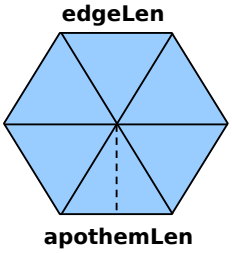
CS 218 – MIPS Assignment #1

Purpose: Become familiar with RISC Architecture concepts, the MIPS Architecture, and QtSpim (the MIPS simulator).
Points: 30

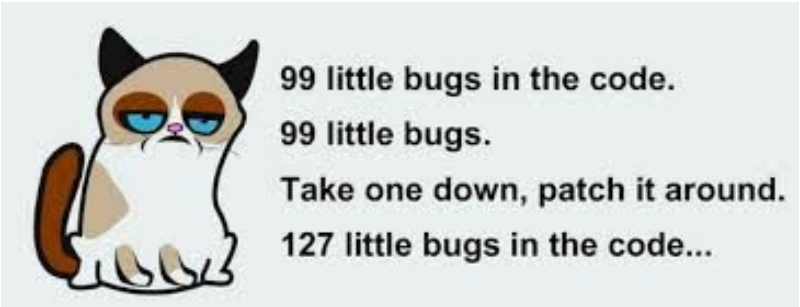
Assignment:

Write a MIPS assembly language program to calculate some geometric information for each regular hexagon[1] in a series of hexagons. Specifically, the program will find the area for each of the hexagon in a set of hexagons. The formula for regular hexagon area is as follows:



$$hexAreas[i] = 6 \times \frac{sideLens[i] \times apothemLens[i]}{2}$$

After the areas have been computed, the program should find the minimum, maximum, estimated median, sum, and average for the areas. The data is not sorted, so we will obtain the estimated median as follows. Since the list length is odd, the estimated median will be computed by summing the first, last, and middle values and then dividing by 3.



The program must display the results to the console window. Your code must perform the calculations and output the values from the array hexAreas. The output should look something like the following.

```
MIPS Assignment #1
Program to calculate area of each hexagon in a series of hexagons.
Also finds min, mid, max, sum, and average for the hexagon areas.


  1440   3822   7524   11484   15660   20700   23976
  31734   39204   43932   701838   731544   762294   779982
  798762   811008   829128   838080   865536   887784   1175346
  1223502   1268604   1305696   1339740   1412796   1495572   1547916

    [... display all numbers ...]

Hexagon min = ?
Hexagon emid = ?
Hexagon max = ?
Hexagon sum = ?
Hexagon ave = ?
```

1 For more information, refer to: https://en.wikipedia.org/wiki/Hexagon

Submission:

- All source files must assemble and execute with QtSpim/SPIM MIPS simulator.
- Submit source file
  - Submit a copy of the program source file via the on-line submission
- Once you submit, the system will score the project and provide feedback.
  - If you do not get full score, you can (and should) correct and resubmit.
  - You can re-submit an unlimited number of times before the due date/time (at a maximum rate of 5 submissions per hour).
- Late submissions will be accepted for a period of 24 hours after the due date/time for any given assignment. Late submissions will be subject to a ~2% reduction in points per an hour late. If you submit 1 minute - 1 hour late -2%, 1-2 hours late -4%, … , 23-24 hours late -50%. This means after 24 hours late submissions will receive an automatic 0.

Program Header Block

All source files must include your name, section number, assignment, NSHE number, and program description. The required format is as follows:

```
#  Name: <your name>
#  NSHE ID: <your id>
#  Section: <section>
#  Assignment: <assignment number>
#  Description: <short description of program goes here>
```

Failure to include your name in this format will result in a reduction of points.

Scoring Rubric

Scoring will include functionality, code quality, and documentation. Below is a summary of the scoring rubric for this assignment.

Criteria	Weight	Summary
Assemble	-	Failure to assemble will result in a score of 0.
Program Header	3%	Must include header block in the required format (see above).
General Comments	7%	Must include an appropriate level of program documentation.
Program Functionality (and on-time)	90%	Program must meet the functional requirements as outlined in the assignment. Must be submitted on time for full score.

Data Declarations

Use the following data declarations:

```
sideLens:
          .word      15,   25,   33,   44,   58,   69,   72,   86,   99,  101
          .word     369,  374,  377,  379,  382,  384,  386,  388,  392,  393
          .word     501,  513,  524,  536,  540,  556,  575,  587,  590,  596
          .word     634,  652,  674,  686,  697,  704,  716,  720,  736,  753
          .word     107,  121,  137,  141,  157,  167,  177,  181,  191,  199
          .word     102,  113,  122,  139,  144,  151,  161,  178,  186,  197
          .word       1,    2,    3,    4,    5,    6,    7,    8,    9,   10
          .word     202,  209,  215,  219,  223,  225,  231,  242,  244,  249
          .word     203,  215,  221,  239,  248,  259,  262,  274,  280,  291
          .word     251,  253,  266,  269,  271,  272,  280,  288,  291,  299
          .word    1469, 2474, 3477, 4479, 5482, 5484, 6486, 7788, 8492

apothemLens:
          .word      32,   51,   76,   87,   90,  100,  111,  123,  132,  145
          .word     634,  652,  674,  686,  697,  704,  716,  720,  736,  753
          .word     782,  795,  807,  812,  827,  847,  867,  879,  888,  894
          .word     102,  113,  122,  139,  144,  151,  161,  178,  186,  197
          .word    1782, 2795, 3807, 3812, 4827, 5847, 6867, 7879, 7888, 1894
          .word     206,  212,  222,  231,  246,  250,  254,  278,  288,  292
          .word     332,  351,  376,  387,  390,  400,  411,  423,  432,  445
          .word      10,   12,   14,   15,   16,   22,   25,   26,   28,   29
          .word     400,  404,  406,  407,  424,  425,  426,  429,  448,  492
          .word     457,  487,  499,  501,  523,  524,  525,  526,  575,  594
          .word    1912, 2925, 3927, 4932, 5447, 5957, 6967, 7979, 7988

hexAreas:
          .space    436

len:      .word     109

haMin:    .word     0
haEMid:   .word     0
haMax:    .word     0
haSum:    .word     0
haAve:    .word     0

LN_CNTR   =  7
```

Note, the **`.space 436`** directive reserves 436 bytes which will store 109 words (109*4).