ФУНКЦИОНАЛНО ПРОГРАМИРАНЕ

Магдалина Тодорова
magda@fmi.uni-sofia.bg
todorova_magda@hotmail.com
кабинет 517, ФМИ

ВЛОЖЕНИ ДЕФИНИЦИИ И БЛОКОВА СТРУКТУРА.

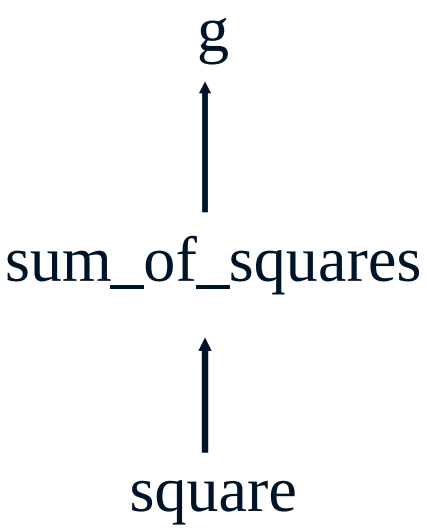
ПРОЦЕДУРИ И ПРОЦЕСИТЕ, КОИТО ТЕ ГЕНЕРИРАТ

1. Вложени дефиниции и блокова структура

Тъй като тялото на функция (процедура) е редица от изрази, възможно е в редицата да са дефинирани (вложени) други процедури. За процедурите, в дефинициите на които са вложени дефинициите на други процедури, се казва, че имат блокова структура.

***Пример за програма с блокова структура*:**

```
(define (g a)
    (define (square x)
        (* x x))
    (define (sum_of_squares x y)
        (+ (square x) (square y)))
    (sum_of_squares (+ a 3) (* a 2)))
```



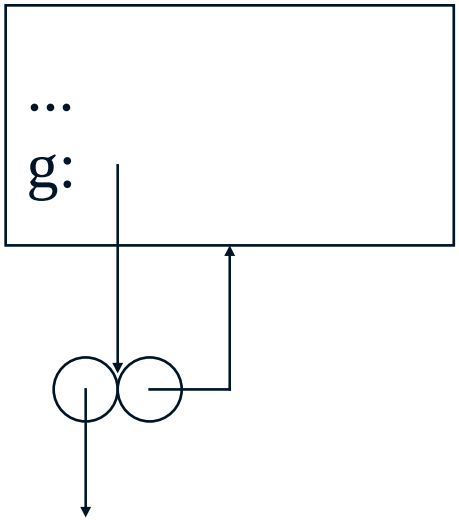
1. Вложени дефиниции и блокова структура

***Пример за програма с блокова структура*:**

```
(define (g a)
    (define (square x)
        (* x x))
    (define (sum_of_squares x y)
        (+ (square x) (square y)))
    (sum_of_squares (+ a 3) (* a 2)))
```

