Program 6 (Option A)

Due Dec 8, 2019 by 11:59pm **Points** 50 **Submitting** a file upload **File Types** asm
Available after Nov 17, 2019 at 12am

Due: Week 10, Sunday, 11:59 PM Pacific USA Time Zone

Note: You only need to submit one version of Program 6 (**EITHER** Option A **OR** Option B). Since program 6 is the final homework assignment in CS271, it's recommended that you strive to write clear, clean code that is easy to read. This can be used to form part of a portfolio that you show potential employers.

Objectives

1. Designing, implementing, and calling low-level I/O procedures
2. Implementing and using a macro

Problem Definition

- Implement and test your own *ReadVal* and *WriteVal* procedures for unsigned integers. Your *ReadVal* implementation will accept a numeric string input from the keyboard and will compute the corresponding integer value. For example, if the user entered a string "1234" then the numeric value 1234 would be computed (and stored in the requested OFFSET). *WriteVal* will perform the opposite transformation. For example, *WriteVal* can accept a 32 bit unsigned int and display the corresponding ASCII representation on the console (e.g. if *WriteVal* receives the value 49858 then the text "49858" will be displayed on the screen).
- Implement macros *getString* and *displayString*. The macros may use Irvine's *ReadString* to get input from the user, and *WriteString* to display output.
- Additional details are as follows:
  - *getString* should display a prompt, then get the user's keyboard input into a memory location.
  - *displayString* should print the string which is stored in a specified memory location.
  - *ReadVal* should invoke the *getString* macro to get the user's string of digits. It should then convert the digit string to numeric, while validating the user's input.
  - *WriteVal* should convert a numeric value to a string of digits and invoke the *displayString* macro to produce the output.
- Once you have implemented the two procedures you will then demonstrate their usage by creating a small test program. The program will get 10 valid integers from the user and store the numeric values into an array. The program will then display the list of integers, their sum, and the average value of the list.

Example Program Operation

```
Demonstrating low-level I/O procedures
Written by: Author Name

Please provide 10 decimal integers.
Each number needs to be small enough to fit inside a 32 bit register.
After you have finished inputting the raw numbers I will display a list
of the integers, their sum, and their average value.
Please enter an integer number: 156
Please enter an integer number: 51d6fd
     ERROR: You did not enter an integer number or your number was too big.
Please try again: 34
Please enter an integer number: 186
Please enter an integer number: 15616148561615630
     ERROR: You did not enter an integer number or your number was too big.
Please try again: 145.0
     ERROR: You did not enter an integer number or your number was too big.
Please try again: 345
Please enter an integer number: 5
Please enter an integer number: 23
Please enter an integer number: 51
Please enter an integer number: 0
Please enter an integer number: 56
Please enter an integer number: 11

You entered the following numbers:
156, 34, 186, 345, 5, 23, 51, 0, 56, 11
The sum of these numbers is: 867
The average is: 86
Thanks for playing!
```

Requirements

1. The title, programmer's name, and brief instructions must be displayed on the screen.
2. Your ReadVal procedure must be designed to utilize the LODSB instruction. WriteVal must be written to utilize the STOSB instruction.
3. User’s numeric input must be validated the hard way: read the user's input as a string, and convert the string to numeric form. If the user enters non-integers or the number is too large for 32-bit registers, an error message needs to be displayed and the number should be discarded.
4. Addresses of prompts, identifying strings, and other memory locations must be passed by reference to the macros. This is another way of saying that your macros will be designed to accept the OFFSET of prompts, identifying strings, etc.
5. All procedure parameters need to be passed on the system stack (either by value or by reference).
6. Procedures (except main) should not refer to .data segment variables by name. Procedures are allowed to use local variables when appropriate (section 8.2.9 in the textbook). It is acceptable for procedures to refer to **global constants** by name.

7. Used registers must be saved and restored by the called procedures and macros.
8. The stack must be “cleaned up” by the called procedure.
9. Each procedure must have a procedure header that follows the format discussed during lecture.
10. The code and the output must be well-formatted.
11. The usual requirements regarding documentation, readability, user-friendliness, etc., apply.

Additional Notes

1. For this assignment you are allowed to assume that the total sum of the numbers will fit inside a 32 bit register.
2. When you are converting between a string and an integer (or vice versa), your code will need to perform some math. As a hint, consider the string "52832". In order to convert this string to a decimal number, you simply need to sum the value of each column (I suggest you start from the rightmost column). In this case, the conversion will occur as 2*1 + 3*10 + 8*100 + 2 * 1000 + 5 * 10000. This gives the sum 52832 which can then be stored in the requested OFFSET.
3. When displaying the average, you may round down to the nearest integer. For example if the sum of the 10 numbers is 3577 you may display the average as 357.
4. 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

- (1 pt) Number each line of user input and display a running subtotal of the user’s numbers.
- (2 pts) Implement your code so that it correctly handles signed integers (i.e. allow the user to enter negative numbers as well as positive numbers).
- (3 pts) Implement the getString and displayString macros using the Win32 API functions that are discussed in chapter 11.1 of the textbook (rather than utilizing the Irvine library). This implies that your macros **cannot** use the Irvine ReadString or WriteString procedures.

