

Program #5

Due Feb 27 by 11:59pm **Points** 50 **Submitting** a file upload **File Types** asm



Due: Week 8, Sunday, 11:59 PM Pacific USA Time Zone

Objectives

1. Using indirect addressing and/or base-indexed addressing
2. Passing parameters on the stack
3. Generating “random” numbers
4. Working with arrays

Description

Write a MASM program to perform the tasks shown below. Be sure to test your program and ensure that it rejects incorrect input values.

1. Introduce the program.
2. Get a user *request* in the range [*min* = 15 .. *max* = 200].
- enerate *request* random integers in the range [*lo* = 100 .. *hi* = 999], storing them in consecutive elements of an *array*.
- isplay the list of integers before sorting, 10 numbers per line.
5. Sort the list in descending order (i.e., largest first).
6. Calculate and display the median value, rounded to the nearest integer.
7. Display the sorted list, 10 numbers per line.

Example Program Operation

```
Sorting Random Integers
Programmed by Author Name
This program generates random numbers in the range [100 .. 999],
displays the original list, sorts the list, and calculates the
median value. Finally, it displays the list sorted in descending order.
How many numbers should be generated? [15 .. 200]: 10
Invalid input
```

```

How many numbers should be generated? [15 .. 200]: 16
The unsorted random numbers:
680      329      279      846      123      101      427      913      255      736
431      545      984      391      626      803
The median is 488.
The sorted list:
984      913      846      803      736      680      626      545      431      427
391      329      279      255      123      101
Thanks for using my program!

```

Requirements

1. The title, programmer's name, and brief instructions must be displayed on the screen.
2. The program must validate the user's request.
3. The **main** procedure must consist (mostly) of procedure calls. It should be a readable "list" of what the program will do.
4. *min*, *max*, *lo*, and *hi* must be declared and used as **global constants**.
5. All procedure parameters must be passed on the system stack.
6. Each procedure will implement a section of the program logic, i.e., each procedure will specify how the logic of its section is implemented. The program must be modularized into at least the following procedures and sub-procedures:
 - main
 - introduction
 - get data {parameters: *request* (reference)}
 - fill array {parameters: *request* (value), *array* (reference)}
 - sort list {parameters: *array* (reference), *request* (value)}
 - exchange elements (for most sorting algorithms): {parameters: *array*[*i*] (reference), *array*[*j*] (reference), where *i* and *j* are the indices of elements to be exchanged}
 - display median {parameters: *array* (reference), *request* (value)}
 - display list {parameters: *array* (reference), *request* (value), *title* (reference)}
7. Parameters must be passed by value or by reference on the system stack as listed above.
8. There must be just one procedure to display the list. This procedure must be called twice: once to display the unsorted list, and once to display the sorted list.
9. Procedures (except main) should not reference .data segment variables by name. *request*, *array*, and *titles* for the sorted/unsorted lists should be declared in the .data segment, but procedures must use them as parameters. Procedures are allowed to use local variables when appropriate (section 8.2.9 in the textbook). **Global constants** are OK.

10. The program must use appropriate addressing modes for array elements.
11. The two lists must be identified when they are displayed (use the title parameter for the display procedure).
12. Each procedure must have a procedure header that follows the format discussed during lecture.
13. The code and the output must be well-formatted.
14. The usual requirements regarding documentation, readability, user-friendliness, etc., apply.

Notes

1. **DO NOT** put this off - it is a much more time-intensive project than any of Programs #1 - #4.
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 Lecture slides.)
3. The Selection Sort is probably the easiest sorting algorithm to implement. Here is a version of the descending order algorithm, where request is the number of array elements being sorted, and exchange is the code to exchange two elements of an array:

```
for (k=0; k<request-1; k++) {  
    i = k;  
    for (j=k+1; j<request; j++) {  
        if (array[j] > array[i])  
            i = j;  
    }  
    exchange(array[k], array[i]);  
}
```



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 (may be rounded).

5. If you choose to use the LOCAL directive while working on this program be sure to read section 8.2.9 in the Irvine textbook. LOCAL variables will affect the contents of the system stack!

Extra Credit Options

- (0.5 pt) Add a greeting message that contains EXACTLY ONE TA name at the beginning of the program. If that TA happens to be your grader, then you get the points :)
- (1 pt) Display the numbers ordered by column instead of by row.
- (3 pts) Implement the sorting functionality using a recursive Merge Sort algorithm. For a graphical explanation of the algorithm, I recommend this [GeeksForGeeks article](https://www.geeksforgeeks.org/merge-sort/) [_https://www.geeksforgeeks.org/merge-sort/](https://www.geeksforgeeks.org/merge-sort/). You may add additional procedures to your

program as needed. Remember that all parameters must be passed on the system stack.