блок на процедурата g

1. Вложени дефиниции и блокова структура

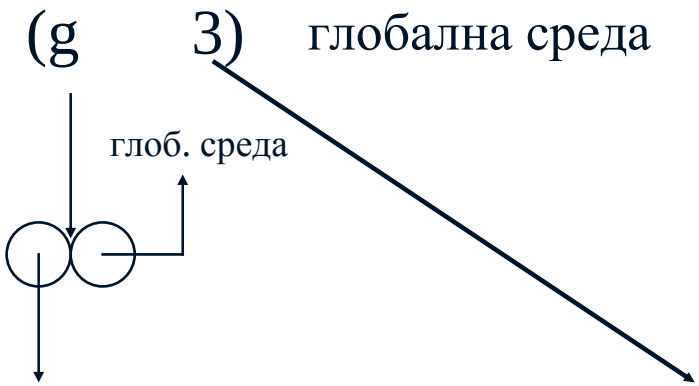


глобална среда

параметри: a

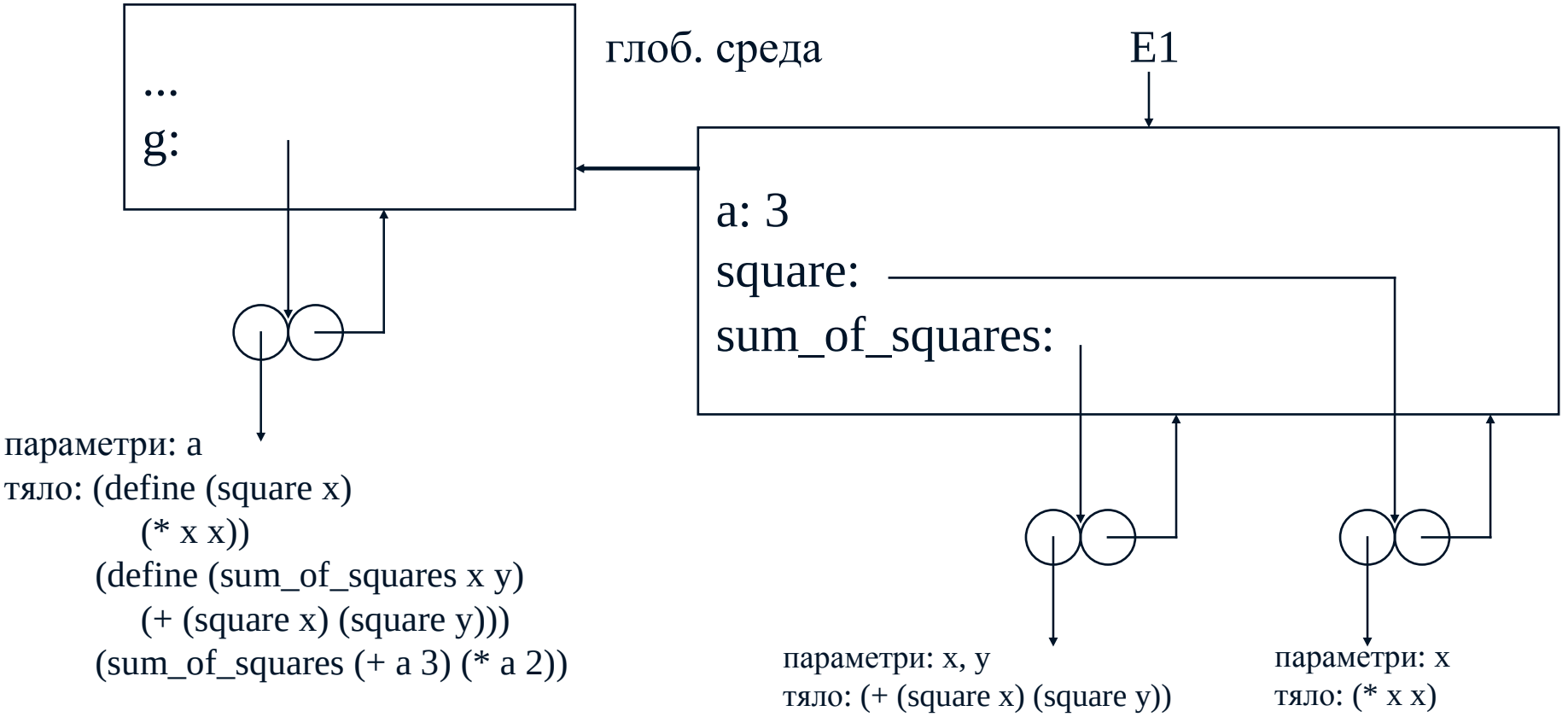
тяло: (define (square x)
(* x x))
(define (sum_of_squares x y)
(+ (square x) (square y)))
(sum_of_squares (+ a 3) (* a 2))

