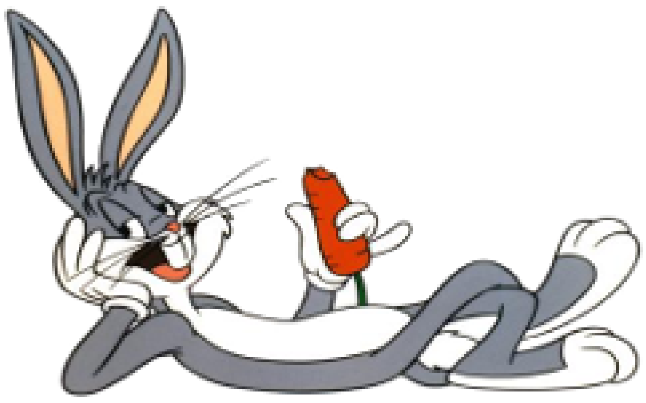

B1- C Pool

B-CPE-042

Rush#2

Luhn Code

v1.7

Rush#2

Luhn Code

binary name: : rush2
repository name: : CPool_rush2
repository rights: : ramassage-tek
language: : C
group size: : 2
compilation: : via Makefile, including *all*, *clean*, *fclean* and *re* rules

- Your repository must contain the totality of your source files, but no useless files (binary, temp files, obj files,...).
- All the bonus files (including a specific Makefile) should be in a directory named *bonus*.
- Error messages have to be written on the error output, and the program should then exit with the 84 error code (0 if there is no error).

Only the team leader's delivery repository will be evaluated.
Segfault, bus error, floating exception are all grounds for disqualification!

Allowed function: write

It is **absolutely mandatory** to complete your required project **perfectly** in order to earn bonus points for extra projects.

Team work

Here are a couple of important rules to follow:

- The **team leader** must register his/her group for the oral presentation.
- You must complete your oral presentation on Sunday, at the right time, and *with all of your partners*.
- Every member of the group should be fully aware of the work you will have completed. Each member will be questioned, and your group's grade will be based on the worst explanations.
- You have to do everything within your power to contact your partners; look at their intranet profile, find them on Facebook or by any other mean. **Excuses regarding group problems will not be accepted**.
  If, after you have tried **everything**, and your partner is still unreachable, send an email to the local staff **ASAP**.

Luhn Code

In mathematics, and more specifically in modular arithmetics, the Luhn formula is used for its applications in cryptology.

The Luhn algorithm or Luhn formula, also known as the "modulus 10" or "mod 10" algorithm, is a simple checksum formula used to validate a variety of identification numbers, such as credit card numbers, IMEI numbers, and Canadian Social Insurance Numbers.

It was created in the 1960s, by German IBM engineer Hans Peter Luhn as a method used to validate identification numbers. It has become famous namely because of its adoption by credit card companies right after it was created.

The algorithm is in the public domain and is in wide use today. It is not intended to be a cryptographically secure hash function; it was designed to protect against accidental errors, not malicious attacks.

Most credit cards and many government identification numbers use the algorithm as a simple method of distinguishing valid numbers from random numbers collections.

Part 1 - Validation of Luhn Code

The goal of this part is to create a program which verifies the validity of a Luhn code.

Your program should accept any given numbers after the '**-c**' flag and verify it's validity.
You must display '**OK**' if the number is a valid Luhn code or '**KO**' if it's not (on the standard output).
In case of an error (such as missing parameters, invalid flag, etc.), your program should display a message explaining the error on the **standard error output**.
The program must work as follows:

```
Terminal
~/B-CPE-042> ./rush2 -c 5140256942783646 | cat -e
OK$
```

```
Terminal
~/B-CPE-042> ./rush2 -c 5140256942783546 | cat -e
KO$
```

Watch out! The return value is also important. The program must exit with the value '**0**' if everything went smoothly, or with the value '**84**' in case of error.

Part 2 - Calculation of Luhn key

The goal of this part is add a functionality to your program: complete the code given as parameter.

More specifically, you have the complete the given number by displaying the missing digit.
For instance, in case of a bank number, you'll be given the first 15 numbers and you need to find and display the 16th number.

The program must be prototyped as follows:

```
Terminal
~/B-CPE-042> ./rush2 -f 192924593889831 | cat -e
6$
```

Watch out! The return value is also important. The program must exit with the value '**0**' if everything went smoothly, or with the value '**84**' in case of error.

Bonus

Now, you know how to verify numbers with the Luhn algorithm. You can try to add more option to the program, such as generating valid bank numbers, IMEI and other code based on the Luhn formula.
Or, you could try to implement other checksum or error detection algorithms, such as the Verhoeff algorithm or the Damm algorithm.

Every bonus you make must be placed in a subdirectory called **bonus**.