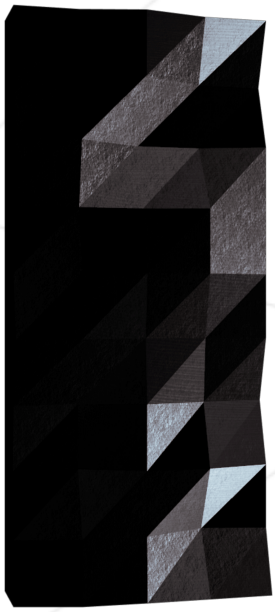

Go Piscine

Rush 02

Summary: This document is the subject for Rush02 of the Go Piscine @ 42Tokyo.

Contents

Chapter I

Instructions

- Each member of the group can register the whole group to defense.
- The group MUST be registered to defense.
- Any question concerning the subject would complicate the subject.
- You have to follow the submission procedures for all your exercises.
- This subject could change up to an hour before submission.
- You will have to handle errors coherently. Feel free to either print an error message, or simply return control to the user.
- Rushes exercises have to be carried out by group of 2, 3 or 4.
- You must therefore do the project with the imposed team and show up at Your defense slot, with all of your teammates.
- You project must be done by the time you get to defense. The purpose of defense is for you to present and explain your work.
- Each member of your group must be fully aware of the works of the project. Should you choose to split the workload, make sure you all understand what everybody's done. During the defense, you'll be asked questions and the final grade will be based on the worst explanations.
- It goes without saying, but gathering the group is your responsibility. You've got all the means to get in contact with your teammates: phone, email, carrier pigeon, spiritism, etc. So don't bother blurping up excuses. Life isn't fair, that's just the way it is.
- However, if you've really tried everything one of your teammates remains unreachable: do the project anyway, and we'll try and see what we can do about it during the defense. Even if the group leader is missing, you still have access to the submission directory.
- You must use the latest version of Go.

Make sure the subject that was originally assigned to your group works perfectly before considering bonuses: If a bonus subject works, but the original one fails the tests, you'll get 0.

Chapter II

Rush02

	Exercise 00
Rush02	
Turn-in directory : *ex*00/	
Files to turn in : `*`	
Allowed packages : `github.com/42tokyo/ft, os`	
Allowed builtin functions : `append, cap, copy, delete, len, make, new`	

II.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:

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

II.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)`

II.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 :

```
$> cat sample.fillit | cat -e
....$
##..$
.#..$
.#..$
$
....$
####$
....$
....$
$
#...$
###.$
....$
....$
$
....$
##..$
.##.$
....$
$> go run *.go sample.fillit | cat -e
DDAA$
CDDA$
CCCA$
BBBB$
$>
```

Another example :

```
$> cat sample.fillit | cat -e
....$
....$
####$
....$
$
....$
...$
..##$
..##$
$> go run *.go sample.fillit | cat -e
error$
$>
```

Last Example :

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

II.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 III

Bonus

	Exercise 01
Bonus	
Turn-in directory : $ex01/$	
Files to turn in : `*`	
Allowed packages : `Everything except external packages`	
Allowed builtin functions : `*`	

Add more functionality to ex00.
For example:

- Fillit map generation functionality.
- Visualization of the solver.
- Able to solve impossible.fillit from the resource.

Other (creative) functionality will be graded too.
For each functionality, 5 points are given. (Max 25points)