## 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 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. Generate **ARRAYSIZE** random integers in the range [**LO** = 10 ... **HI** = 29], storing them in consecutive elements of an array **randArray**. **ARRAYSIZE** should be set to 200.  
   HINT: Use Irvine’s **RandomRange** and **Randomize** procedures 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 **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 [10, 29] is seen in **randArray**.  
   For example, **counts[0]** should equal the number instances of the value `10` in array. **counts[14]** should equal the number of instances of the value `24` in **randArray**.
8. Display the array **counts**, 20 numbers per line with one space between each value.

#### **Program Requirements**

1. 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.
2. **LO**, **HI**, and **ARRAYSIZE** **must** be declared as constants.  
   NOTE: We will be changing these constant value to ensure they are implemented correctly. Expect ranges as follows  
   **LO**: 7 to 12  
   **HI**: 27 to 32   
   **ARRAYSIZE**: 180 to 220
3. Strings **must** be passed by reference.
4. The program **must** use Register Indirect addressing for ***all*** reading/writing ofarray elements
5. Base+Offset **may only**be used for addressing for accessing parameters on the runtime stack.
6. 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), LO (global), HI (global), ARRAYSIZE (global)}
   4. **sortList** {parameters: someArray (reference, input/output), ARRAYSIZE (global)}
   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), ARRAYSIZE (global)}
   7. **displayList** {parameters: someTitle (reference, input), someArray (reference, input), ARRAYSIZE global)}
   8. **countList** {parameters: someArray1 (reference, input), ARRAYSIZE (global), someArray2 (reference, output), LO (global), HI (global)}
7. Parameters **must** be passed on the system stack, by value or by reference (as noted above).
8. There **must** be just 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
9. All lists **must** be identified when they are displayed (use the someTitle parameter for the **displayList** procedure).
10. 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 section, but **must** be passed to procedures on the stack. Constants **LO**, **HI**, and **ARRAYSIZE** may be used as global.
11. Procedures **may** use local variables when appropriate.
12. The program **must** be fully documented and laid out according to the [CS271 Style Guide](https://canvas.oregonstate.edu/courses/1810861/files/85561736/download?wrap=1)[Download CS271 Style Guide](https://canvas.oregonstate.edu/courses/1810861/files/85561736/download?download_frd=1). This includes a complete header block for identification, description, etc., a comment outline to explain each section of code, and proper procedure headers/documentation.

#### **Notes**

1. You are now allowed to use the **USES** directive.
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/1810861/files/87258362/download?wrap=1)[Download CS271 Irvine Procedure Reference](https://canvas.oregonstate.edu/courses/1810861/files/87258362/download?download_frd=1)for more information on using these procedures.
3. 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.
4. Check the [Course Syllabus](https://canvas.oregonstate.edu/courses/1810861/files/85708476/download?wrap=1)[Download Course Syllabus](https://canvas.oregonstate.edu/courses/1810861/files/85708476/download?download_frd=1)for late submission guidelines.
5. Find the assembly language instruction syntax and help in the [CS271 Instructions Guide](https://canvas.oregonstate.edu/courses/1810861/files/85562428/download?wrap=1)[Download CS271 Instructions Guide](https://canvas.oregonstate.edu/courses/1810861/files/85562428/download?download_frd=1).
6. To create, assemble, run,  and modify your program, follow the instructions on the course [Syllabus Page](https://canvas.oregonstate.edu/courses/1810861/assignments/syllabus)’s "Tools" tab.

