Introduction

This program is significantly more difficult than previous programs, partially due to the more advanced concepts.  However, the Rubric (see below) now has a number of point deductions for not meeting requirements. It is not uncommon for a student to generate a program that meets the **Program Description** but violates several **Program Requirements**, causing a *significant* loss in points. Please carefully review the Rubric to avoid this circumstance.

The purpose of this assignment is to reinforce concepts related to usage of the runtime stack, proper modularization practices, use of the STDCALL calling convention, use of arrays, and Indirect Operands addressing modes (CLO 3, 4, 5).

1. Using Indirect Operands, Register Indirect, and/or Base+Offset addressing
2. Passing parameters on the runtime stack
3. Generating "random" numbers
4. Working with arrays

What you must do

**Program Description**

Write and test a MASM program to perform the following tasks (check the Requirements section for specifics on program modularization):

1. **Introduce**the program.
2. **Declare**global constants **ARRAYSIZE**, **LO**, and **HI**. **Generate ARRAYSIZE** random integers in the range from **LO** to **HI** (inclusive), storing them in consecutive elements of array **randArray**. (e.g. for **LO** = 20 and **HI** = 30, generate values from the set [20, 21, ... 30])
   * **ARRAYSIZE** should be initially set to 200,
   * **LO** should be initially set to 15
   * **HI** should be initially set to 50
   * *Hint:*Call **Randomize** once in **main** to generate a random seed. Later, use **RandomRange** to generate each random number.
3. **Display** the list of integers before sorting, 20 numbers per line with one space between each value.
4. **Sort**the list in ascending order (i.e., smallest first).
5. **Calculate**and **display**the median value of the sorted **randArray**, rounded to the nearest integer. (Using [Round Half Up Links to an external site.](https://simple.wikipedia.org/wiki/Rounding#Round_half_up)rounding)
6. **Display**the sorted **randArray**, 20 numbers per line with one space between each value.
7. **Generate**an array **counts** which holds the number of times each value in the range [**LO**, **HI**] ([15, 50] for default constant values) is seen in **randArray**, even if the number of times a value is seen is zero.  
   For example with **LO**=15, **counts[0]** should equal the number of instances of the value `15` in array. **counts[14]** should equal the number of instances of the value `29` in **randArray**. Note that some value may appear zero times and should be reported as zero instances.
8. **Display**the array **counts**, 20 numbers per line with one space between each value.