In order to ensure you receive credit for any extra credit work, 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--
```

Please refer back to the documentation for Program 1 to see a sample of the extra credit format.

Program 5 Rubric



Criteria	Ratings			Pts
<p>Files Correctly Submitted</p> <p>Submitted file is correct assignment and is an individual .asm file.</p>	<p>1 pts</p> <p>Full Marks</p>	<p>0 pts</p> <p>No Marks</p>	1 pts	
<p>Program Assembles & Links</p> <p>Submitted program assembles and links without need for clarifying work for TA and/or messages to the student. This assumes the program is actually an attempt at the assignment. Non-attempts which compile/link earn no points.</p>	<p>1 pts</p> <p>Full Marks</p>	<p>0 pts</p> <p>No Marks</p>	1 pts	
<p>Documentation - Identification Block - Header</p> <p>Name, Date, Program number, etc as per syllabus are included in Identification Block</p>	<p>1 pts</p> <p>Full Marks</p>	<p>0 pts</p> <p>No Marks</p>	1 pts	
<p>Documentation - Identification Block - Program Description</p> <p>Description of functionality and purpose of program is included in identification block.</p> <div data-bbox="79 974 149 1039">▶</div>	<p>1 pts</p> <p>Full Marks</p>	<p>0 pts</p> <p>No Marks</p>	1 pts	
<p>Documentation - Procedure Headers</p> <p>Procedure headers describe functionality and implementation of program flow. Should also list pre- and post-conditions and registers changed.</p>	<p>2 pts</p> <p>Full Marks</p>	<p>1 pts</p> <p>Headers without Conditions</p> <p>Descriptive headers but lacking pre- and post-conditions and 'registers changed'</p>	<p>0 pts</p> <p>No Marks</p>	2 pts

Criteria	Ratings		Pts
<p>Documentation - In-line Comments</p> <p>In-line comments contribute to understanding of program flow (from section comments) but are not line-by-line descriptions of moving memory to registers.</p>	<p>1 pts Full Marks</p>	<p>0 pts No Marks</p>	1 pts
<p>Program Executes</p> <p>Program executes and makes some attempt at the assigned functionality.</p>	<p>2 pts Full Marks</p>	<p>0 pts No Marks</p>	2 pts
<p>Completeness - Displays Title & Programmer Name</p> <p>Program prints out the programmer's name and Program Title</p>	<p>1 pts Full Marks</p>	<p>0 pts No Marks</p>	1 pts
<p>Completeness - Displays Introduction</p> <p>Displays program introduction. Program introduction should describe functionality of program.</p>	<p>1 pts Full Marks</p>	<p>0 pts No Marks</p>	1 pts
<p>Completeness - Gets / Validates data from User</p> <p>Utilizes ReadInt or ReadDec to receive user input. Saves values in appropriately-named identifiers for validation. Validates that user-entered values are within the advertised limits.</p>	<p>2 pts Full Marks</p>	<p>0 pts No Marks</p>	2 pts
<p>Completeness - Displays Unsorted List</p>	<p>1 pts Full Marks</p>	<p>0 pts No Marks</p>	1 pts

Criteria	Ratings		Pts
Completeness - Displays Median	1 pts Full Marks	0 pts No Marks	1 pts
Completeness - Displays Sorted List	1 pts Full Marks	0 pts No Marks	1 pts
Completeness - Lists are displayed 10 numbers per line	1 pts Full Marks	0 pts No Marks	1 pts
Completeness - Lists are labeled "unsorted" and "sorted"	1 pts Full Marks	0 pts No Marks	1 pts
Correctness - Array declaration List array declared as 200 integers	1 pts Full Marks	0 pts No Marks	1 pts
<div>▶</div> Correctness - Random Number Generation Generates random numbers correctly in range	2 pts Full Marks	0 pts No Marks	2 pts
Correctness - Fill Array Stores random numbers consecutively in array	2 pts Full Marks	0 pts No Marks	2 pts

Criteria	Ratings			Pts
Correctness - List is sorted in descending order	5 pts Full Marks	3 pts Ascending order List is sorted in ascending (instead of descending) order	0 pts No Marks	5 pts
Correctness - Median Calculation Correctly calculates and rounds median	2 pts Full Marks	0 pts No Marks		2 pts
Requirements - Modular Design Uses procedures: main, introduction, getData, fillArray, sortList, displayMedian, displayList	8 pts Full Marks	6 pts All except median All procedures exist and used except displayMedian. Median calculation done in main or grouped in other procedure.	0 pts No Marks	8 pts
Requirements - Parameter Passing Parameters are passed to procedures on the system stack (arrays, strings, and return values by OFFSET; others by value)	5 pts Full Marks	3 pts All but strings Value-parameters and return parameters passed on stack but array/string offsets are not.	0 pts No Marks	5 pts
Requirements - displayList correctly used Two different CALLs to the same displayList procedure (one for the unsorted list, one for the sorted). If there are two different list display procedures - zero points	2 pts Full Marks	0 pts No Marks		2 pts

Criteria	Ratings		Pts
Requirements - Limits as CONSTANTS Random range bounds and user input bounds are declared and used as constants.	2 pts Full Marks	0 pts No Marks	2 pts
Requirements - No non-CONSTANT global references outside of main Deduct 5 points if there are any non-CONSTANT global references outside of MAIN	0 pts Full Marks	0 pts No Marks	0 pts
Requirements - Correct Array Addressing Modes Deduct 5 points if indirect addressing (register as pointer) or base-index is not used for array processing.	0 pts Full Marks	0 pts No Marks	0 pts
Coding Style - Uses appropriately named identifiers Identifiers named so that a person reading the code can intuit the purpose of a variable, constant, or label just by reading its name. <div data-bbox="79 976 149 1040">▶</div>	1 pts Full Marks	0 pts No Marks	1 pts
Coding Style - Readability Program uses readable white-space, indentation, and spacing as per the Indentation Style Guide. Logical sections are separated by white space.	1 pts Full Marks	0 pts No Marks	1 pts
Output Style - Readability Program output is easy to read	1 pts Full Marks	0 pts No Marks	1 pts

Criteria	Ratings		Pts
Extra Credit Successfully guess the grader (0.5 pts) Displays the numbers ordered by column instead of by row (1 pt) Implements recursive merge sort algorithm (3 pts)	0 pts Full Marks	0 pts No Marks	0 pts
Late Penalty Remove points here for late assignments. (Enter negative point value)	0 pts Full Marks	0 pts No Marks	0 pts
			Total Points: 50

