

# Arrays, strings and parameter-less functions in Assembly

Luís Nogueira

lmn@isep.ipp.pt

2022/2023

## 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.
  - Unless clearly stated otherwise, the needed data structures for each exercise **must be declared as global variables in the main C module**
  - The code should be commented and indented
  - Implement the following functions in Assembly and test them using a program in C
1. Implement a function `int seven_count(void)` that returns the number of seven chars ('7') in a string pointed by `ptr1`.
  2. Implement a function `void str_copy_porto(void)` that copies the string pointed by `ptr1` to the string pointed by `ptr2`, exchanging each occurrence of the character 'o' by 'u', considering lower case characters only.
  3. Implement a function `void str_copy_porto2(void)` that copies the string pointed by `ptr1` to the string pointed by `ptr2`, exchanging each occurrence of the character 'o' by the character 'u', considering lower and upper case characters.
  4. Implement a function `void vec_add_two(void)` that adds two (2) to all the elements of an array of type `int`, with `int num` elements and pointed by `ptrvec`.

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

Then, implement another function `long vec_avg(void)` in Assembly that uses the value returned by `long vec_sum(void)` to compute the average.

6. Implement a function `int encrypt(void)` that adds three (3) to all the characters, that are not 'a' or space, of the string pointed by `ptr1`. The function should return the number of changed characters.
7. Implement a function `int decrypt(void)` that decrypts the string pointed by `ptr1` and encrypted by `int encrypt(void)`. The function should return the number of changed characters.
8. Implement the function `long test_even(void)` that tests if the number in the variable `long even` 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 `long vec_sum_even(void)` in Assembly that returns the sum of all the even elements of an array of type `long` pointed by `ptrvec`, with `int num` elements.

9. Implement a function `short* vec_search(void)` that searches `short x` in an array of type `short`, pointed by `ptrvec`, 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(void)` 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_greater10(void)` that returns the number of elements of an array of type `int`, pointed by `ptrvec`, with `int num` elements, that are greater than 10.
12. Implement a function `unsigned char vec_zero(void)` that zeroes the elements of an array of type `int`, pointed by `ptrvec`, with `int num` elements, that are greater or equal to 100. The function should return the number of changed elements.
13. Implement a function `void keep_positives(void)` that changes an array of type `short`, with `unsigned short num` elements and pointed by `ptrvec`, by replacing all the negative numbers by their respective indexes on the array, keeping the positive elements unchanged.

14. Implement the function `int exists(void)`, that tests if `int x` exists more than once in the array of type `int` pointed by `ptrvec` 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(void)` that computes the number of elements in the array of type `int` pointed by `ptrvec` that do not have duplicates.

15. Implement a function `int sum_first_byte()` that returns the sum of the first byte of all the elements of the array of type `long` pointed by `ptrvec`, with `int num` elements.
16. Implement a function `void swap()` 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(void)` that, given the address of an array of type `short` pointed by `ptrvec`, with `int num` elements, sorts the array in descending order.
18. Implement a function `int sort_without_reps(void)` 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(void)` 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 pointed by `ptrfreq`, it should fill `ptrfreq` with the absolute frequency of the grades stored in `ptrgrades`.
20. Implement a function `int count_max(void)` that, given the address of an array of type `int` pointed by `ptrvec`, with `int num` elements, counts how many elements in `ptrvec` satisfy the condition  $v_i < v_{i+1} > v_{i+2}$ .