# PW1 - Makefile

**Objective:** Discover `Makefile`.

As a beginning, subscribe to the GRAF course on Moodle (key is graf-M1-info) :
Masters / ▶ M1 - Informatique / ▶ M1 - Semestre 7 / ▶ M1 GRAF

Copy in a directory, in your workspace, the files contained into the Moodle "ZIP archive of the files to compile".

These files are `premier.c`, `fctPermier.c`, `fctPermier.h`, `saisie.c` and `saisie.h` (sorry the file names are in French!). They are for testing if an entered number is a prime number or not. The main program is `premier.c`, the input function is in the file `saisie.c`, it is declared inside `saisie.h`, the test function is in the file `fctPremier.c` and is declared in `fctPremier.h`.

## Step 1 : Compilation "at hand"

Compile this program at hand, i.e. by typing the following instructions under `bash`:

```
smith[TP1]% gcc -c saisie.c
smith[TP1]% gcc -c fctPremier.c
smith[TP1]% gcc -o premier premier.c saisie.o fctPremier.o -lm
```

When one wants to re-compile, he has to type again everything, without knowing if all files have need being compiled again. The `make` command allows to automate the compilation, and takes the dependencies between files into account so that only the files that actually need to be compiled again are compiled. For example, the modification of a `.h` file can result in compiling all the files in which it is included, and only those files.

## Step 2 : First `makefile`

The `make` command interprets a file named `makefile` or more usually `Makefile`. If this file has to be named differently (such as "`myName`" for example), it is possible to type `make -f myName`. In this file we define the variables useful to the compilation and set rules such as:

```
target1 : dep1 dep2 ...
          compilation command or other
```

The dependencies are either other targets, or files whose last modification date will launch or not the command `command`. Between the margin et the command, it is mandatory to have at least one *tabulation*. Thus the following makefile:

```
# Makefile 1

# main rule with link edition
premier : premier.c saisie.o fctPremier.o
        gcc -o premier premier.c saisie.o fctPremier.o -lm

# Rule 1
saisie.o : saisie.c saisie.h
```

```
        gcc -c saisie.c

# Rule 2
fctPremier.o : fctPremier.c fctPremier.h
        gcc -c fctPremier.c

# Clean the intermediate files
clean :
        rm *.o

# suppress all the files resulting from the compilation
veryclean : clean
        rm premier
```

## Step 3 : Building 6 other `makefile`

Write successive `makefile` for using:

- variables
- internal variables (see Table 1)

| | |
|---|---|
| `$@` | Target name |
| `$<` | First dependency name |
| `$^` | Dependencies name |
| `$?` | List of all dependencies more recent than the target |
| `$*` | File name without suffix |

Table 1: Internal Variables

- inference rules
- list of source and object files
- automatic list of source and object files
- building of directories and libraries
- ...

There are many ways of writing makefiles. Several compilation techniques are available. A GNU `makefile` documentation can be found at:

- `http://www.gnu.org/software/make/manual/make.html`
- `http://fr.wikipedia.org/wiki/GNU_Make`.