Introduction to C Programming

Luís Nogueira | Paulo Ferreira | Raquel Faria

André Andrade | Carlos Gonçalves | Marta Fernandes

Cláudio Maia | Ricardo Severino

{lmn,pdf,arf,lao,cag,ald,crr,rar}@isep.ipp.pt

2019/2020

1. Implement a C program that displays the size of the following data types:
   - *char*, *int*, *unsigned int*, *long*, *short*, *long long*, *float*, *double*

2. Implement a function *int sum(int a, int b)* that returns the sum of two integers. Your program should invoke that function inside a loop until the returned result is less than 10.

3. Implement a function *void fill_array(int *vec)* that fills an entire array with 30 integer values entered using the keyboard. Then, the program should calculate and print the average of the values previously stored in the array.

4. Implement a function *int count(int *vec, int n, int value)* that counts the number of times that a *value* appears in an array *vec* with *n* elements.

5. Implement a function *int string_to_int(char *str)* that transforms a string into an equivalent integer value. For example, the string "12345" must be transformed into the integer 12345. In this exercise do not use the *atoi()* function.

6. Using the previous function, implement a program that calculates the average of two numbers entered using the keyboard in the form of string and display the result on the screen.

7. Implement a function *int count_words(char *str)* that receives a string and returns the number of words in the string. Consider that words are separated by a single space. Test your function with several different strings.

8. Implement a program that reads a string that represents a real number. Create two functions: one that returns an integer, referring to the integer part of the number and another that returns an integer representing the fractional part.

   Example:

```
char x[] = "123.456";
int x_int = integer_part(x);        /* assigns 123 to x_int */
int x_frac = fractional_part(x);    /* assigns 456 to x_frac */
```

9. Implement a program that, given a string that represents an integer, identifies in which format the number is represented: binary, octal, decimal or hexadecimal. It is assumed that the number is represented in the smallest base indicated.

10. Review the document "Modules and Makefiles" available in Moodle. Adapt the sample *prog_avg* program as follows:
    - Add the following function in "*average.c*": *int average_array (int v [], int n)* which calculates the average of the n integer numbers of array v;
    - The "*main.c*" file should be changed to also invoke this new feature;
    - Create a Makefile to specify the construction of *prog_avg*.

11. Change the previous exercise to add the following function to "*average.c*": *int average_global_array()* which calculates the average of the *g_n* numbers of array *g_v*. For that, you should add the following global variables:
    - `int g_n;`
    - `int g_v[100];`

    These two global variables are used to pass the values to the *average_global_array ()* function. The "*main.c*" file should be changed to also invoke this new function.

12. Create a Makefile to compile the following program. Analyze and justify the output that the program produces.

```
/***** File main.c *****/
#include <stdio.h>
#include "size_string.h"

int main() {
   char x[] = "I will master ARQCP";
   printf("Size =%u\n", sizeof(x));
   printf("Size =%u\n", size_string_wrong (x));
   printf("Size =%u\n", size_string_correct(x));

   char y[25] = "I will master ARQCP";
   printf("\n Size =%u\n", sizeof(y));
   printf("Size =%u\n", size_string_wrong (y));
   printf("Size =%u\n", size_string_correct(y));
   return 0;
}
```

```
/***** file size_string.h *****/
unsigned int size_string_wrong (char s[]);
unsigned int size_string_correct (char s[]);
```

```
/***** file size_string.c *****/
unsigned int size_string_wrong (char s[]) {
   return sizeof(s);
}

unsigned int size_string_correct (char s[]) {
   unsigned int cont=0;
   while(s[cont]!=0)
      cont++;
   return cont;
}
```

13. Write a program that prints the full multiplication table. The program must be compiled by a Makefile that makes use of variables and suffix rules. Your program should be structured as follows:

    - "*line.c*" - implements the function *void line(int x, int y)* that prints a line of a multiplication table;
    - "*multiplication_table_n.c*" - implements the function *void multiplication_table_n(int n)* that makes use of the *line()* function to print a complete table from n * 1 to n * 10;
    - "*multiplication_table.c*" - implements the function *void multiplication_table(void)* that makes use of the *multiplication_table_n()* function to print the full multiplication table;
    - "*main.c*" - invokes the *multiplication_table()* function.