**Program Requirements**

1. The program **must** be constructed using procedures. *At least* the following procedures/parameters are required:  
   **NOTE:** Regarding the syntax used below...  
   **procName** {parameters: *varA* (value, input), *varB* (reference, output)} indicates that procedure **procName** must be passed *varA* as a value and *varB* as a reference, and that *varA* is an input parameter and *varB* is an output parameter. *You may use more parameters than those specified but try to only use them if you****need****them.*
   1. **main**
   2. **introduction** {parameters: *intro1* (reference, input), *intro2* (reference, input), ...)
   3. **fillArray** {parameters: *someArray* (reference, output)}  **NOTE: *LO*, *HI,* *ARRAYSIZE* will be used as globals within this procedure**.
   4. **sortList** {parameters: *someArray* (reference, input/output)} **NOTE: *ARRAYSIZE* will be used as a global within this procedure.**
   5. **exchangeElements** (if your sorting algorithm exchanges element positions): {parameters: *someArray[i]* (reference. input/output), *someArray[j]* (reference, input/output), where i and j are the indexes of elements to be exchanged}
   6. **displayMedian** {parameters: *someTitle* (reference, input), *someArray* (reference, input)} **NOTE: *ARRAYSIZE* will likely be used as a global within this procedure.**
   7. **displayList** {parameters: *someTitle* (reference, input), *someArray* (reference, input)} **NOTE: *ARRAYSIZE* will likely be used as a global within this procedure.**
   8. **countList** {parameters: *someArray1* (reference, input), *someArray2* (reference, output)} **NOTE: *LO*, *HI, and ARRAYSIZE*will be used as globals within this procedure.**
2. Procedures (except **main**) **must not** reference data segment variables by name. There is a **significant**penalty attached to violations of this rule.  **randArray**, **counts**, titles for the sorted/unsorted lists, etc... should be declared in the .data preceding **main**, but **must** be passed to procedures on the stack.
   1. Constants **LO**, **HI**, and **ARRAYSIZE** may be used as globals.
   2. Parameters **must** be passed on the system stack, by value or by reference, as noted above (see [Module 7, Exploration 1 - Passing Parameters on the Stack](https://canvas.oregonstate.edu/courses/1910943/pages/exploration-1-passing-parameters-on-the-stack) for method).
   3. Strings/arrays **must** be passed by reference.
3. Since you will not be using globals (except the constants) the program **must** use one (or more) of the addressing modes from the explorations (e.g. *Register Indirect* or *Indexed Operands* addressing for reading/writing array elements, and *Base+Offset* addressing for accessing parameters on the runtime stack.)  
   See [Module 7, Exploration 2 - Arrays in Assembly and Writing to Memory](https://canvas.oregonstate.edu/courses/1910943/pages/exploration-2-arrays-in-assembly-and-writing-to-memory) for details.
4. The programmer’s name and program title, and a description of program functionality (in student's own words) to the user must appear in the output.
5. **LO**, **HI**, and **ARRAYSIZE** **must** be declared as constants.  
   NOTE: We **will** be changing these constant values to ensure they are implemented correctly. Expect ranges as follows  
   **LO**: 5 to 20  
   **HI**: 30 to 60   
   **ARRAYSIZE**: 20 to 1000
6. There **must** be only one procedure to display arrays. This procedure **must** be called three times:
   1. to display the unsorted array
   2. to display the sorted array
   3. to display the counts array
7. All lists **must** be identified when they are displayed (use the *someTitle* parameter for the **displayList** procedure).
8. Procedures **may** use local variables when appropriate.
9. The program **must** be fully documented and laid out according to the [CS271 Style Guide](https://canvas.oregonstate.edu/courses/1910943/files/94857574/download?wrap=1)

[Actions](https://canvas.oregonstate.edu/courses/1910943/assignments/8972781?module_item_id=22461289)

. This includes a complete header block for identification, description, etc., a comment outline to explain each block of code, and proper procedure headers/documentation.

**Notes**

1. You are now allowed to use the **USES** directive, but only for preserving/restoring registers. You may not use its extended syntax to create local labels for stack-passed parameters.
2. The Irvine library provides procedures for generating random numbers. Call **Randomize** once at the beginning of the program (to set up so you don't get the same sequence every time), and call **RandomRange** to get a pseudo-random number. See the documentation in the [CS271 Irvine Procedure Reference](https://canvas.oregonstate.edu/courses/1910943/files/94857677/download?wrap=1)

[Actions](https://canvas.oregonstate.edu/courses/1910943/assignments/8972781?module_item_id=22461289)

 for more information on using these procedures.

1. The median is calculated after the array is sorted. It is the "middle" element of the sorted list. If the number of elements is even, the median is the average of the middle two elements.
2. Check the [Course Syllabus](https://canvas.oregonstate.edu/courses/1910943/files/94871569/download?wrap=1)

[Actions](https://canvas.oregonstate.edu/courses/1910943/assignments/8972781?module_item_id=22461289)

 for late submission guidelines.

1. Find the assembly language instruction syntax and help in the [CS271 Instructions Guide](https://canvas.oregonstate.edu/courses/1910943/files/95544921/download?wrap=1)

[Actions](https://canvas.oregonstate.edu/courses/1910943/assignments/8972781?module_item_id=22461289)

.

1. To create, assemble, run,  and modify your program, follow the instructions on the course [Syllabus Page](https://canvas.oregonstate.edu/courses/1910943/assignments/syllabus)’s "Tools" tab.

