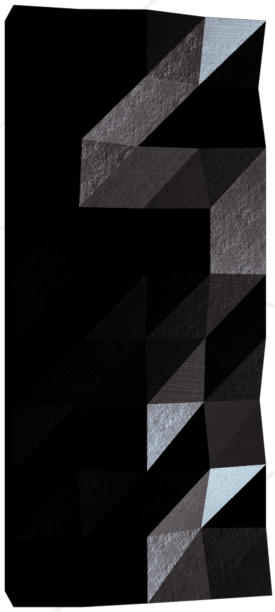


Fillit

Can you feel it ?

Pedago Team pedago@42.fr

Summary: This is the story of a piece of **`Tetris`**, *one little square and a dev walk into a bar...*

Contents

Chapter 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.

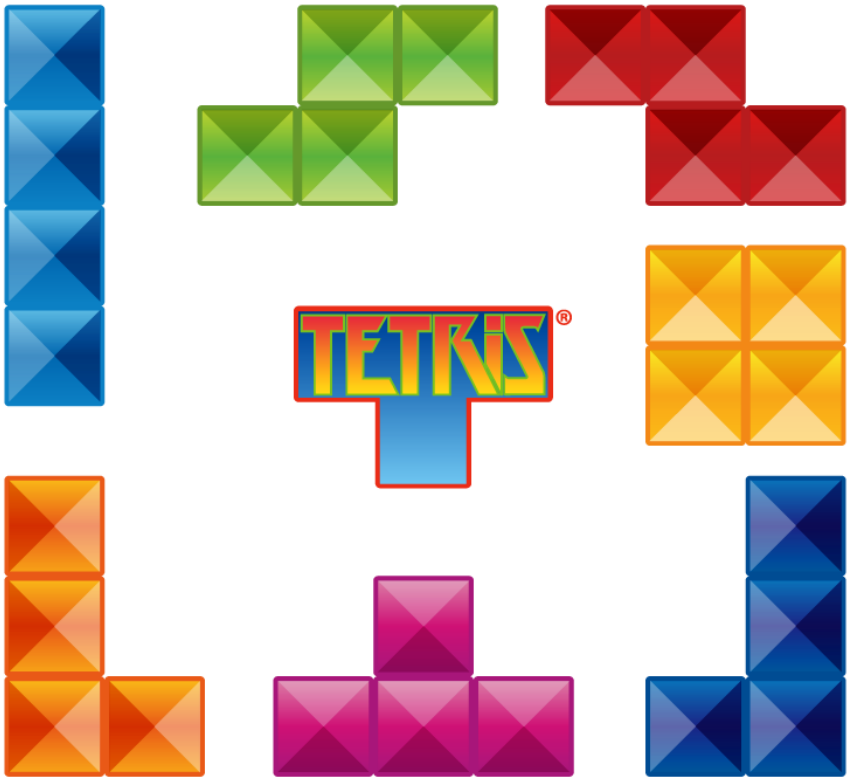
On August 18, 2005, WildSnake Software announced Pajitnov will be collaborating with them to release a new line of puzzle games.

Chapter II

Introduction

`Fillit` is a project that let you discover and/or familiarize yourself with a recurring problematic in programming: searching the optimal solution among a huge set of possibilities, in a respectable timing. In this particular project, you will have to find a way to assemble a given `Tetriminos` set altogether in the smallest possible square.

A `Tetriminos` is a 4-blocks geometric figure you probably already heard of, thanks to the popular game `Tetris`.



Chapter III

Goals

`Fillit` is not about 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 in order to create the smallest square possible.

Obviously, your main goal is to find the smallest square in the minimal amount of time, despite an exponentially growing number of possibilities each time a piece is added.

You should think carefully about how you will structure your data and how to solve this problem, if you want your program answers before the next millenium.

Chapter IV

General instructions

- Your project must be written in C and must respect the Norme coding standard.
- The allowed functions are : `exit`, `open`, `close`, `write`, `read`, `malloc` and `free`.
- Your Makefile must compile your code without relinks.
- It must contain the following rules : `all`, `clean` ,`fclean` and `re`.
- You must compile your binary with the `Wall`, `Wextra` and `Werror` flags. Any other flag are forbidden , especially those for optimising purposes.
- The binary must be named `fillit` and located in the root directory of your repository.
- You must submit a file called `author` containing your username followed by a '\n' at the root of your repository:

```
$>cat -e author
xlogin$
ylogin$
```

- Your project cannot leaks.

Chapter V

Mandatory part

V.1 Program entry

Your executable must take only one parameter, a file which contains a list of `Tetriminos` to assemble. This file has a very specific format : every `Tetrimino` must exactly fit in a 4 by 4 chars square and all `Tetrimino` are **separated by an newline** each.

If the number of parameters sent to your executable is not `1`, your program must display its `usage` and exit properly. If you don't know what a `usage` is, execute the command `cp` without arguments in your Shell. It will give you an idea. Your file should contain between 1 and 26 `Tetriminos`.

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

- Precisely 4 lines of 4 characters, each followed by a new line (well... a 4x4 square).
- A `Tetrimino` is a classic piece of `Tetris` composed of 4 blocks.
- Each character must be either a block character(`'#'`) or an empty character (`'.'`).
- Each block of a `Tetrimino` must touch at least one other block on any of his 4 sides (up, down, left and right).

A few examples of valid descriptions of `Tetriminos`:

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

A few examples of invalid descriptions of `Tetriminos`

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

Because each `Tetrimino` fills only 4 of the 16 available boxes, it is possible to describe the same Tetrimino in multiple ways. However, a rotated `Tetrimino` describes a different `Tetrimino` from the original, in the case of this project. This means 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 distincts `Tetriminos` on every aspect :

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

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

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

...and 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 every `Tetriminos` with each others in order to make the smallest possible square. But in some cases, this square should contains holes when some given pieces won't fit in perfectly with others.
Even if they are embedded in a 4x4 square, each `Tetrimino` is defined by its minimal boundaries (their `'#'`). The 12 remaining empty characters will be ignored for the `Tetriminos` assemble process.

`Tetriminos` are ordered by they apparition order in the file. Among all the possible candidates for the smallest square, we only accept the one where Tetriminos is placed on their most upper-left position.

Example :

Considering the two following `Tetriminos` (`'#'` will be replaced by digits for understanding purposes):

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

The smallest square you can make with those 2 pieces is 4-char wide, but there is many possible 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 assembled square on the standard output. To identify each `Tetrimino` in the square solution, you will assign a capital letter to each `Tetrimino`, starting with `'A'` and increasing for each new `Tetrimino`.

If the file contains at least one error, your program must display `error` on the standard output followed by a newline and have to 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

Because of the strictness of the moulinette, you must respect the same turn-in protocol as the libft one. All your sources and headers must be in the same folder. You can have two different folders, one for the libft and one for fillit.

Chapter VI

Turn-in and peer-evaluation

Turn-in your work on your `GiT` repository as usual. Only the files available in this repository will be evaluated.

After peer-evaluations, your work will be submitted to the Moulinette (additional to any peer-evaluation depending on your Cursus). This moulinette includes an arbitrary timeout that will stop the execution of your program if it takes too long to find a solution for a given test. This test will be then considered as false, obviously.