CS 218 - Assignment #6

Purpose: Become familiar with data conversion, addressing modes, and assembly language macro's.
Points: 100

Background:

The Nonary[1] numbering system (also known as base-9) is a positional notation numeral system using none (9) as its base. For base 10, the number fifteen, written as "15" in the base ten numbering system which means "1 sets of ten and 5 units". For base 9, that quantity is instead written as "16" in Nonary notation or base nine which means "1 set of 9 and 6 units". For example:

base-10	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...
base-9	1	2	3	4	5	6	7	8	10	11	12	13	14	15	16	...

Assignment

Write an assembly language program to convert ASCII/nonary string to integers and to convert integers to ASCII/nonary strings. The main will display the strings to the screen. Using the provided template, the program has a series steps as follows:

1. Write the code to convert a string of ASCII digits representing a nonary value into an integer (double-word sized). This code should be placed in the provided main at the marked location (step #1) and will convert the string **`aNonaryLength`** (nonary representation) into an integer stored in the variable **`length`**. This should ***not*** be a macro.

2. Convert the code from step #1 into a macro, ***aNonary2int***, which is called multiple times in the next part of the provided template. The empty macro shell is at the top of the provided template at the marked location (step #2). Once the string is converted to an integer, the volume of a cube can be calculated and the **`volumes[]`** array populated. The formula for calculating the volume of a cube is:

$$\boldsymbol{volumes[i] \ = \ edge[i]^3}$$

3. Add the code to compute the statistics; sum, average, minimum, and maximum. This will read the **volumes[]** array (when populated). *Note*, you will not be able to test this code until step #2 is completed.

4. Write the code to convert an integer into a string of ASCII digits representing the nonary value (NULL terminated) This code should be placed in the provided main at the marked location (step #4) and will convert the integer stored in the variable **`cubeSum`** into a string **`cubeSumString`** (ASCII/nonary representation). The ASCII version of the number should be **`STR_LENGTH`** (globally available constant) characters (including the NULL), right justified with the appropriate number of leading blanks. Refer to the sample output for an example. This should ***not*** be a macro.

[1] For more information, refer to: https://en.wikipedia.org/wiki/Ternary_numeral_system#Compact_ternary_representation:_base_9_and_27

5. Convert the code from step #4 into a macro, ***int2aNonary***, which is called multiple times in the next part of the provided template. The empty macro shell is at the top of the provided template at the marked location (step #5).

The codeGrade is configured to test each step, 1-5, individually. As such, it is possible to upload and test the code after each step.

The provided main will also invoke a print macro, which will display the strings to the screen. The print macro does ***not*** perform any error checking, so the data must be correct in order for the display to work. *Note*, since the program displays the results to the screen, typing the program name (without the debugger), will display the results to the screen.

All data must be treated as ***unsigned***. As such, the DIV/MUL would be used (not IDIV/IMUL). The JA/JAE/JB/JBE must be used (as they are for unsigned data). You may assume valid/correct data. As such, no error checking is required. You may add additional variables as needed.

Debugging Tips

The most important step is to create an algorithm and document the algorithm in comments. This should be done ***before*** the code is written. Comment each part of the algorithm (so you can match the algorithm to the appropriate subset of code).

It is suggested that you develop a debugger input file first (based on previous ones) carefully verifying the debugger commands based on the specific data types.

Additionally, since macro's can be difficult to debug. To address this, the code for step 1 should be working before attempting step 2.

The code for a macro will not be displayed in the source window. In order to see the macro code, display the machine code window (**View → Machine Code Window**). In the window, the machine code for the instructions are displayed. The step and next instructions will execute the entire macro. In order to execute the macro instructions, the **stepi** and **nexti** commands must be used (which are only used for macro's).

To help check results, an on-line base conversion is available at the following URL:
http://www.cleavebooks.co.uk/scol/calnumba.htm.

Debugger Commands:

Below is an example of some of the commands to display a few of the variables within DDD.

```
x/dw &length
x/40dw &perimsArray
```

Note, in DDD, select **View → Execution Window** to display a window that shows the output.

Example Output:

Below is an example output of the program.

```
ed-vm%
ed-vm% ./ast06
-----------------------------------------------------
CS 218 - Assignment #6
Cube Calculations

Cube Volumes:
         13630            58821           114478
        273201           210418           604858
       1886630          2767221          3300111
       3761761          6658000         11545251
      14466471         15541000         22772021
      26370261         30604008         37205868
      33745738         40187381         45667238
      55261808         60872638         65333561
      71000000         77088631         87878681
     101054401        111444778        114478000
     124223208        137071751        160415438
     171420630        180734108        208284458
     281178678        308062381        315015408
     341121758        345826368        401146858
     441420631        475338388        503426221
     526178158        600325751        706515081
     765218000        886002888

Cube Sum:     11112461667
Cube Ave:       173576135
Cube Min:           13630
Cube Max:       886002888

ed-vm%
ed-vm%
```

Submission:

- All source files must assemble and execute on Ubuntu with **`yasm`**.
- Submit source files
  - 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.
- 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: <4-digit-section>
;  Assignment: <assignment number>
;  Description: <short description of program goes here>
```

Failure to include your name in this format will result in a loss of up to 3%.

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. *Note*, ***must*** include comments for the conversion algorithm being used. Omitting these comments will zero the comments score.
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.