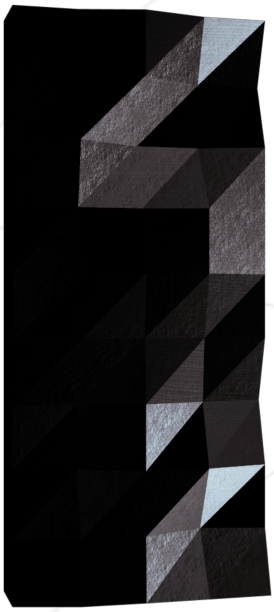


Fillit

Can you feel it ?

Pedago Team pedago@42.fr

Résumé: C'est l'histoire d'une pièce de **`Tetris`**, *d'un petit carré et d'un(e) dev qui entrent dans un bar...*

Table des matières

Chapitre I

Préambule

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.

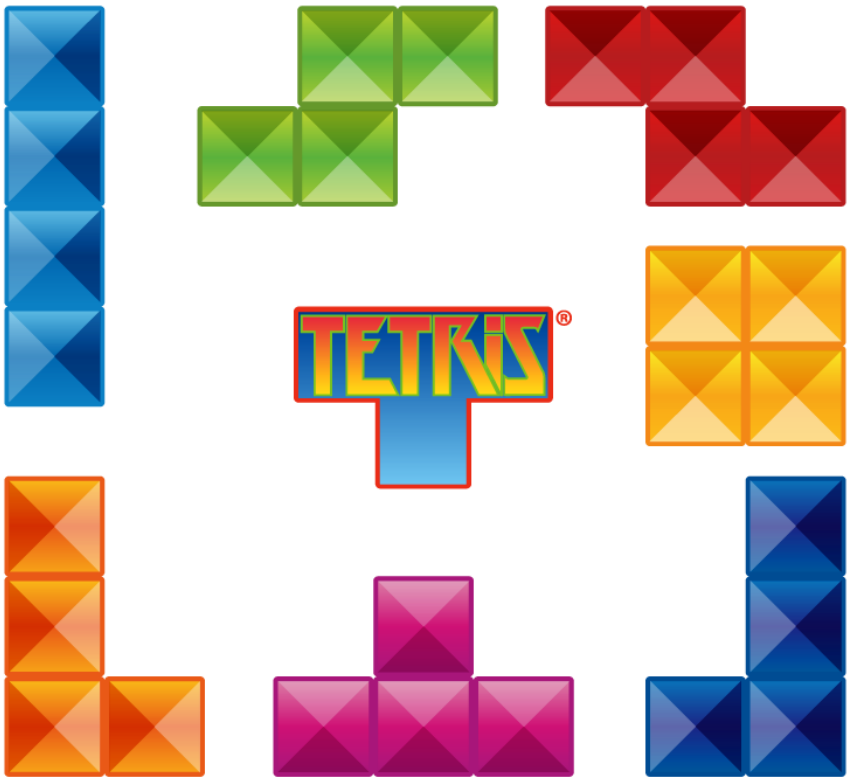
On August 18, 2005, WildSnake Software announced Pajitnov will be collaborating with them to release a new line of puzzle games.

Chapitre II

Introduction

`Fillit` est un projet vous permettant de decouvrir et/ou de vous familiariser avec une problematique récurrente en programmation : la recherche d'une solution optimale parmi un très grand nombre de possibilités, dans un délai raisonable. Dans le cas de ce projet, il s'agira d'agencer des `Tetriminos` entre eux et de déterminer le plus petit carré possible pouvant les acueillir.

Un `Tetriminos` est une figure géométrique formée de 4 blocs que vous connaissez grâce au célèbre jeu `Tetris`.



Chapitre III

Objectifs

`Fillit` ne consiste pas à recoder `Tetris`, mais reste une variante du jeu dans l'esprit. Votre programme prendra en paramètre un fichier décrivant une liste de `Tetriminos` qu'il devra ensuite agencer entre eux pour former le plus petit carré possible. Le but est bien entendu de trouver ce plus petit carré le plus rapidement possible malgré un nombre d'agencements qui croît de manière explosive avec chaque pièce supplémentaire.

Vous devrez donc bien réflechir à vos structures de données et à votre algorithme de résolution pour que votre programme réponde avant l'an 3000.

Chapitre IV

Consignes générales

- Votre projet doit être en C et à la Norme.
- Les fonctions autorisées sont : `exit`, `open`, `close`, `write`, `read`, `malloc` et `free`.
- Votre Makefile devra compiler votre rendu et proposer au moins les règles suivantes : `all`, `clean`, `fclean` et `re`.
- Vous devez compiler votre exécutable avec les flags `Wall`, `Wextra` et `Werror`. Tout autre flag, et en particulier d'optimisation, est interdit.
- L'exécutable doit s'appeller `fillit` et se trouver dans le répertoire racine de votre dépot.

