CS 218 - Assignment #6

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

Background:

The Senary[1] numbering system (also known as base-6) is a positional notation numeral system using six (6) as its base. The number fifteen (that is, the number written as "15" in the base ten numbering system) is instead written as "23" in Senary notation or base six (meaning "2 sets of 6 and 3 units", instead of "2 sets of ten and 3 units").

base-10	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**...**
base-6	**1**	**2**	**3**	**4**	**5**	**10**	**11**	**12**	**13**	**14**	**15**	**20**	**21**	**22**	**23**	**...**

Assignment

Write an assembly language program to convert ASCII/senary string to integers and to convert integers to ASCII/senary 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 senary 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 **`aSenaryLength`** (senary 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, ***aSenary2int***, 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 perimeter of a hexagon can be calculated and the **perimsArray[]** array populated. The formula for calculating the area of a square is:

$$\boldsymbol{perimsArray[i] = 6 \times sideLens[i]}$$

3. Add the code to compute the statistics; sum, average, minimum, and maximum. This will read the **perimsArray[]** 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 senary 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 **`perimSum`** into a string **`perimSumString`** (ASCII/senary 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/Senary

5. Convert the code from step #4 into a macro, ***int2aSenary***, 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.

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
Perimeter Calculations

Perimeters:
        230           430           1500
       2150          2340           2510
       3020          3220           3420
       4110          4320           4500
       5120          5320           5550
      10210         10340          10520
      20020         22130          23250
      31310         32040          35200
      41120         42000          44300
      52300         53210          55130
     102340        112030         123450
     202450        234500         245130
     312430        335250         412300
     543210

Perimeters Sum:     10202110
Perimeters Ave:        54131
Perimeters Min:          230
Perimeters Max:       543210

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.