

Задача.

Напишете програма, която отпечатва на стандартния изход следната рекурсивна фигура:

[illegible]

Оценяване:

- Решението се състои от текста на програмата и кратко описание на коректността ѝ;
- Печелят първите 3 правилни и най-кратки решения с точност до еквивалентни лексеми (token-и).

За *феновете* на *CodeGolf-a*: Позволени са всякакви съкращения на програмата, стига тяхната реализация да се поддържа от стандартни компилатори или интерпретатори за съответния език.

Успех!

Примерно решение:

I. Един подход към задачата е да строим фигурата итеративно, започвайки от "X". Ако текущата итерация съдържа редовете r_1, \dots, r_n , то следващата итерация може да се разглежда като съставена от 3 части: горна част, съдържаща редовете $r_1.(n \text{ интервала}).r_1, \dots, r_n.(n \text{ интервала}).r_n$, средна част, съдържаща редовете $(n \text{ интервала}).r_1.(n \text{ интервала}), \dots, (n \text{ интервала}).r_n.(n \text{ интервала})$ и долна част еднаква с горната. Търсената фигура представлява четвъртата итерация от този процес.

Кратки решения имплементиращи тази идея са например:



/автор С.Ф.

```
c,l=["X"]," "
for i in range(4):
    t=[s+l+s for s in c]
    c,l=t+[l+s+l for s in c]+t,l+l+l
for s in c:print(s)
```



/автор Б.С.

```
def f(a)
  s=' '*a.length
  r=a.map{|l|l+s+l}
  r+a.map{|l|s+l+s}+r
end
puts f f f f [?*
```



/ автор А.Н.

```
g a b=a++b++a  
f s=g[g l(s>>" ")|l<-s][g(s>>" ")l|l<-s]  
main=mapM putStrLn$ff$ff$ff["x"]
```

II. Друг подход е да представим n -тата итерация от посочената фигура като начален фрагмент с размери $3^n \times 3^n$ от безкрайна матрица със символа "X", от която последователно сме „изтрили“ всички символи с индекси, различни по четност след целочислено делене на 1, 3, 9, 27 и т.н. В този смисъл дали даден символ ще бъде изтрит на някоя от посочените итерации зависи единствено от индексите му.

Тази идея също може да се реализира лесно на различни програмни езици и парадигми.

Ще разгледаме обаче следното Golfer-ско решение на C (GCC):



/автор С.Ф

```
u(x,y){x|y&&(x+y&1||u(x/3,y/3));}
main(r,c){for(r=-1;r++<80;puts(""))
for(c=-1;c++<80;printf(u(r,c)?" ":"X"));
```

По същество CodeGolf-ът представлява вид състезателно програмиране, при което се търси най-късият възможен код, реализиращ дадена програма. Един от подходите за това е използването на специфични езици (например APL, J, GolfScript, Pyth), поддържащи особен набор от вградени функции с къси имена. Този подход често води до много кратки решения за сметка на нетривиални алгоритми. Друг подход е „изкуствено“ намаляне дължината на кода (например C, Haskell, Python, Ruby), чрез премахване на интервали, кратки имена на променливи и функции, свойства на компилатори и др.

Първото, което прави впечатление в горната програма е липсата на `include` директиви и описание на типове. При компилиране с GCC (например v7.1.1) обаче `printf` и `puts` от `stdio.h` се декларираат имплицитно. Също така се ползва свойството за имплицитно използване на `int` при липса на описание на тип (част от C89 стандарт). Допълнителни символи се печелят и с дефинирането на `g` и `s` като аргументи на `main` вместо декларация.

Да разгледаме функцията, която илюстрира основната идея на решението:

$$u(x, y) \{x | y \& \& (x+y \& 1 | | u(x/3, y/3)) ; \}$$

Тя проверява дали елемент (x, y) бива изтрит, започвайки от първата итерация. За целта трябва x и y да са различни по четност или елементът да се изтрие в контекста на следващата итерация с индекси $(x/3, y/3)$, като елемент $(0, 0)$ е винаги неизтрит. Частта $x|y$ проверява дали x и y не са едновременно 0, а $x+y\&1$ проверява дали x и y са различни по четност, еквивалентно на дали сборът им е нечетен.