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 6a Rubric

Criteria	Ratings		Pts
Files Correctly Submitted Submitted file is correct assignment and is an individual .asm file.	**1.0 pts** **Full Marks**	**0.0 pts** **No Marks**	1.0 pts
Program Assembles & Links 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.	**1.0 pts** **Full Marks**	**0.0 pts** **No Marks**	1.0 pts
Documentation - Identification Block - Header Name, Date, Program number, etc as per syllabus are included in Identification Block	**1.0 pts** **Full Marks**	**0.0 pts** **No Marks**	1.0 pts
Documentation - Identification Block - Program Description Description of functionality and purpose of program is included in identification block.	**1.0 pts** **Full Marks**	**0.0 pts** **No Marks**	1.0 pts
Documentation - Procedure Headers Procedure headers describe functionality and implementation of program flow. Should also list pre- and post-conditions and registers changed.	**2.0 pts** **Full Marks**	**0.0 pts** **No Marks**	2.0 pts
Documentation - In-line Comments In-line comments contribute to understanding of program flow (from section comments) but are not line-by-line descriptions of moving memory to registers.	**2.0 pts** **Full Marks**	**0.0 pts** **No Marks**	2.0 pts
Program Executes Program executes and makes some attempt at the assigned functionality.	**2.0 pts** **Full Marks**	**0.0 pts** **No Marks**	2.0 pts
Completeness - Gets / Validates User Numbers No non-numeric characters should be accepted	**3.0 pts** **Full Marks**	**0.0 pts** **No Marks**	3.0 pts
Completeness - Displays Sum	**1.0 pts** **Full Marks**	**0.0 pts** **No Marks**	1.0 pts
Completeness - Displays Average	**1.0 pts** **Full Marks**	**0.0 pts** **No Marks**	1.0 pts

Criteria	Ratings			Pts
Completeness - Test program gets 10 integers from user	**1.0 pts** **Full Marks**	**0.0 pts** **No Marks**		1.0 pts
Correctness - String to Number Conversion	**3.0 pts** **Full Marks**	**0.0 pts** **No Marks**		3.0 pts
Correctness - Num to String Conversion All instances of conversion from Num to String generate correct output.	**3.0 pts** **Full Marks**	**0.0 pts** **No Marks**		3.0 pts
Correctness - Calculations	**2.0 pts** **Full Marks**	**0.0 pts** **No Marks**		2.0 pts
Correctness - getString macro works	**1.0 pts** **Full Marks**	**0.0 pts** **No Marks**		1.0 pts
Correctness - displaysString macro works	**1.0 pts** **Full Marks**	**0.0 pts** **No Marks**		1.0 pts
Requirements - Main Functionality / Hierarchy Main procedure calls sub-procedures	**1.0 pts** **Full Marks**	**0.0 pts** **No Marks**		1.0 pts
Requirements - Procedures implemented with logical hierarchy	**4.0 pts** **Full Marks**	**0.0 pts** **No Marks**		4.0 pts
Requirements - User input is read as string & converted to numeric (with error checking) User input needs to be checked for non-numeric characters. If the user enters a number that exceeds the capacity of a 32 bit value the program must display an error and reject the number.	**5.0 pts** **Full Marks**	**3.0 pts** **Lacking error checking**	**0.0 pts** **No Marks**	5.0 pts

Criteria	Ratings			Pts
Requirements - Output is converted from numeric to string Both for NUM and AVERAGE	**4.0 pts** **Full Marks**	**2.0 pts** **Either/Or** Output is converted for sum but not average, or average but not sum	**0.0 pts** **No Marks**	4.0 pts
Requirements - Used registers saved/restored by procedures	**2.0 pts** **Full Marks**		**0.0 pts** **No Marks**	2.0 pts
Requirements - Parameters passed on stack, cleaned up by called procedure	**4.0 pts** **Full Marks**		**0.0 pts** **No Marks**	4.0 pts
Requirements - Implements & uses Macros to display strings	**1.0 pts** **Full Marks**		**0.0 pts** **No Marks**	1.0 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.	**1.0 pts** **Full Marks**		**0.0 pts** **No Marks**	1.0 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.	**1.0 pts** **Full Marks**		**0.0 pts** **No Marks**	1.0 pts
Output Style - Readability Program output is easy to read	**1.0 pts** **Full Marks**		**0.0 pts** **No Marks**	1.0 pts
Extra Credit (1 pt) Numbers each line of user input / displays running subtotal of user's numbers | (2 pts) Handles signed integers | (3 pts) getString and displayString use the Win32 API to read/write all console output (rather than the Irvine library)	**0.0 pts** **Full Marks**		**0.0 pts** **No Marks**	0.0 pts
Late Penalty Remove points here for late assignments. (Enter negative point value)	**0.0 pts** **Full Marks**		**0.0 pts** **No Marks**	0.0 pts

Criteria	Ratings	Pts
		Total Points: 50.0