First:________________ Last:______________________ Instructor (Circle):**NT MT ME JV RY**

Scoring The correct output values are shown in the figure on the right. Your grade will be based both on the numerical results returned by your program and on your programming style. In particular, write code that is easy to understand, easy to debug, easy to change. Please employ good labels, pretty structure, and good comments.

Performance Score= Run by TA		TA:

```
UART #1
 Exam2_HeartMonitor
*****Test of Size*****
 Yes, Your Size= 3, Score = 5
 Yes, Your Size= 6, Score = 10
 Yes, Your Size= 8, Score = 15
 Yes, Your Size= 1, Score = 20
 Yes, Your Size= 0, Score = 25
*****Test of InZone*****
 Yes, Your InZone= 2, Score = 29
 Yes, Your InZone= 2, Score = 33
 Yes, Your InZone= 1, Score = 37
 Yes, Your InZone= 3, Score = 40
 60 to 200 are correct, Score = 50
*****Test of Range*****
 Yes, Your= 11, Score = 55
 Yes, Your= 22, Score = 60
 Yes, Your= 140, Score = 65
 Yes, Your= 130, Score = 70
 Yes, Your= 0, Score = 75
*****Test of FindResting*****
 Rate = {60, 61, 62, 62, 60 Yes, Your= 61
 Rate = {68, 80, 80, 79, 59, 78, 68, 77}
 Rate = {59, 50, 55, 84, 61, 58, 51, 60}
 Rate = {88, 52, 54, 65, 57, 52, 85, 62,
 Rate = {} Yes, Your= 255
Final Score = 100
End of Exam2_HeartMonitor
```

I promise to follow these rules

This is a closed book exam. You must develop the software solution using the **Keil uVision** simulator. You have 60 minutes, so allocate your time accordingly. You must bring a laptop and are allowed to bring only some pens and pencils (no books, cell phones, hats, disks, CDs, or notes). Each person works alone (no groups). You have full access to **Keil uVision**, with the **Keil uVision** help. You may use the Window's calculator. You sit in front of a computer and edit-assemble-run-debug the programming assignment. You do NOT have access the book, internet or manuals. You may not access your network drive or the internet. You are not allowed to discuss this exam with other EE319K students until Friday evening.

The following activities occurring during the exam will be considered scholastic dishonesty:

1) running any program from the PC other than **Keil uVision**, or a calculator,
2) communicating with **anyone else** except for the instructors **by any means** about this exam until Friday
3) using material/equipment other than a pen/pencil,
4) hard-coding so it outputs answers that give points without actually solving the problem,
5) modifying anything other than **Exam2CPart.c and Exam2AsmPart.c**

Students caught cheating will be turned to the Dean of Students.

Your signature is your promise that you have not cheated and will not cheat on this exam, nor will you help others to cheat on this exam:

Signed: ____ November 8, 2017

Procedure

First, you will log onto the computer and download files from the web as instructed by the TAs.

Web site: **http://users.ece.utexas.edu/~valvano/Volume1/Exam2HR**
User: jon
Password: 42

UNZIP the folder placing it **ON THE DESKTOP**. You are not allowed to archive this exam. Within **Keil uVision** open the project, put your name on the first comment line of the file **Exam2CPart.c**. Before writing any code, please build and run the system. You should get output like the figure above (but a much lower score). You may create backup versions of your program. If you wish to roll back to a previous version, simply open one of the backup versions.

My main program will call your functions multiple times, and will give your solution a performance score of 0 to 100. *You should not modify my main program or my example data.* Each time you add a block of code, you should run my main program, which will output the results to the **`UART#1`** window. To see the window execute: **View -> Serial Windows -> UART #1**. After you are finished, raise your hand and wait for a TA. The TA will direct you on how to complete the submission formalities. The TA will run your program in front of you and record your performance score on your exam cover sheet. The scoring page will not be returned to you.

Important Notes:

- Your functions should work for all cases given to it by the grader.
- The description of functions here is less detailed than that in the comments above the function in the source files. Refer to them before attempting your solution.

The exam has four parts a) through d), details of which are given in the starter code. Parts a and b are in assembly (**`Exam2AsmPart.s`**); parts c and d are in C (**`Exam2CPart.c`**).