## Resources

Additional resources for this assignment

* [Project Shell with Template.asm](https://canvas.oregonstate.edu/courses/1810861/files/85561500/download?wrap=1)[Download Project Shell with Template.asm](https://canvas.oregonstate.edu/courses/1810861/files/85561500/download?download_frd=1)
* [CS271 Style Guide](https://canvas.oregonstate.edu/courses/1810861/files/85561736/download?wrap=1)[Download CS271 Style Guide](https://canvas.oregonstate.edu/courses/1810861/files/85561736/download?download_frd=1)
* [CS271 Instructions Reference](https://canvas.oregonstate.edu/courses/1810861/files/85562428/download?wrap=1)[Download CS271 Instructions Reference](https://canvas.oregonstate.edu/courses/1810861/files/85562428/download?download_frd=1)
* [CS271 Irvine Procedure Reference](https://canvas.oregonstate.edu/courses/1810861/files/87258362/download?wrap=1)[Download CS271 Irvine Procedure Reference](https://canvas.oregonstate.edu/courses/1810861/files/87258362/download?download_frd=1)

## 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). When you resubmit a file in Canvas, Canvas can attach a suffix to the file, e.g., the file name may become Proj5\_ONID-1.asm. Don't worry about this name change as no points will be deducted because of this.

## Example Execution

**Generating, Sorting, and Counting Random integers! Programmed by Stephen**  
  
**This program generates 200 random numbers in the range [10 ... 29], displays the**  
**original list, sorts the list, displays the median value of the list, displays the**   
**list sorted in ascending order, then displays the number of instances of each**   
**generated value, starting with the number of 10s.**  
  
**Your unsorted random numbers:**  
**11 24 18 19 26 10 19 15 11 20 16 21 21 19 24 18 12 10 19 20**  
**12 13 13 21 13 21 15 20 28 24 28 14 16 20 28 21 29 24 11 19**  
**24 13 22 19 20 10 12 13 13 15 13 26 27 24 17 17 27 21 21 18**  
**15 15 24 25 13 17 26 27 29 27 27 15 29 18 20 21 23 20 12 13**  
**22 15 19 20 22 18 13 16 20 22 15 23 18 22 12 29 18 18 12 24**  
**20 14 16 14 18 18 14 22 18 11 12 21 15 16 21 21 29 13 26 16**  
**17 15 13 21 10 20 12 24 28 10 18 18 19 28 20 26 23 15 16 16**  
**13 22 28 24 24 12 23 15 24 12 18 24 22 15 19 23 13 27 16 24**  
**29 15 24 28 18 23 28 21 16 23 22 10 24 12 20 16 25 17 25 12**  
**23 22 18 15 23 16 13 24 12 13 24 24 14 15 19 13 25 18 28 13**  
  
**The median value of the array: 19**  
  
**Your sorted random numbers:**  
**10 10 10 10 10 10 11 11 11 11 12 12 12 12 12 12 12 12 12 12**  
**12 12 12 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13**  
**13 14 14 14 14 14 15 15 15 15 15 15 15 15 15 15 15 15 15 15**  
**15 15 16 16 16 16 16 16 16 16 16 16 16 16 17 17 17 17 17 18**  
**18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 19 19 19 19**  
**19 19 19 19 19 19 20 20 20 20 20 20 20 20 20 20 20 20 20 21**  
**21 21 21 21 21 21 21 21 21 21 21 21 22 22 22 22 22 22 22 22**  
**22 22 23 23 23 23 23 23 23 23 23 24 24 24 24 24 24 24 24 24**  
**24 24 24 24 24 24 24 24 24 24 25 25 25 25 26 26 26 26 26 27**  
**27 27 27 27 27 28 28 28 28 28 28 28 28 28 29 29 29 29 29 29**  
  
**Your list of instances of each generated number, starting with the number of 10s:**  
**6 4 13 18 5 16 12 5 17 10 13 13 10 9 19 4 5 6 9 6**  
  
**Goodbye, and thanks for using this program!**

## Extra Credit (Original Project Definition must be Fulfilled)

To receive points for any extra credit options, you **must** add one print statement to your program output **per extra credit** which describes the extra credit you chose to work on. You ***will not receive extra credit points*** unless you do this. The statement must be formatted as follows...

--Program Intro--  
\*\*EC: DESCRIPTION  
  
--Program prompts, etc—

#### **Extra Credit Options**

1. Display the numbers ordered by column instead of by row. (1pt)
2. Generate the numbers directly into a file, then read the file into the array. This may modify your procedure parameters significantly. (3pts)