1. Вложени дефиниции и блокова структура

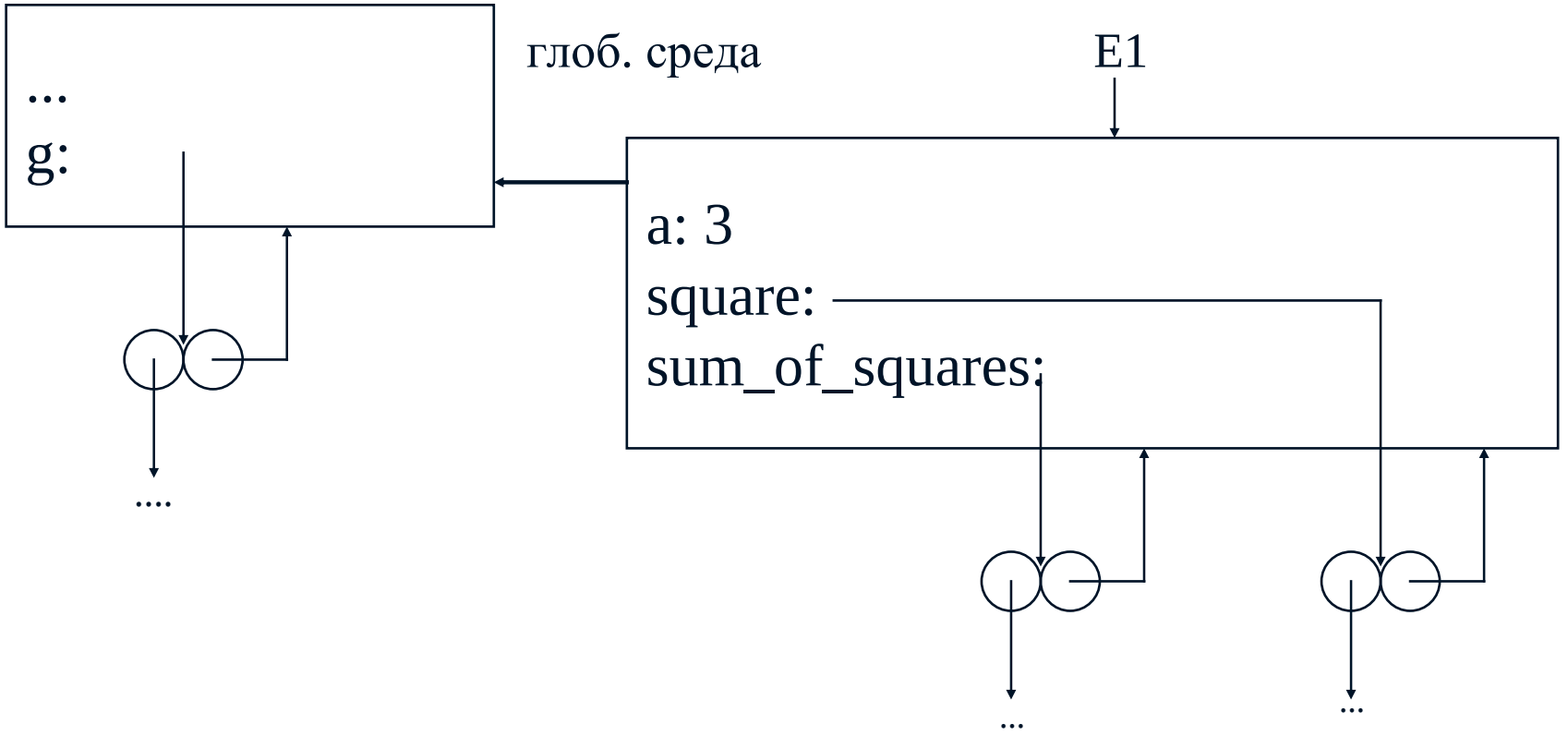



```
тяло: (define (square x)
          (* x x))
      (define (sum_of_squares x y)
         (+ (square x) (square y)))
      (sum_of_squares (+ a 3) (* a 2))
```

