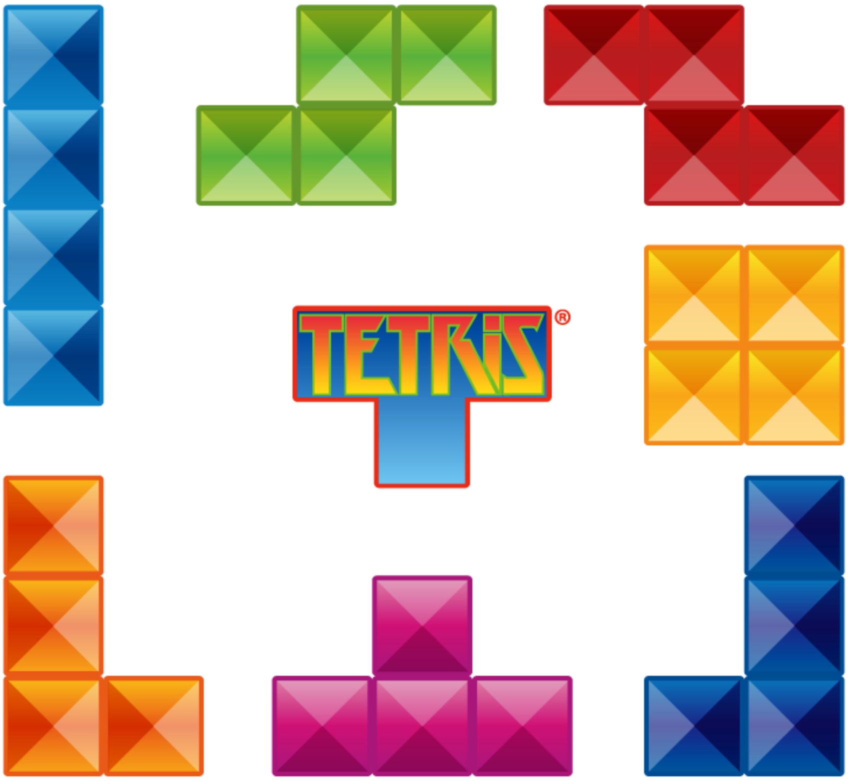
# Chapitre II

# Introduction

`Fillit` is a project who let you discover and/or familiarize yourself with a recurring problematic in programming : searching for the optimal solution among a huge set of possibilities. In this particular project, you will be charged of creating an algorithm which fits some `Tetriminos` together into the smallest possible square.

A `Tetriminos` is a 4-blocks geometric figure that most of you might knows thanks to the popular game `Tetris`.



# Chapitre V

# Mandatory part

## V.1 Program entry

Your executable must take as parameter one (and only one) file which contains a list of `Tetriminos` to arrange. This file has a very specific format : every `Tetriminos` description consists of 4 lines and each `Tetriminos` are **separated by an empty line**

If the number of parameters given to your executable is different from `1`, your program must display his `usage` and exit properly. If you dont know what a `usage` is, execute the command `cp` without arguments in your Shell to get an idea.

The description of a `Tetriminos` must respect the following rules :

- Precisely 4 lines of 4 characters followed by a new line.
- A `Tetriminos` is a classic piece of `Tetris` composed of 4 blocks.
- Each character must be either a `'#'` when the character is one of the 4 blocks of a `Tetriminos` or a `'.'` if it's empty.
- Each block of a `Tetriminos` must be in contact with at least one other block on any of his 4 sides.

A few examples of valid descriptions of `Tetriminos` :

```
....    ....    ####    ....    .##.    ....    .#..    ....    ....
..##    ....    ....    ....    ..##    .##.    ###.    ##..    .##.
..#.    ..##    ....    ##..    ....    ##..    ....    #...    ..#.
..#.    ..##    ....    ##..    ....    ....    ....    #...    ..#.
```

A few examples of invalid descriptions of `Tetriminos`

```
####    ...#    ##...   #.      ....    ..##    ####    ,,,,    .HH.
...#    ..#.    ##...   ##      ....    ....    ####    ####    HH..
....    .#..    ....    #.      ....    ....    ####    ,,,,    ....
....    #...    ....            ....    ##..    ####    ,,,,    ....
```

Because each Tetriminos occupies only 4 of the 16 available boxes, it is possible to describe the same Tetrimino in multiple ways. However, the rotation of a Tetrimino describes a different Tetrimino from the original in the case of this project. This means that no rotation is possible on a Tetrimino, when you will arrange it with the others.