Resources

Additional resources for this assignment

* [Project Shell with Template.asm](https://canvas.oregonstate.edu/courses/1910943/files/94857201/download?wrap=1)[Download Project Shell with Template.asm](https://canvas.oregonstate.edu/courses/1910943/files/94857201/download?download_frd=1)
* [CS271 Style Guide](https://canvas.oregonstate.edu/courses/1910943/files/94857574/download?wrap=1)

[Actions](https://canvas.oregonstate.edu/courses/1910943/assignments/8972781?module_item_id=22461289)

* [CS271 Instructions Reference](https://canvas.oregonstate.edu/courses/1910943/files/95544921/download?wrap=1)

[Actions](https://canvas.oregonstate.edu/courses/1910943/assignments/8972781?module_item_id=22461289)

* [CS271 Irvine Procedure Reference](https://canvas.oregonstate.edu/courses/1910943/files/94857677/download?wrap=1)

[Actions](https://canvas.oregonstate.edu/courses/1910943/assignments/8972781?module_item_id=22461289)

What to turn in

Turn in a single .asm file (the actual Assembly Language Program file, not the Visual Studio solution file).  File must be named "Proj5\_ONID.asm" where ONID is your ONID username. Failure to name files according to this convention may result in reduced scores (or ungraded work).

Example Execution

**Generating, Sorting, and Counting Random integers!                      Programmed by Stephen**  
**This program generates 200 random integers between 15 and 50, inclusive.**  
**It then displays the original list, sorts the list, displays the median value of the list,**  
**displays the list sorted in ascending order, and finally displays the number of instances**  
**of each generated value, starting with the number of lowest.**  
  
**Your unsorted random numbers:**  
**40 36 34 36 49 38 35 32 28 22 36 24 49 18 30 31 48 20 20 17**  
**46 40 38 22 46 24 37 27 48 18 27 39 40 49 22 42 43 21 29 28**  
**23 27 21 38 30 33 17 19 40 37 29 19 15 15 49 23 45 37 29 45**  
**17 49 46 43 40 46 23 16 48 44 24 33 19 28 26 28 36 49 43 39**  
**27 15 25 17 49 38 31 19 41 42 42 19 15 32 45 43 15 15 43 22**  
**26 39 15 49 39 25 22 34 42 36 30 19 47 48 38 47 43 33 38 19**  
**40 19 30 41 25 27 23 35 23 16 48 20 23 42 29 17 50 30 19 44**  
**27 35 16 35 31 17 50 46 40 16 25 27 24 49 15 20 18 34 38 47**  
**50 37 33 23 37 46 16 49 46 31 47 47 41 42 23 35 42 34 43 44**  
**26 23 47 47 28 26 16 42 44 45 48 36 26 26 33 45 43 33 39 17**  
  
**The median value of the array: 34**  
  
**Your sorted random numbers:**  
**15 15 15 15 15 15 15 15 16 16 16 16 16 16 17 17 17 17 17 17**  
**17 18 18 18 19 19 19 19 19 19 19 19 19 20 20 20 20 21 21 22**  
**22 22 22 22 23 23 23 23 23 23 23 23 23 24 24 24 24 25 25 25**  
**25 26 26 26 26 26 26 27 27 27 27 27 27 27 28 28 28 28 28 29**  
**29 29 29 30 30 30 30 30 31 31 31 31 32 32 33 33 33 33 33 33**  
**34 34 34 34 35 35 35 35 35 36 36 36 36 36 36 37 37 37 37 37**  
**38 38 38 38 38 38 38 39 39 39 39 39 40 40 40 40 40 40 40 41**  
**41 41 42 42 42 42 42 42 42 42 43 43 43 43 43 43 43 43 44 44**  
**44 44 45 45 45 45 45 46 46 46 46 46 46 46 47 47 47 47 47 47**  
**47 48 48 48 48 48 48 49 49 49 49 49 49 49 49 49 49 50 50 50**  
  
**Your list of instances of each generated number, starting with the smallest value:**  
**8 6 7 3 9 4 2 5 9 4 4 6 7 5 4 5 4 2 6 4**  
**5 6 5 7 5 7 3 8 8 4 5 7 7 6 10 3**  
  
**Goodbye, and thanks for using my program!**