



Fillit

Ты можешь почувствовать это?

Резюме: Это история о кусочке `Тетриса`, об одном квадрате и о том, как разработчик заходит в бар...

Глава I.

Предисловие

Алексей Леонидович Пажитнов - российский разработчик видеоигр и компьютерный инженер, который разработал популярную игру «Тетрис», работая в Дородницинском вычислительном центре Академии наук СССР, научно-исследовательском центре, созданном советским правительством.

Пажитнов родился 14 марта 1956 года в Москве. В детстве он был фанатом головоломок и играл в игрушки «пентамино». Создавая тетрис, он черпал вдохновение в этих игрушках.

Пажитнов создал «Тетрис» с помощью **Дмитрия Павловского** и **Вадима Герасимова** в 1984 году. Игра, впервые доступная в Советском Союзе, появилась на Западе в 1986 году. **Пажитнов** также создал менее известное продолжение игры «Тетрис», названное «Welltris», которое имеет тот же принцип, но в трехмерной среде, где игрок видит игровую площадку сверху. «Тетрис» лицензировался и управлялся советской компанией ЭЛОРГ, которая была создана специально для этой цели и рекламировалась под слоганом «Из России с любовью» (дословный перевод: «Из России с удовольствием!»). Поскольку он работал на советское правительство, **Пажитнов** не получал гонорара.

Пажитнов вместе с **Владимиром Похилко** переехал в США в 1991 году, а позже, в 1996 году, вместе с **Хенком Роджерсом** основал компанию «The Tetris Company». Он помогал разрабатывать головоломки в версиях «Yoshi's Cookie» для «Super NES» и разработал игру «Pandora's Box», которая включает в себя более традиционные головоломки.

Он работал в «Microsoft» с октября 1996 года по 2005 год. Там он работал над группами «Microsoft Entertainment Pack» включавших в себя: «The Puzzle Collection», «MSN Mind Aerobics» и «MSN Games». Новая, улучшенная версия «Hexic» от **Пажитнова**, «Hexic HD», была включена в каждый новый пакет «Xbox 360 Premium».

18 августа 2005 года «WildSnake Software» объявила, что **Пажитнов** будет сотрудничать с ними, чтобы выпустить новую линейку головоломок.

Глава II.

Вступление

Fillit - это проект, который позволяет вам обнаружить и/или ознакомиться с повторяющейся проблемой в программировании: поиск оптимального решения среди огромного набора возможностей в приемлемые сроки. В этом конкретном проекте вам нужно будет найти способ собрать данный набор **tetriminos** на минимально возможном квадрате.

Tetriminos - это геометрическая фигура из 4 блоков, о которой вы, вероятно, уже слышали благодаря популярной игре «Тетрис».



Глава III.

Цели

Fillit не занимается перекодированием игры «тетрис», даже если это все еще вариант этой игры. Ваша программа примет файл в качестве параметра, который содержит список **tetriminos**, и упорядочит их так, чтобы создать наименьший возможный квадрат.

Очевидно, ваша главная цель - найти наименьший квадрат за минимальное время, несмотря на экспоненциально растущее количество возможностей каждый раз, когда добавляется кусок.

Вам следует хорошо подумать о том, как вы будете структурировать свои данные и как решить эту проблему, если вы хотите, чтобы ваша программа ответила до следующего тысячелетия.



Глава IV.

Основные инструкции

- Ваш проект должен быть написан на **C** и должен соответствовать **нормам** написания кода.
- Разрешены для использования функции: **exit**, **open**, **close**, **write**, **read**, **malloc** и **free**.
- Все пространство памяти, выделенное кучей, должно быть должным образом освобождено при необходимости. Утечки памяти недопустимы.
- Ваш **Makefile** должен компилировать ваш код без повторных ссылок(повторное связывание).
- Ваш **Makefile** должен содержать следующие правила: **all**, **clean**, **fclean** и **re**.
- Вы должны скомпилировать свой двоичный файл с флагами **-Wall**, **-Wextra** и **-Werror**. Любые другие флаги запрещены, особенно в целях оптимизации.
- Бинарный файл должен называться **fillit** и находиться в корневом каталоге вашего репозитория.
- Вам нужно будет отправить в корень вашего репозитория файл с именем **author**, содержащий ваше имя пользователя, за которым следует символ **\n**.

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

Глава V.

