



# B1 - Unix & C Lab Seminar

---

B-CPE-100

## Day 03

---

First C Programming





# Day 03

language: C



- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.



- Arrays and strings are **forbidden** for every task.
- Don't push your `main` function into your delivery directory, we will be adding our own. Your files will be compiled adding our `main.c` and our `my_putchar.c` files.
- You are only allowed to use the `my_putchar` function to complete the following tasks, but **don't push it** into your delivery directory, and don't copy it in *any* of your delivered files.
- If one of your files prevents you from compiling with `*.c`, the Autograder will not be able to correct your work and you will receive a 0.



Create your repository at the beginning of the day and submit your work on a regular basis!  
The delivery directory is specified within the instructions for each task.  
In order to keep your repository clean, pay attention to `gitignore`.



## TASK 00 - CODING STYLE

At Epitech, every C code must comply with our Coding Style: [epitech\\_c\\_coding\\_style.pdf](#).



Read it carefully! The code quality is an important factor and will be evaluated!

In order to check if your project is compliant with the coding style (for the most part), you can use the `coding-style.sh` script available in [this github repository](#).

Using it is very simple:

1. Ensure that Docker is installed on your machine.
2. Clone the [Epitech coding style checker scripts repository](#).
3. Launch the script with two parameters:
4. the path to the directories with your source code
5. the path where to put the report files containing the results
6. You have your results!



Docker and the script is preinstalled if you're using the official dump (just type `coding-style` in your shell).

Here's a basic example of how to use it:

```
Terminal
~/B-CPE-100> coding-style /path/to/your/repository .
~/B-CPE-100> cat coding-style-reports.log
```

## TASK 01 - MY\_PRINT\_ALPHA

Delivery: `my_print_alpha.c`

Write a function that, beginning with `a`, displays the lowercase alphabet in ascending order, on a single line. It must be prototyped as follows:

```
int my_print_alpha(void);
```



## TASK 02 - MY\_PRINT\_REVALPHA

---

**Delivery:** my\_print\_revalpha.c

Write a function that, beginning with **z**, displays the lowercase alphabet in descending order, on a single line. It must be prototyped as follows:

```
int my_print_revalpha(void);
```

## TASK 03 - MY\_PRINT\_DIGITS

---

**Delivery:** my\_print\_digits.c

Write a function that displays all the digits, on a single line, in ascending order. It must be prototyped as follows:

```
int my_print_digits(void);
```

## TASK 04 - MY\_ISNEG

---

**Delivery:** my\_isneg.c

Write a function that displays either **N** if the integer passed as parameter is negative or **P**, if positive or null. It must be prototyped as follows:

```
int my_isneg(int n);
```

## TASK 05 - MY\_PRINT\_COMB

Delivery: my\_print\_comb.c

Write a function that displays, in ascending order, all the numbers composed by three **different** digits numbers (012, 013, 014, 015, 016, 017, 018, 019, 023, ..., 789). Given three digits (all different), only the smallest number composed by those digits must be displayed. It must be prototyped as follows:

```
int my_print_comb(void);
```

```
Terminal
~/B-CPE-100> ./a.out
012, 013, 014, 015, 016, 017, 018, 019, 023, ..., 789
```



Neither 987 nor 999 is to be displayed (as an example).

## TASK 06 - MY\_PRINT\_COMB2

Delivery: my\_print\_comb2.c

Write a function that displays, in ascending order, all the different combinations of two two-digit numbers (00 01, 00 02, 00 03, 00 04, 00 05,...,01 99, 02 03, ..., 98 99). It must be prototyped as follows:

```
int my_print_comb2(void);
```

```
Terminal
~/B-CPE-100> ./a.out
00 01, 00 02, 00 03, 00 04, 00 05,...,01 99, 02 03, ..., 98 99
```



## TASK 07 - MY\_PUT\_NBR

---

Delivery: my\_put\_nbr.c

Write a function that displays the number given as a parameter. It must be able to display all the possible values of an `int`, and must be prototyped as follows:

```
int my_put_nbr(int nb);
```



For instance, `my_put_nbr(42)` displays 42, `my_put_nbr(0)` displays 0, `my_put_nbr(-2147483647)` displays -2147483647.

## TASK 08 - TESTING

---

Delivery: tests/tests\_my\_put\_nbr.c

It is highly recommended to test your functions as you develop them. It is common practice to create a function named `main` (and a designated file to host it) to check the functions separately.

Create a directory named `tests`.

Create a `main` function within a file named `tests_my_put_nbr.c`, to be stored in the `tests` directory.

This function must contain all the necessary calls to `my_put_nbr` in order to cover all of the function's possible situations (both regular or irregular).

For instance, for the `my_isneg` function, you could have a file similar to the following:

```
int main()
{
    my_isneg(0);
    my_isneg(21);
    my_isneg(-21);
    return (0);
}
```



Testing is an important part of software development. There is no excellence without testing.

## TASK 09 - MY\_PRINT\_COMBN

---

**Delivery:** my\_print\_combn.c

Write a function that displays, in ascending order, all the numbers composed by  $n$  **different** digits numbers ( $n$  being given as a parameter). Given  $n$  digits (all different), only the smallest number composed by those digits must be displayed. It must be prototyped as follows:

```
int my_print_combn(int n)
```



my\_print\_combn(3) gives the same result as my\_print\_comb.