Takuzu

Assignment 1: User Interface

The objective of this assignment is to start the project: set-up a git repository, create the file hierarchy for your project, write a build system (Makefile) and develop a user interface through basic command line arguments.

For this assignment you will work by groups on two.

1. Setting the GIT repository

- Start by reading the git tutorials cited in the lecture (if lecture is not sufficient)
- Set up your global git configuration (name, email, editor)
- Go to the ISTIC gitlab server (https://gitlab.istic.univ-rennes1.fr)
- Create a repository named *takuzu*
    - Invite your buddy (binôme) as a developer
    - Invite the professor in charge of your group + the professor in charge of the lecture (Isabelle Puaut)
- Get a local version of your repository on your computer (*git clone*, through *https* or *ssh* if you have set your SSH key on the Gitlab server – strongly recommended)

From now, you are able to pull and push from any place.

2. Create file hierarchy

1. Create the file hierarchy (with empty files for now) that contains the following files:

```
Makefile
include/
      \-- takuzu.h
src/
      |-- Makefile
      \-- takuzu.c
doc/
tests/
```

The `include/takuzu.h` file must contain:

```c
#ifndef TAKUZU_H
#define TAKUZU_H

#endif /* TAKUZU_H */
```

2. Write a `main()` function with the following prototype: `int main (int argc, char*argv[])` that will simply write `Hello World!` on `stdout`.
3. Write the `file src/Makefile` with the following targets
    a. all: generate the takuzu binary file from the source files
    b. clean: remove all temporary files + binary file generated by the compilation
    c. help: display the targets of the `Makefile` with a short description

    The `Makefile` should define variables CFLAGS, CPPFLAGS, LDFLAGS and the target .PHONY

4. Create the file `Makefile` at the root of the project and write the same targets than previously (all, clean, help). This Makefile is expected to be used by users of your project that want to run it and do not want to know the files it contains.

3. Options and arguments parser

Write an options and arguments parser that supports the following options, even if the code for most of them will be just empty for now:

```
sh> ./takuzu -h
Usage:      takuzu [-a|-o FILE|-v|-h] FILE
            takuzu -g[SIZE] [-u|-o FILE|-v|-h]
Solve or generate takuzu grids of size: 4, 8 16, 32, 64
-a, --all               search for all possible solutions
-g[N], --generate[=N]   generate a grid of size NxN (default:8)
-o FILE, --output FILE  write output to FILE
-u, --unique            generate a grid with unique solution
-v, --verbose           verbose output
-h, --help              display this help and exit
```

1. Set up the option parser structure using getopt_long() to support the `'-h'` or `'--help'` option and exit with EXIT_SUCCESS
2. Manage option `'-v'` or `'--verbose'` to set up a Boolean variable `verbose` to true if the option is set.
3. Manage the option `'-o'` or `'--output'` to print the output of the software to a file (by default, `stdout` is used). The expected information to be stored in the output file will be the important result of the software only (ability to solve the grid, number of solutions, solution(s) if any).
   Suggested functions to use: `fopen(), fclose()`.
4. Manage the options `'-a'` and `'-u'` and relate them to variables `bool all`, and `bool unique`. In general, using global variables is not recommended but using a global structure that stores the software configuration is allowed for the project.
5. Manage the option `'-g'` to store the configuration options. The code will check that N is an integer and is among supported grid sizes (4/8/16/32/64).
   Suggested functions to use: `strtol(), atoi()`.
6. The software will have two modes: solver or generator (solver mode by default, generation mode when option `'-g'` is set). Write the code needed to detect in which mode we are and exit with an error when inconsistent options are found in the command line (`'-u'` on solver mode, `'-a'` in generation mode). Check that all parameters are given (grid to solve in solver mode, with readable file). If one of these prerequisites is not met, then fail by displaying an error message and return EXIT_FAILURE.

Here is an example of output:

```
sh> ./takuzu
takuzu: error: no input grid given!
sh> ./takuzu -u
takuzu: warning: option 'unique' conflict with solver mode,exiting!
```

Suggested functions to use: `perror()`, `err()`, `errx()`, `warn()`, `warnx()`.

4. Submitting your first assignment

Once you wish to submit your assignment, create a for your first assignment and push your code to the central repository.

```
sh> git commit
sh> git tag assignment-1
sh> git push --set-upstream origin assignment-1
```

Each assignment will be assigned a different tag (`assignment-1`, … `assignment-6`).

Then call the professor to check your code and test it operates properly.