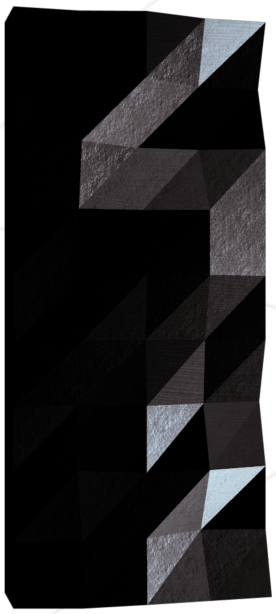


Fillit

Can you feel it ?

Team pedago pedago@staff.42.fr

Résumé: How to fit an elephant in a glass of whiskey ?

Table des matières

Chapitre I

Forewords

Alexey Leonidovich Pajitnov is a Russian video game designer and computer engineer who developed the popular game Tetris while working for the Dorodnitsyn Computing Centre of the Soviet Academy of Sciences, a Soviet government-founded R&D center.

Pajitnov was born on March 14, 1956 in Moscow. As a child, he was a fan of puzzles and played with pentomino toys. In creating Tetris, he drew inspiration from these toys.

Pajitnov created Tetris with the help of Dmitry Pavlovsky and Vadim Gerasimov in 1984. The game, first available in the Soviet Union, appeared in the West in 1986.

Pajitnov also created the lesser known sequel to Tetris, entitled Welltris, which has the same principle but in a three dimensional environment where the player sees the playing area from above. Tetris was licensed and managed by Soviet company ELORG which had been founded especially for this purpose, and advertised with the slogan "From Russia with Love" (on NES : "From Russia With Fun !"). Because he was employed by the Soviet government, Pajitnov did not receive royalties.

Pajitnov, together with Vladimir Pokhilko, moved to the United States in 1991 and later, in 1996, founded The Tetris Company with Henk Rogers. He helped design the puzzles in the Super NES versions of Yoshi's Cookie and designed the game Pandora's Box, which incorporates more traditional jigsaw-style puzzles.

He was employed by Microsoft from October 1996 until 2005. While there he worked on the Microsoft Entertainment Pack : The Puzzle Collection, MSN Mind Aerobics and MSN Games groups. Pajitnov's new, enhanced version of Hexic, Hexic HD, was included with every new Xbox 360 Premium package.

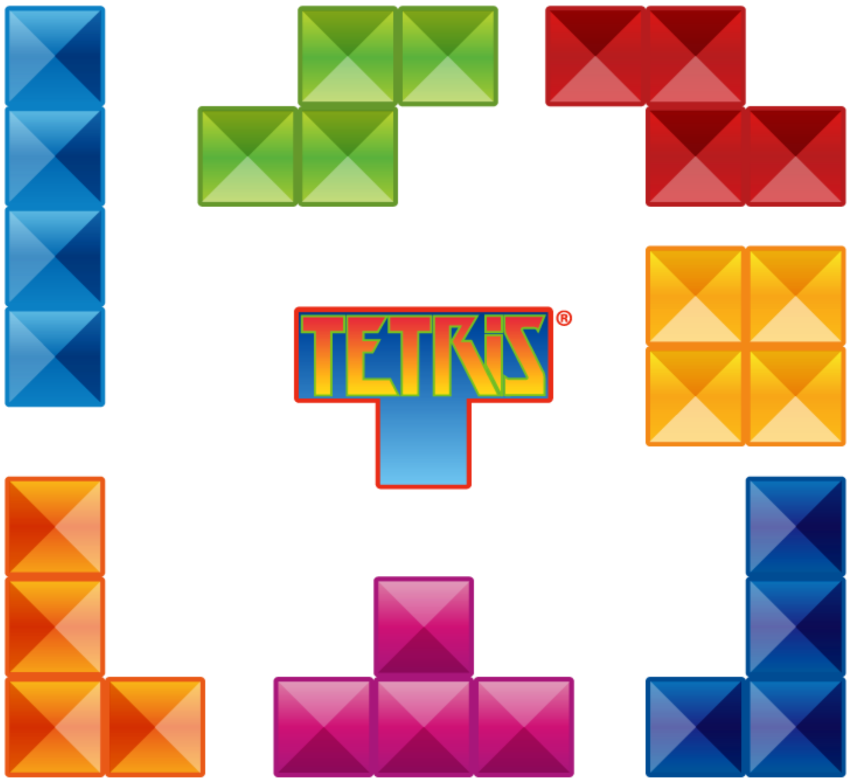
On August 18, 2005, WildSnake Software announced Pajitnov will be collaborating with them to release a new line of puzzle games.

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 III

Goals

Fillit does not consist of recoding Tetris, even if it's still a variant of this game. Your program will take a file as parameter which contains a list of Tetriminos and arrange them to create the smallest square possible. Obviously, your main goal is to find this smallest square in the minimal amount of time, despite a exponentially growing number of possibilities each time a piece is added.

Then, you will have to think carefully about your data structures and your solving algorithm, if you want your program answers before the next millenium.

Chapitre IV

General instructions

- Your project must be written in C and must be according to the Norme coding standard.
- The allowed functions are : `exit`, `open`, `close`, `write`, `read`, `malloc` and `free`.
- Your Makefile must compile your work and present the following rules : `all`, `clean`,`fclean` et `re`.
- You must compile your executable with the flags `Wall`, `Wextra` and `Werror`. Any other flag, and especially for optimising purposes, are forbidden.
- The executable must be named `fillit` and located in the root directory of your repository.

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.

Chapitre VI

Return and peer-evaluation

Return your work on your deposit `GiT` as per usual. Only the files present on your deposit will be submitted to evaluation.

Your work will be evaluated by a moulinette (additional to any peer-evaluation depending on your CURSUS/=COURSE). This moulinette includes an arbitrary timeout that will stop the execution of your program if it takes too long to find a solution to a test. This test will be then considered false, obviously.