Обязательная часть

[V.1] Вступление в программу

Ваш исполняемый файл должен принимать только один параметр - файл, содержащий список **tetrimino** для сборки. Этот файл имеет очень специфический формат: каждое **tetrimino** должно точно соответствовать квадрату 4 на 4 символа, и все **tetrimino** разделяются новой строкой.

Если количество параметров, отправленных вашему исполняемому файлу, не равно 1, ваша программа должна отображать его использование и завершаться должным образом. Если вы не знаете, что такое использование, выполните команду **cp** без аргументов в своей оболочке. Это даст вам представление. Ваш файл должен содержать от 1 до 26 **tetrimino**.

Описание **tetrimino** должно соответствовать следующим правилам :

- Ровно 4 строки по 4 символа, за каждой следует новая строка (ну... квадрат 4x4).
- **Tetrimino** - это классическая игра «Tetris», состоящая из 4 блоков.
- Каждый символ должен быть либо символом блока ('#'), либо пустым символом ('.').
- Каждый блок **tetrimino** должен касаться хотя бы одного другого блока с любой из его 4 сторон (вверх, вниз, влево и вправо).

Несколько примеров действительных описаний **tetrimino** :

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

Несколько примеров НЕдействительных описаний **tetrimino** :

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

Поскольку каждое **tetrimino** заполняет только 4 из 16 доступных квадратов, можно описать одно и то же **tetrimino** разными способами. Однако в случае с этим проектом повернутое **tetrimino** описывает другое **tetrimino**, отличное от оригинала. Это означает, что на **tetrimino** невозможно вращение, когда вы будете совмещать его с другими.

Таким образом, эти **tetrimino** полностью эквивалентны во всех аспектах :

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

Эти 5 **tetrimino**, в свою очередь, являются пятью отличительными **tetrimino** во всех аспектах:

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

Наконец, вот пример допустимого файла, который должна разрешить ваша программа:

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

... и пример недопустимого файла, который ваша программа должна отклонить по нескольким причинам:

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


[V.2] Самый маленький квадрат

Цель этого проекта - расположить все **tetrimino** друг с другом так, чтобы получился как можно меньший квадрат. Но в некоторых случаях в этом квадрате должны быть дыры, когда одни фигуры не подходят друг другу. Даже если они заключены в квадрат **4x4**, каждое **tetrimino** определяется своими минимальными границами (символом '#'). Оставшиеся **12** пустых символов будут проигнорированы в процессе сборки **tetrimino**.

Tetrimino отсортированы по порядку появления в файле. Среди всех возможных кандидатов на самый маленький квадрат мы принимаем только тот, в котором **tetrimino** находится в самом верхнем левом положении.

Пример :

Принимая во внимание два следующих **tetrimino** (символ '#' будет заменен цифрами для понимания):

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

Наименьший квадрат, который вы можете сделать из этих двух частей, имеет ширину **4** символа, но есть много возможных вариантов, которые вы можете увидеть прямо ниже:

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

Согласно приведенному выше правилу правильным решением будет **a)**.

[V.3] Вывод программы

Ваша программа должна отображать самый маленький собранный квадрат на стандартном выводе. Чтобы идентифицировать каждое **tetrimino** в квадратном решении, вы назначите каждому **tetrimino** заглавную букву, начиная с символа 'A' и увеличиваясь для каждого нового **tetrimino**.

Если файл содержит хотя бы одну ошибку, ваша программа должна отображать ошибку в стандартном выводе с последующим переводом строки и должна завершиться должным образом.

Tetrimino отсортированы по порядку появления в файле. Среди всех возможных кандидатов на самый маленький квадрат мы принимаем только тот, в котором **tetrimino** находится в самом верхнем левом положении.

Пример :

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

Другой пример :

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

Последний пример :

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

[V.3] Автоматическая коррекция

Из-за строгости мулинета вы должны соблюдать тот же протокол возврата, что и в библиотеке **libft**. Все ваши источники и заголовки должны находиться в одной папке. У вас может быть две разные папки: одна для **libft** и одна для **fillit**.

Глава VI.

Сдача и экспертная оценка

Сдавайте свою работу в репозиторий `git` как обычно. Оцениваться будут только файлы, доступные в этом репозитории.

После экспертной оценки ваша работа будет отправлена в `Moulinette` (в дополнение к любой экспертной оценке в зависимости от вашего Cursus). Во время проверки `Moulinette` включает произвольный тайм-аут, который остановит выполнение вашей программы, если поиск решения для данного теста займет слишком много времени. Очевидно, что тогда этот тест будет считаться ложным.