Part a) Write an **assembly** subroutine called **`Size`** that calculates the size of a 16-bit signed array of numbers. The variable-sized array is terminated with a -32768 (0x8000). Here is the prototype:

`uint32_t Size(const int16_t *pt);`

This function uses *call-by-reference* for its parameter, and MUST be written in assembly

Part b) Write an **assembly** function called **`InZone`** that determines if you are in the fat-burning exercise zone, and MUST be written in assembly. Here is the prototype:

`uint32_t InZone(uint32_t rest, uint32_t age, uint32_t HR);`

Part c) Write the function called **`Range`** in **`Exam2CPart.c`**. This function will find the range of twelve 8-bit signed numbers, and MUST be written in C, with C99 syntax.

Part d) Write the function called **`FindResting`** in **`Exam2CPart.c`**. This function will find the average resting heart rate from an array of measurements, and MUST be written in C, with C99 syntax.

Submission Guidelines:

- Log onto Canvas and submit your **Exam2CPart.c** and **Exam2AsmPart.s** source files into the Exam2 submission link. Be careful because only one submission will be allowed.

Memory access instructions

```
LDR    Rd, [Rn]        ; load 32-bit number at [Rn] to Rd
LDR    Rd, [Rn,#off]   ; load 32-bit number at [Rn+off] to Rd
LDR    Rd, =value      ; set Rd equal to any 32-bit value (PC rel)
LDRH   Rd, [Rn]        ; load unsigned 16-bit at [Rn] to Rd
LDRH   Rd, [Rn,#off]   ; load unsigned 16-bit at [Rn+off] to Rd
LDRSH  Rd, [Rn]        ; load signed 16-bit at [Rn] to Rd
LDRSH  Rd, [Rn,#off]   ; load signed 16-bit at [Rn+off] to Rd
LDRB   Rd, [Rn]        ; load unsigned 8-bit at [Rn] to Rd
LDRB   Rd, [Rn,#off]   ; load unsigned 8-bit at [Rn+off] to Rd
LDRSB  Rd, [Rn]        ; load signed 8-bit at [Rn] to Rd
LDRSB  Rd, [Rn,#off]   ; load signed 8-bit at [Rn+off] to Rd
STR    Rt, [Rn]        ; store 32-bit Rt to [Rn]
STR    Rt, [Rn,#off]   ; store 32-bit Rt to [Rn+off]
STRH   Rt, [Rn]        ; store least sig. 16-bit Rt to [Rn]
STRH   Rt, [Rn,#off]   ; store least sig. 16-bit Rt to [Rn+off]
STRB   Rt, [Rn]        ; store least sig. 8-bit Rt to [Rn]
STRB   Rt, [Rn,#off]   ; store least sig. 8-bit Rt to [Rn+off]
PUSH   {Rt}            ; push 32-bit Rt onto stack
POP    {Rd}            ; pop 32-bit number from stack into Rd
ADR    Rd, label       ; set Rd equal to the address at label
MOV{S} Rd, <op2>       ; set Rd equal to op2
MOV    Rd, #im16       ; set Rd equal to im16, im16 is 0 to 65535
MVN{S} Rd, <op2>       ; set Rd equal to -op2
```

Branch instructions

```
B    label   ; branch to label    Always
BEQ  label   ; branch if Z == 1   Equal
BNE  label   ; branch if Z == 0   Not equal
BCS  label   ; branch if C == 1   Higher or same, unsigned ≥
BHS  label   ; branch if C == 1   Higher or same, unsigned ≥
BCC  label   ; branch if C == 0   Lower, unsigned <
BLO  label   ; branch if C == 0   Lower, unsigned <
BMI  label   ; branch if N == 1   Negative
BPL  label   ; branch if N == 0   Positive or zero
BVS  label   ; branch if V == 1   Overflow
BVC  label   ; branch if V == 0   No overflow
BHI  label   ; branch if C==1 and Z==0  Higher, unsigned >
BLS  label   ; branch if C==0 or  Z==1  Lower or same, unsigned ≤
BGE  label   ; branch if N == V   Greater than or equal, signed ≥
BLT  label   ; branch if N != V   Less than, signed <
BGT  label   ; branch if Z==0 and N==V  Greater than, signed >
BLE  label   ; branch if Z==1 or N!=V  Less than or equal, signed ≤
BX   Rm      ; branch indirect to location specified by Rm
BL   label   ; branch to subroutine at label, return address in LR
BLX  Rm      ; branch to subroutine indirect specified by Rm
```

Interrupt instructions

```
CPSIE  I                ; enable interrupts  (I=0)
CPSID  I                ; disable interrupts (I=1)
```