Chapitre V

Partie obligatoire

V.1 L'entrée du programme

Votre exécutable doit prendre en paramètre un (et un seul) fichier décrivant la liste des `Tetriminos` à agencer. Ce fichier est formaté de façon très précise : chaque description d'un `Tetriminos` est sur 4 lignes et deux `Tetriminos` sont **séparés par une ligne vide**.

Si le nombre de paramètres passés à votre exécutable est différent de `1`, votre programme doit afficher son `usage` et quitter proprement. Si vous ignorez ce qu'est un `usage`, lancez la commande `cp` sans argument dans votre shell pour vous faire une idée. Il n'y aura jamais plus de 26 `Tetriminos` dans un fichier de description.

La description d'un `Tetrimino` doit respecter les règles suivantes :

- Exactement 4 lignes de 4 caractères suivis d'un retour à la ligne.
- Un `Tetriminos` est une pièce de `Tetris` classique composée de 4 blocs.
- Chaque caractère doit être, soit un `'#'` lorsque la case correspond à l'un des 4 blocs d'un `Tetriminos`, soit un `'.'` lorsque la case est vide.
- Chaque bloc d'un `Tetriminos` doit être en contact avec au moins un autre bloc sur l'un ou l'autre de ses 4 côtés.

Quelques exemples de descriptions de `Tetriminos` valides :

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

Quelques exemples de descriptions de `Tetriminos` invalides :

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

Chaque `Tetriminos` n'occupant que 4 cases des 16 cases disponibles, il est donc possible de décrire le même Tetriminos de plusieurs façons différentes. Toutefois, la rotation d'un `Tetriminos` décrit un `Tetriminos` différent de l'original dans le cadre de ce projet. Cela signifie qu'aucune rotation n'est possible sur un `Tetriminos` lorsque vous l'agencerez par rapport aux autres.

Ces `Tetriminos` sont donc parfaitement équivalents à tous points de vue :

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

Ces 5 `Tetriminos` sont, quand à eux, 5 `Tetriminos` parfaitement distincts à tous points de vue :

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

Pour terminer, voici un exemple de fichier de description valide que votre programme doit accepter de résoudre,

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

Ainsi qu'un exemple de fichier de description invalide que votre programme doit rejeter pour plusieurs raisons :

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

V.2 Le plus petit carré

Le but de ce projet est d'agencer les `Tetriminos` entre eux pour former le plus petit carré possible, sachant que ce carré peut présenter des trous quand les pièces données ne s'emboîtent pas parfaitement.

Chaque `Tetriminos`, bien que présenté sur une grille de 16 cases, n'est défini que par ses cases pleines (ses `'#'`). Les 12 `'.'` restants sont ignorés pour l'agencement des `Tetriminos` entre eux.

Les `Tetriminos` sont placés dans l'ordre dans lequel ils apparaissent dans le fichier. Parmi les différentes solutions possibles réalisant le plus petit carré, sera retenue la solution où chaque Tetriminos est disposé le plus en haut, puis le plus à gauche possible, au moment de son placement.

Exemple :

Considèrons les deux `Tetriminos` suivants (les `'#'` sont remplacés par des chiffres pour simplifier la lecture des resulats) :

```
1...    ....
1...    ....
1... ET ..22
1...    ..22
```

Le plus petit carré formé par ces 2 pièces fait 4 cases de coté, mais il en existe plusieurs versions que vous pouvez voir ci-dessous :

```
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
```

D'après les règles du jeu, la bonne solution est donc `a)`.

V.3 La sortie du programme

Votre programme doit afficher le plus petit carré solution sur la sortie standard. Pour pouvoir identifier chaque `Tetriminos` dans le carré solution, vous assignerez une lettre majuscule (en commencant avec `'A'`) à ce `Tetriminos` dans l'ordre où ils apparaissent dans le fichier de description.

Si le fichier de description comporte au moins une erreur, votre programme doit afficher `error` sur la sortie standard et quitter proprement.

Exemple :

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

Autre Exemple :

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

Dernier Exemple :

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

V.4 Correction automatique

La Moulinette étant quelque peu exigeante, nous vous demandons de respecter la même norme de rendu que pour la libft. Vous devez rendre vos sources et headers du fillit dans un unique dossier, et les sources et headers de votre libft dans un unique dossier également.

Chapitre VI

Rendu et peer-évaluation

Rendez-votre travail sur votre dépot `GiT` comme d'habitude. Seul le travail présent sur votre dépot sera évalué.

Après vos peer-evaluations, votre travail sera évalué par une Moulinette. Cette Moulinette aura un timeout arbitraire qui arrêtera l'exécution de votre programme si celui-ci met trop de temps à trouver la solution d'un test. Ce test sera alors compté comme faux, bien entendu.