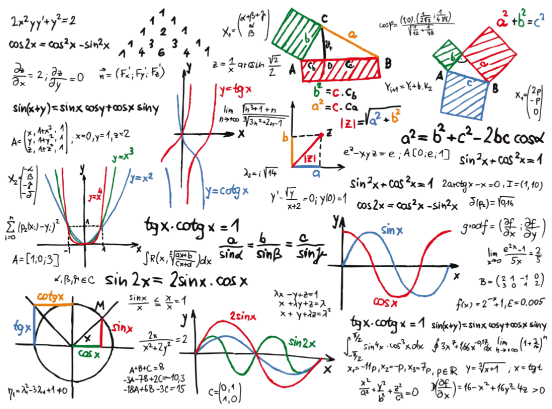

B-1- Mathematics

B-MAT-500

303make

Makefile and Dependencies

v1.0

303make

Makefile and Dependencies

binary name:	303make
repository name:	303make
repository rights:	ramassage-tek
language:	C, C++, perl 5, python 3 (≥ 3.5), ruby 2 (≥ 2.2), php 5.6, bash 4
group size:	1 to 2
compilation:	via Makefile, including re, clean and fclean 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 potential 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).

Subject

As you know, when compiling a program, only the files that are dependent on recently modified sources are effectively recompiled during a Make function call. In order to do this, this binary parses the Makefile in the current folder so that a dependency graph can be generated. Next, it analyzes this graph in order to execute the minimum amount of compilations.

In this project you must simulate the behavior of Make by reconstructing the dependency graph, and then display all of the commands to be executed after a source file is modified.
First, you'll need to display the adjacency matrix and the dependency strings.

Makefile's variables aren't taken into account.

Pay attention to the scheduling of the compilation tasks!

Prototyping

```
Terminal
~/B-MAT-500> ./303make makefile [file]
```

makefile contains the name of a file that replaces the Makefile
file contains the name of the recently modified file

Bonus

- Makefile variable management
- Management of several source files

Examples

```
Terminal
~/B-MAT-500> cat MakefileTest
tty:  tty.o fc.o
cc -o tty tty.o fc.o

tty.o:  tty.c fc.h
cc -c tty.c

fc.o:  fc.c fc.h
cc -c fc.c
```

```
Terminal
~/B-MAT-500> ./303make MakefileTest
[0 0 0 0 0 0]
[0 0 0 0 0 0]
[1 1 0 0 0 0]
[0 0 1 0 0 1]
[0 0 0 0 0 0]
[0 1 0 0 1 0]

fc.c -> fc.o -> tty
fc.h -> fc.o -> tty
fc.h -> tty.o -> tty
fc.o -> tty
tty.c -> tty.o -> tty
tty.o -> tty
```

For the second part of the above test, the dependency strings are in alphabetical order.

Terminal

```
~/B-MAT-500> ./303make MakefileTest tty.c
cc -c tty.c
cc -o tty tty.o fc.o
```

Terminal

```
~/B-MAT-500> ./303make MakefileTest fc.h
cc -c fc.c
cc -c tty.c
cc -o tty tty.o fc.o
```

Terminal

```
~/B-MAT-500> ./303make MakefileTest tty

~/B-MAT-500>  ./303make MakefileTest tty > /dev/null ; echo $?
0
```

Terminal

```
~/B-MAT-500> ./303make MakefileTest totoy > /dev/null ; echo $?
84
```