Най-съществената част от тази функция е липсата на return statement. На ниво assembler такова нещо фактически няма - подпрограмата просто записва резултата си в регистър еах.

Функцията се изпълнява на следния принцип (псевдокод на assembler):

```
u:
    ; сметни x|y и запиши в еах
    mov     еах, x
    or      еах, y
    test    еах, еах
    je      .L1          ; скок при ZF, т.е еах е 0
    ; сметни x+y&1 и запиши в еах
    mov     еах, x
    add     еах, y
    and     еах, 1
    test    еах, еах
    jne     .L2          ; скок при не ZF, т.е еах не е 0 (в случая е 1)
    ; подготви рекурсивното извикване
    ...
    call    u            ; сега еах е резултатът от рекурсията - 0 или 1
    test    еах, еах
    je      .L1          ; скок при ZF, т.е еах е 0
.L2:
    (mov     еах, 1 или nop при липса на return statement)
    jmp     .END
.L1:
    (mov     еах, 0 или nop при липса на return statement)
.END:
    leave
    ret
```

При внимателен анализ се забелязва, че когато програмата стигне под .L2 или .L1, mov операцията е излишна, тъй като регистър еах вече съдържа търсената стойност.

Същата програма може да се реализира по-кратко на специализиран език за CodeGolf. Например:

Pyth

/автор С.Ф.

```
M?<+GH1"X"?n%G2%H2" "g/G3/H3=T0W<T81=k""V81=k+kgTN)=ThTk
```

Еквивалентността на двете програми може да се види непосредствено след прочитане на втората като псевдокод:

define g(G,H):	; M
If (G+H<1) then return "X"	; ?<+GH1"X"
Elseif (G mod 2)≠(H mod 2) then return " "	; ?n%G2%H2" "
Else return g(G div 3, H div 3)	; g/G3/H3
Set T to 0	; =T0
While T < 81 do	; W<T81
Set k to ""	; =k""
For N from 0 to 80 do	; V81
Set k to k.g(T, N)	; =k+kgTN
End For	;)
Set T to T+1	; =ThT
Print k	; k

Други кратки решения:



/автор С.Ф. (Golfer-ско решение без рекурсия)

```
main(r,c,x,y){for(r=-1;r++<80;puts(""))for(c=-1;c++<80;
printf(x|y?" ":"X"))for(x=r,y=c;x|y&&x+y&1^1;x/=3,y/=3);}
```



/автор С.Ф. (вариант на втория подход)

```
s r c|r+c<1='X'| (r+c)`mod`2==1=' '|True=s`div`3)(c`div`3)
main=mapM putStrLn[[s r c|c<-[0..80]]|r<-[0..80]]
```



/автор З.М. (вариант на втория подход)

```
#include <stdio.h>

#define to_base3_9(x) ( (x) / 3 * 4 + (x) % 3 )
#define to_base3_81(x) ( to_base3_9((x) / 9) * 16 + to_base3_9((x) % 9) )

int main() {
    for (unsigned i = 0; i < 6561; i++) {
        putchar((to_base3_81(i / 81) ^ to_base3_81(i % 81)) & 0x55 ? ' ' : 'X');

        if (i % 81 == 80)
            putchar('\n');
    }
}
```



/автор Б.С. (вариант на първия подход)

```
c="x"
s=" "
z[a_]:={{a,s,a},{s,a,s},{a,s,a}}
Print@*StringJoin/@Nest[ArrayFlatten[#{c->z[c],s->z[s]}]&,c,4]
```



/автор П.П. (друг подход за рекурсивно строене на фигурата)

```
#include<bits/stdc++.h>
using namespace std;
vector<string> v(81);
void f(int n, int r, int c){
    if (n == 1) v[r]+=c;
    else for (int i = 0; i < 9; i++) f(n-1, r+i/3*pow(3,n-2), i%2?' ':c);
}
int main() {
    f(5, 0, '*');
    for (string s: v) cout << s << endl;
}
```