality to your program, *continually* update your **documentation** (see "Date last modified" at the top). For example:

```
02/12/2014 (your initials) Completed AddHistoryRecord() but
           there is a bug: it is not recording the initial set
           of vital signs?
02/13/2014 (your initials) Bug found; hmr was incremented
           wrong; moving on to document PRE/POST on functions.
```

(7) Consider an appropriate strategy for dealing with the situation when the history array is full, but a new set of vital signs is to be added. Document and implement your strategy. How can you test this?

(8) Add an additional field to your `struct oneVitalHistoryRecord` to store the "time" that the vital signs were recorded; you can store the time as saved in the `time_t` variable called `now`. *What exactly is this value? Make sure you document that.*

(9) Store the given time (nSecs) associated with each set of vital signs in your data structure.

(10) Implement the search function, `FindVitalRecord()`. Note that this function doesn't *delete* any records, rather it just finds the record in your data structure at the time `nSecs`.

```
long FindVitalRecord(long nSecs, const oneVitalHistoryRecord vitalHistory[ ], long hmr )
/*
PRE: nSecs(assigned) with a result of time() function and
       hmr(assigned) and 0 <= hmr < MAX_HISTORY and
       oneVitalHistoryRecord[0..(hmr-1)](assigned)
POST: if nSecs found within oneVitalHistoryRecord[]
       then RETURNS index of array cell where found
       otherwise RETURNS "NOT_FOUND" (constant indicating "not found")
*/
```

(11) Implement the function `DeleteHistoryRecord()`. You write the PRE and POST for this function.

A sample output from using `FindVitalRecord()` and `DeleteHistoryRecord()` is shown below:

```
:
:
------------------------------
RECORD [00]
  raw time:     1233286115
  Local time and date: Thu Jan 29 19:28:35 2009
  temp(F):      98.6
  pulse(BPM):     86

RECORD [01]
  raw time:     1233286119
  Local time and date: Thu Jan 29 19:28:39 2009
  temp(F):      100.1
  pulse(BPM):     83


SIMULATION OVER.

DELETE a record; Enter a RAW TIME: 1233286119


(Now show all remaining records after delete)
------------------------------
RECORD [00]
  raw time:     1233286115
  Local time and date: Thu Jan 29 19:28:35 2009

  temp(F):      98.6
  pulse(BPM):     86
```

Here is a sample of part of the grade key I'll use to help you complete proper documentation:

4 - **Source Code**: **`History_medPing_Main.cpp`** printed in Landscape and stapled hardcopy submitted in class on Friday, Feb. 28th.

STYLE:
3/2/1 - INDENTation is (excellent/good/fair)
3/2/1 - use of BLANK LINES between sections of code is (excellent/good/fair)
1 - VARIABLES have good names (use `camelCase` often)

DOCUMENTATION:
1 - Name and Date-Last-Revised listed
1 - Summary/Purpose given
1 - INPUT explained (mention data type(s) from *where?*)
1 - OUTPUT explained *to where?*

1 - each variable has an associated comment (including local variables in functions)

5/3/1 - inout && **Pre/Post conditions** listed for all/most/few functions