Those Tetriminos are then perfectly equivalents on every aspect :

```
##..    .##.    ..##    ....    ....    ....
#...    .#..    ..#.    ##..    .##.    ..##
#...    .#..    ..#.    #...    .#..    ..#.
....    ....    ....    #...    .#..    ..#.
```

These 5 Tetriminos are, for their part, 5 entirely distinct Tetriminos on every aspect :

```
##..    .###    ....    ....    ....
#...    ...#    ...#    ....    .##.
#...    ....    ...#    #...    .##.
....    ....    ..##    ###.    ....
```

To finish, here is an example of a valid file your program must resolve :

```
$> cat -e valid_sample.fillit
...#$
...#$
...#$
...#$
$
....$
....$
....$
####$
$
.###$
...#$
....$
....$
$
....$
..##$
.##.$
....$
$>
```

As well as an example of invalid file your program must reject for multiple reasons :

```
$> cat -e invalid_sample.fillit
...#$
...#$
...#$
...#$
....$
....$
....$
####$
$
$
.###$
...#$
....$
....$
$
....$
..##$
.##.$
....$
$>
```

## V.2 The smallest square

The goal of this project is to arrange the Tetriminos among themselves to make the smallest possible square, but in some cases, this square may have holes when some given pieces won't fit perfectly with others.
Each Tetrimino, even if presented on a 16 box grid, is only defined by its full boxes (his '#'). The 12 remaining Tetriminos will be ignored for the arrangement of Tetriminos among themselves.

The Tetriminos are ordered as they appear in the file. Among the different solutions possible to make the smallest square, only the solution where Tetriminos is placed on their most upper-left position, will be accepted/retained.

Example :

If we consider the two following Tetriminos (the '#' are replaced by digits for understanding purposes) :

```
1...    ....
1...    ....
1... AND ..22
1...    ..22
```

The smallest square formed by those 2 pieces is 4 boxes wide, but there is many versions that you can see right below :

```
a)      b)      c)      d)      e)      f)
122.    1.22    1...    1...    1...    1...
122.    1.22    122.    1.22    1...    1...
1...    1...    122.    1.22    122.    1.22
1...    1...    1...    1...    122.    1.22

g)      h)      i)      j)      k)      l)
.122    .1..    .1..    221.    ..1.    ..1.
.122    .122    .1..    221.    221.    ..1.
.1..    .122    .122    ..1.    221.    221.
.1..    .1..    .122    ..1.    ..1.    221.

m)      n)      o)      p)      q)      r)
22.1    .221    ...1    ...1    ...1    ...1
22.1    .221    22.1    .221    ...1    ...1
...1    ...1    22.1    .221    22.1    .221
...1    ...1    ...1    ...1    22.1    .221
```

According to the rule above, the right solution is then a)

## V.3 Program output

Your program must display the smallest square solution on the standard output. To identify each `Tetriminos` in the square solution, you will assign a capital letter (starting with `'A'`) to each `Tetriminos` in the order they appear in the file. A file will contain 26 `Tetriminos` maximum.

If the file contains at least one error, your program must display `error` on the standard output and will exit properly.

*Example :*

```
$> cat sample.fillit | cat -e
....$
##..$
.#..$
.#..$
$
....$
####$
....$
....$
$
#...$
###.$
....$
....$
$
....$
##..$
.##.$
....$
$> ./fillit sample.fillit | cat -e
DDAA$
CDDA$
CCCA$
BBBB$
$>
```

*Another example :*

```
$> cat sample.fillit | cat -e
....$
....$
####$
....$
$
....$
...$
..##$
..##$
$> ./fillit sample.fillit | cat -e
error$
$>
```

*Last Example :*

```
$> cat sample.fillit | cat -e
...#$
...#$
...#$
...#$
$
....$
....$
....$
####$
$
.###$
...#$
....$
....$
$
....$
..##$
.##.$
....$
$
....$
.##.$
.##.$
....$
$
....$
....$
##..$
.##.$
$
##..$
.#..$
.#..$
....$
$
....$
###.$
.#..$
....$
$> ./fillit sample.fillit | cat -e
ABBBB.$
ACCCEE$
AFFCEE$
A.FFGG$
HHHDDG$
.HDD.G$
$>
```

## V.4 Automatic correction

Due to the demanding nature of the moulinette, we ask you to respect the same turn-in protocol asked by the libft. All of your program sources and headers must be in the same folder. You can have two different folders, one for the libft and one for fillit.