Prepare your answers to the following questions in a plain text file. Submit your file to the Curator system by the posted deadline for this assignment. No late submissions will be accepted.

You will submit your answers to the Curator System (www.cs.vt.edu/curator) under the heading HW3.

For the following five questions, consider the problem of implementing a program that could take a list of nonnegative integers and produce a histogram showing how many of the integers fall into each of the equivalence classes modulo 5. For example, given the integers:

19, 13, 2, 7, 11, 5, 23, 29, 17, 3, 47, 21, 84, 77, 14, 92, 36, 41, 32, 9

we would want the following histogram:

0: 1
1: 4
2: 7
3: 3
4: 5

The solution will consist of a collection of four functions, whose C-style interfaces might look like this:

```c
main()
// calls mod5_histogram() to produce and print the histogram
mod5_histogram(int list[], int Sz)
// calls make_histogram() and then calls prnt_histogram()
make_histogram(int list[], int Sz, int histo[])
// process list[] and compute values in histo[]
prnt_histogram(int Sz, int histo[])
// display histogram, as formatted above, to console
```

Of course, we are interested in a MIPS assembly implementation, not a C implementation.

1. [15 points] Draw a diagram of the activation record that main would need to create before calling mod5_histogram. Assume that main labels its relevant declared variables with the same names as used above for the function parameters.

2. [15 points] Draw a diagram of the activation record that mod5_histogram would need to create before calling make_histogram. Again, assume the caller uses the names shown above for its relevant declared variables.

3. [15 points] One possible implementation of mod5_histogram is outlined below. Note that it is legal to have multiple data directives; the effect is equivalent to combining them all into a single section.

```assembly
mod5_histogram:
.data
histo: .word 0, 0, 0, 0, 0       # space for mod5 histogram

.text
bltz   $a1, exit           # quit on invalid list
exit: jr      $ra
```

Another approach is shown below. In this version, mod5_histogram allocates enough space on the stack to store the histogram array, and then passes the address of that space to the procedures it calls.
mod5_histogram:

```
addi $sp, $sp, -20       # allocate space for histogram on stack
sw $zero, 16($sp)      # initialize entries to zero
sw $zero, 12($sp)
sw $zero,  8($sp)
sw $zero,  4($sp)
sw $zero,  0($sp)
bltz $a1, exit      # quit on invalid list
	// interesting but irrelevant stuff omitted . . .
addi $sp, $sp, 20        # deallocate space for histogram
exit:   jr $ra
```

Is there any reason to prefer one of these approaches over the other? If so, give a brief but clear explanation of why.

4. [25 points] Here is a possible implementation of the missing portion of the second version of mod5_histogram given above.

```
make:   or $a2, $sp, $zero     # pass address of histogram
        jal make_histogram
print:  or $a1, $sp, $zero     # pass address of histogram
        li $a0, 5              # pass size of histogram
        jal prnt_histogram
```

a) Explain the logical error(s) that prevent the implementation of the missing portion shown above from working properly.

b) Write a corrected version of the missing portion of mod5_histogram that will work properly. You may add new instructions, or remove any of the instructions given in this question, but you may not modify any of the instructions given in the relevant part of question 5.

5. [30 points] Write an implementation of the MIPS assembly procedure _make_histogram_. Your implementation must conform to the interface given earlier, and to the (corrected) second version of _mod5_histogram_. Be sure to comment your solution, and to test it with SPIM. Solutions that are not commented as discussed in class may be assigned a score of zero.