**CS315 Program 3  
Spring 2018  
Due: Friday Mar. 30th, 11:59 pm**

The purpose of this program is to help you understand dynamic array, system stack and multi-level subprogram calls.

This program will first prompt user to input a valid array size (between 1 inclusive to 21 exclusive) and then dynamically allocate an array in memory based on the size. It will then ask user to input integers one by one to fill out the allocated array until it’s full. After input, the program will print out the original array and then do three operations on the array:

1. calculate the summation of all odd numbers in the array and print out the result;
2. dynamically allocate a new array with the same size, reversely copy all elements from the original array in to the new array, and then print the new array to the console;
3. Prompt user for a stride n and print out every element with its index divisible by n. this subprogram should be applied to the reversed array. (Hit: 0 is also divisible by any n)

To achieve this goal, you will need 1 main program associate with 7 subprograms:

1. create\_array

This subprogram will have no argument IN and two arguments OUT:

$sp+0 - array base address (OUT)  
 $sp+4 - array size (OUT)

create\_array should use 0 as argument and call allocate\_array to dynamically allocate a new array. And then call read\_array to fill out this array. It will return base address and array size to the calling location at the end.

1. allocate\_array

This subprogram will have one argument IN and two arguments OUT:

$sp+0 - array size (IN)

$sp+4 - array base address (OUT)

$sp+4 - array size (OUT)

allocate\_array will take the argument passed in and validate it. If the input is valid use this argument as the array size. If not, it will prompt user for a new valid array size. When it has a valid size, this subprogram will dynamically allocate enough memory to hold the array based on the input size, and then return base address and the array size (number of elements) back to the calling location.

1. read\_array

This subprogram will have two arguments IN and no argument OUT:

$sp+0 - array base address (IN)

$sp+4 - array size (IN)

read\_array will read integer values from user and store them into the array until the array is full. A prompt must be printed before each entry is read (e.g. “Enter an integer: “”).

1. print\_array

This subprogram will have two arguments IN and no arguments OUT:

$sp+0 - array base address (IN)

$sp+4 - array size (IN)

print\_array will print all values from the array with each value separated by a space.

1. print\_every\_nth

This subprogram will have three arguments IN and no argument OUT:

$sp+0 - array base address (IN)

$sp+4 - array size (IN)

$sp+8 - stride N (IN)   
  
print\_every\_nth will print every nth value from the array (index divisible by N), starting from the first value (index 0). For example, consider the array [1 2 3 4 5 6 7 8 9 0].

* For N = 3, the output would be [1 4 7 0].
* For N = 4, the output would be [1 5 9].
* If N ≤ 0, the subprogram will print an error message instead.

1. sum\_odd\_values

This subprogram will have two arguments IN and one argument OUT:

$sp+0 - array base address (IN)

$sp+4 - array size (IN)

$sp+8 - sum of odd values (OUT)

sum\_odd\_values will calculate the sum of all odd values in the array and return that sum to calling location. For example: given the array **[8 4 3 6 5 5 2 8 7 2]**, the sum would be **(3+5+5+7=20)**. sum\_odd\_values will NOT print anything (main should print the sum).

1. reverse\_array

This subprogram will have two arguments IN and one argument OUT:

$sp+0 - array base address (IN)  
$sp+4 - array size (IN)

$sp+8 - array base address of reversed array (OUT)

reverse\_array will call allocate\_array to dynamically allocate a new array with the same size as the original one, and then reversely copy all values from the original array into the new array. This subprogram will return the base address of reversed array back to the calling location. Remember the size of reverse array is the same as size of original array so there is no need to return it again.

pseudocode:

revered array [0] = original array [size -1]

revered array [1] = original array [size -2]

…

revered array [size - 1] = original array [0]

* Original Array: [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
* Reversed array: [0, 9, 8, 7, 6, 5, 4, 3, 2, 1]

The structure of the program should as follow:

* Main:
  + Create\_array
    - Allocate\_array
    - Read-array
  + Print\_array
  + Sum\_odd\_values
  + Reverse\_array
    - Allocate\_array
  + Print\_every\_nth

Remember that main should also print the returned sum of odd values after sum\_odd\_values subprogram. You should store array base address, reversed array base address, array length into static variables.

Notes:

* Arguments MUST be passed through the system stack as specified (do not reorder them).
* Important data should be stored in static variables with a proper name
* Subprograms SHOULD NOT access labels from other subprogram’s data section.
* Download and use [subprogram\_template.s] from D2L
* Write sufficiently enough comments, subprogram description and do not forget to fill out register usage section and argument IN & OUT section
* Submit your code to D2L drop box [Program #3] with the following naming scheme: Program3\_YOURLASTNAME.s