Arrays, strings and functions in Assembly

Luís Nogueira

lmn@isep.ipp.pt

2024/2025

Notes:

- Each exercise should be solved in a modular fashion. It should be organised in two or more modules and compiled using the rules described in a Makefile.
- The code should be commented and indented
- The C parameter passing conventions are these:

Operand size (bits)	Argument number					
	1	2	3	4	5	6
64	`%rdi`	`%rsi`	`%rdx`	`%rcx`	`%r8`	`%r9`
32	`%edi`	`%esi`	`%edx`	`%ecx`	`%r8d`	`%r9d`
16	`%di`	`%si`	`%dx`	`%cx`	`%r8w`	`%r9w`
8	`%dil`	`%sil`	`%dl`	`%cl`	`%r8b`	`%r9b`

- Implement the following functions in Assembly and test them using a program in C

1. Implement a function `int five_count(char* ptr)` that returns the number of five chars (`'5'`) in a string pointed by `ptr`.
2. Implement a function `void str_copy_roman(char* ptr1, char* ptr2)` that copies the string pointed by `ptr1` to the string pointed by `ptr2`, replacing each occurrence of the character `'u'` by `'v'`, considering lower case characters only.

3. Implement a function `void str_copy_roman2(char* ptr1, char* ptr2)` that copies the string pointed by `ptr1` to the string pointed by `ptr2`, replacing each occurrence of the character `'u'` by the character `'v'`, considering lower and upper case characters.

4. Implement a function `void vec_add_three(short* ptr, int num)` that adds three (3) to all the elements of an array of type `short`, with `int num` elements and pointed by `ptr`.

5. Implement a function `int vec_sum(int* ptr, short num)` that returns the sum of all the elements of an array of type `int`, pointed by `ptr`, with `short num` elements.

   Then, implement another function `int vec_avg(int* ptr, short num)` in Assembly that uses the value returned by `int vec_sum(int* ptr, short num)` to compute the average.

6. Implement a function `int encrypt(char* ptr)` that adds one (1) to all the characters, that are not `'a'` or space, of the string pointed by `ptr`. The function should return the number of changed characters.

7. Implement a function `int decrypt(char* ptr)` that decrypts the string pointed by `ptr` and encrypted by `int encrypt(void)`.The function should return the number of changed characters.

8. Implement the function `int test_even(int x)` that tests if the number `x` is even. The function should return one (1) if it is even or zero (0) if it is odd.

   Use the previous function to implement another function `int vec_sum_even(int* ptr, int num)` in Assembly that returns the sum of all the even elements of an array of type `int` pointed by `ptr`, with `num` elements.

9. Implement a function `int* vec_search(int* ptr, int num, int x)` that searches `int x` in an array of type `int`, pointed by `ptr`, with `int num` elements, and returns the memory address of it's first occurrence or zero if not found.

10. Implement a function `void str_cat(char* ptr1, char* ptr2, char* ptr3)` that concatenates, in a string pointed by `ptr3`, the strings pointed by `ptr1` and `ptr2`, which have a maximum length of 40 bytes each.

11. Implement a function `int vec_greater12(int* ptr, int num)` that returns the number of elements of an array of type `int`, pointed by `ptrvec`, with `int num` elements, that are greater than 12.

12. Implement a function `unsigned char vec_zero(int* ptr, int num)` that zeroes the elements of an array of type `int`, pointed by `ptr`, with `int num` elements, that are greater or equal to 50. The function should return the number of changed elements.

13. Implement a function `void keep_positives(int* ptr, int num)` that changes an array of type `int`, with `int num` elements and pointed by `ptr`, by replacing all the negative numbers by their respective indexes on the array, keeping the positive elements unchanged.

14. Implement the function `int exists(int* ptr, int num, int x)`, that tests if `int x` exists more than once in the array of type `int` pointed by `ptr` with `int num` elements. The function should return 1 if `int x` has duplicates or 0 if not.

    Use the previous function to implement a function `int vec_diff(int* ptr, int num)` that computes the number of elements in the array of type `int` pointed by `ptr` that do not have duplicates.

15. Implement a function `int sum_third_byte(long* ptr, int num)` that returns the sum of the third byte of all the elements of the array of type `long` pointed by `ptr`, with `int num` elements.

16. Implement a function `void swap(char* ptr1, char* ptr2, int num)` that swaps the content of the arrays of type `char` pointed by `ptr1` and `ptr2`, both with `int num` elements (i.e contents of the first array will be copied to second array and vice versa). The new content of each array must be printed in the main function.

17. Implement a function `void array_sort(int* ptr, int num)` that, given the address of an array of type `int` pointed by `ptr`, with `int num` elements, sorts the array in descending order.

18. Implement a function `int sort_without_reps(short* ptrsrc, short* ptrdest, int num)` that, given the address of an array of type `short` pointed by `ptrsrc`, with `int num` elements, and the address of an empty array of the same size pointed by `ptrdest`, fills the `ptrdest` array with the elements of the `ptrsrc` array in ascending order, eliminating all repeated values. The function must return the number of items placed in the second array.

19. Implement a function `void frequencies(char* ptrgrades, int num, int* ptrfreq)` that, given the address of an array of type `char` pointed by `ptrgrades` with the students' exam grades at ARQCP (a value between 0 and 20), with `int num` elements, and the address of a second array of integers pointed by `ptrfreq`, it should fill `ptrfreq` with the absolute frequency of the grades stored in `ptrgrades`.

20. Implement a function `int count_max(int* ptr, int num)` that, given the address of an array of type `int` pointed by `ptr`, with `int num` elements, counts how many elements in `ptr` satisfy the condition $v_i < v_{i+1} > v_{i+2}$.