Logical instructions

```
AND{S} {Rd,} Rn, <op2> ; Rd=Rn&op2    (op2 is 32 bits)
ORR{S} {Rd,} Rn, <op2> ; Rd=Rn|op2    (op2 is 32 bits)
EOR{S} {Rd,} Rn, <op2> ; Rd=Rn^op2    (op2 is 32 bits)
BIC{S} {Rd,} Rn, <op2> ; Rd=Rn&(~op2) (op2 is 32 bits)
ORN{S} {Rd,} Rn, <op2> ; Rd=Rn|(~op2) (op2 is 32 bits)
LSR{S} Rd, Rm, Rs      ; logical shift right Rd=Rm>>Rs  (unsigned)
```

```
LSR{S} Rd, Rm, #n      ; logical shift right Rd=Rm>>n   (unsigned)
ASR{S} Rd, Rm, Rs      ; arithmetic shift right Rd=Rm>>Rs (signed)
ASR{S} Rd, Rm, #n      ; arithmetic shift right Rd=Rm>>n  (signed)
LSL{S} Rd, Rm, Rs      ; shift left Rd=Rm<<Rs (signed, unsigned)
LSL{S} Rd, Rm, #n      ; shift left Rd=Rm<<n  (signed, unsigned)
```

Arithmetic instructions

```
ADD{S} {Rd,} Rn, <op2> ; Rd = Rn + op2
ADD{S} {Rd,} Rn, #im12 ; Rd = Rn + im12, im12 is 0 to 4095
SUB{S} {Rd,} Rn, <op2> ; Rd = Rn - op2
SUB{S} {Rd,} Rn, #im12 ; Rd = Rn - im12, im12 is 0 to 4095
RSB{S} {Rd,} Rn, <op2> ; Rd = op2 - Rn
RSB{S} {Rd,} Rn, #im12 ; Rd = im12 – Rn
CMP    Rn, <op2>       ; Rn – op2      sets the NZVC bits
CMN    Rn, <op2>       ; Rn - (-op2)   sets the NZVC bits
MUL{S} {Rd,} Rn, Rm    ; Rd = Rn * Rm       signed or unsigned
MLA    Rd, Rn, Rm, Ra  ; Rd = Ra + Rn*Rm    signed or unsigned
MLS    Rd, Rn, Rm, Ra  ; Rd = Ra - Rn*Rm    signed or unsigned
UDIV   {Rd,} Rn, Rm    ; Rd = Rn/Rm         unsigned
SDIV   {Rd,} Rn, Rm    ; Rd = Rn/Rm         signed
```

Notes `Ra Rd Rm Rn Rt represent 32-bit registers`

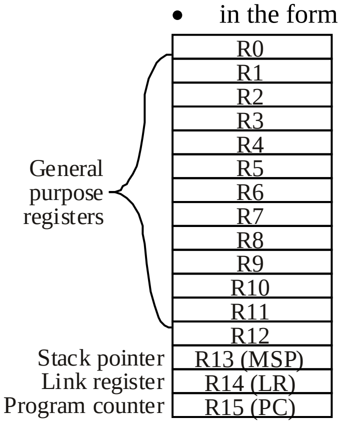
```
value   any 32-bit value: signed, unsigned, or address
{S}     if S is present, instruction will set condition codes
#im12   any value from 0 to 4095
#im16   any value from 0 to 65535
{Rd,}   if Rd is present Rd is destination, otherwise Rn
#n      any value from 0 to 31
#off    any value from -255 to 4095
label   any address within the ROM of the microcontroller
op2     the value generated by <op2>
```

Examples of flexible operand **`<op2>`** creating the 32-bit number. E.g., **`Rd = Rn+op2`**

```
ADD Rd, Rn, Rm         ; op2 = Rm
ADD Rd, Rn, Rm, LSL #n ; op2 = Rm<<n  Rm is signed, unsigned
ADD Rd, Rn, Rm, LSR #n ; op2 = Rm>>n  Rm is unsigned
ADD Rd, Rn, Rm, ASR #n ; op2 = Rm>>n  Rm is signed
ADD Rd, Rn, #constant  ; op2 = constant
```

, where **X** and **Y** are hexadecimal digits:

- produced by shifting an 8-bit unsigned value left by any number of bits
- in the form **0x00XY00XY**
- in the form **0xXY00XY00**
- in the form **0xXYXYXYXY**



Condition code bits

N negative
Z zero
V signed overflow
C carry or unsigned overflow