Pointers in C

Luís Nogueira lmn@isep.ipp.pt

2021/2022

Notes:

- Each exercise should be solved in a modular fashion. It should be organized in two or more modules and compiled using the rules described in a Makefile
- Function signatures cannot be altered or else the program will fail the unit tests
- Unless clearly stated otherwise, the needed data structures for each exercise must be declared locally in the main function and their addresses passed as parameters to the developed auxiliary functions
- The code should be commented and indented

✓ 1 – Implement a program which declares the following variables:

```
int x = 5;
int* xPtr = &x;
float y = *xPtr + 3;
int vec[] = {10, 11, 12, 13};
```

The program should print:

- The value of *x* and *y*
- The address (in Hexadecimal) of *x* and *xptr*
- The value pointed by *xptr*
- The address of *vec*
- The values of *vec [0]*, *vec [1]*, *vec [2]* and *vec [3]*
- The addresses of *vec [0]*, *vec [1]*, *vec [2]* and *vec [3]*

a) Analyze the information that appears on the screen.
b) Explain the relationship between the addresses of *vec*, *vec[0]*, *vec[1]* and *vec[2]*.
c) If the program was run in different computers, would you expect to see the same or different addresses of the variables?

✓ 2 – Implement a function ***void copy_vec(int *vec1, int *vec2, int n)*** that, through the use of pointers, copies *n* integers from *vec1* into *vec2*.

✓ 3 – Implement a function ***int sum_even(int *p, int num)*** that, given an array of integer values, sums all its even elements. The function should receive two parameters: the address of the array and an integer that indicates the number of elements in that array.

✓ 4 – Implement a function ***void upper1(char *str)*** that, given the address of a string, replaces all lowercase letters by their uppercase counterpart. Your function should not use other functions available in the C standard library.

5 – Implement a new function ***void upper2(char *str)*** to solve the previous exercise using pointer arithmetic.

✓ 6 – Implement a function ***void power_ref(int* x, int y)*** that calculates the power x^y. The result should be placed at the address indicated in the first parameter, changing its initial value.

7 – Implement a function ***void array_sort1(int *vec, int n)*** that, given the address of an integer array, orders the same array in ascending order. The content of the array after sorting should be printed in the main function.

8 – Implement a new function ***void array_sort2(int *vec, int n)*** to solve the previous exercise using pointer arithmetic.

✓ 9 – Implement a function ***int sort_without_reps(int *src, int n, int *dest)*** that receives the address of an array *src* with *n* integer elements and the address of an empty second array of the same size. The function should fill the second array with the elements of the first in ascending order, eliminating all repeated values. The function must return the number of items placed in the second array. The content of the second array should be printed in the main function.

✓ 10 – Using pointer arithmetic, implement a function ***int odd_sum(int *p)*** that receives the address of an array of integer values as its single parameter. The first element of the array indicates how many elements are stored on it. The function should return the sum of the odd elements of the array (excluding its first element).

✓ 11 – A palindrome is a word or sentence that reads the same forward as it does backward. Neither spaces nor punctuation are usually taken into consideration when constructing sentences that are palindromes. Implement a function ***int palindrome(char *str)*** function that verifies if a string, whose address is received as a parameter, is a palindrome or not. The function should return 1 or 0, depending on whether or not the string is a palindrome.

Examples:

- Never odd or even
- A man a plan a canal Panama.
- Gateman sees name, garageman sees name tag

✓ 12 – Using pointer arithmetic, implement a function ***void capitalize (char *str)*** to capitalize the words of a sentence. The function should receive the address of a string, changing it accordingly. The new content should be printed in the main function.

Examples:

- "The number must be saved" → "The Number Must Be Saved"
- "The maximum value of this CYCLE" → "The Maximum Value Of This CYCLE"

13 – Using pointer arithmetic, implement a function ***int where_is (char *str, char c, int *p)*** that receives three parameters: the address of a string (*str*), a character (*c*) and the address of an empty integer array (*vec*). The function should iterate through *str* looking for the character *c*. Whenever it finds the desired character it should add to *vec* the index where the character was found in the string. The final content of vec should be printed in the main function. Ensure that the size of the integer array *vec* is enough to accommodate all occurrences of *c* in *str*. The function returns the number of times it has found the character *c*.

14 – Implement a function ***void frequencies(float *grades, int n, int *freq)*** that receives the address of and array *grades* with the students' exam grades at ARQCP (a float value between 0.0 and 20.0), the number of elements in that array (*n*), and the address of a second array (*freq*) to be filled with the absolute frequency of the integer part of the grades stored in the array *grades*. Use pointer arithmetic to solve this exercise.

Example:

- the array *grades* with content {8.23, 12.25, 16.45, 12.45, 10.05, 6.45, 14.45, 0.0, 12.67, 16.23, 18.75} should produce a *freq* array with {1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0, 0}

15 – Implement a program to determine how many sets of three consecutive elements exist in an array of integer values that satisfy the condition $v_i < v_{i+1} < v_{i+2}$. Create three auxiliary functions:

- a function ***void populate(int *vec , int num, int limit)*** to populate an array (*vec*) of 100 integers (*num*) with random values between 0 and 20 (*limit*);
- a function ***int check (int x, int y, int z)*** that verifies if a set of three integers satisfies (1) or not (0) the given condition;
- a function ***void inc_nsets()*** to increment the number of sets. The number of sets should be stored in a global variable defined in the same module of this function.

Each of these functions should be placed in a separate module. Use pointer arithmetic to solve this exercise.

16 – Using pointer arithmetic, implement a function ***char* where_exists(char* str1, char* str2)*** that receives the address of two strings (*str1* and *str2*) and determines if the first string exists within the second. The function should return a pointer to the beginning of the first (or only) occurrence of *str1* within *str2* or NULL otherwise.

17 – Implement a function ***void swap(int* vec1, int* vec2, int size)*** that receives the address of two arrays of the same size and swaps the contents of both arrays (i.e contents of *vec1* will be

copied to *vec2* and vice versa). The new content of each array must be printed in the main function.

18 – Implement a function ***void compress_shorts(short* shorts, int n_shorts, int* integers)*** to "compress" the values of an array of shorts into an array of integers. The function must, in a sequential manner, compress two consecutive shorts into a single integer that must be stored in the array *integers*. Assume that the number of elements in the array shorts is even. The content of the integer array must be printed in the main function.

19 – Implement a function ***char* find_word(char* word, char* initial_addr)*** to search for a word in a string and returns the address in which it was found, or NULL otherwise. The function must find words whether in upper or lower case (consider that the string has only letters and spaces). The function receives in *initial_addr* the address within the string from which the search should be initiated.

20 – Implement a function ***void find_all_words(char* str, char* word, char** addrs)*** that, using the function developed in the previous exercise, find all the occurrences of a word in a string. The function should populate the array *addrs* with the all the addresses of the word found in *str*. Ensure that the array *addrs* has enough space to accommodate all the possible addresses.