1. Вложени дефиниции и блокова структура

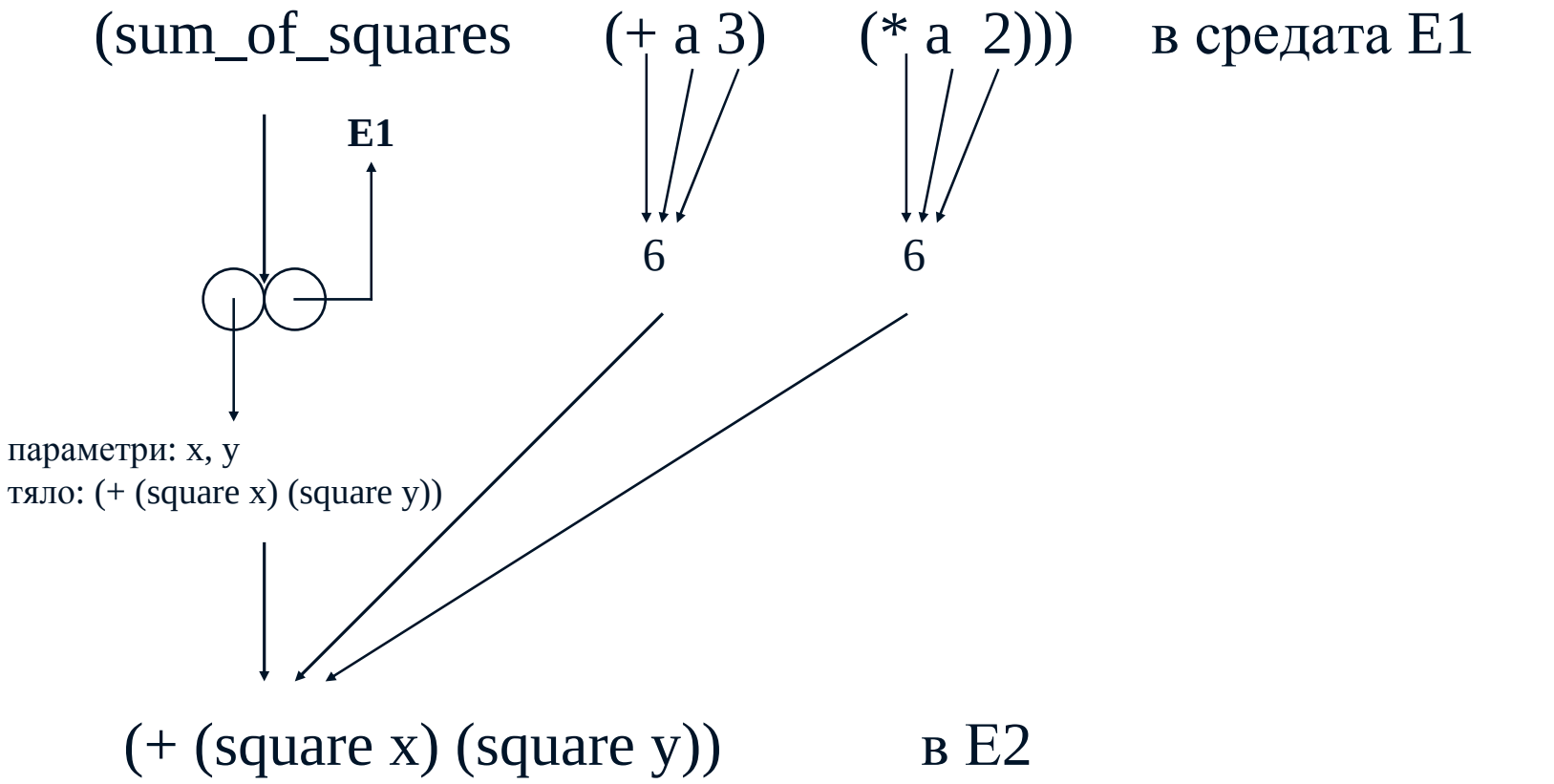


1. Вложени дефиниции и блокова структура

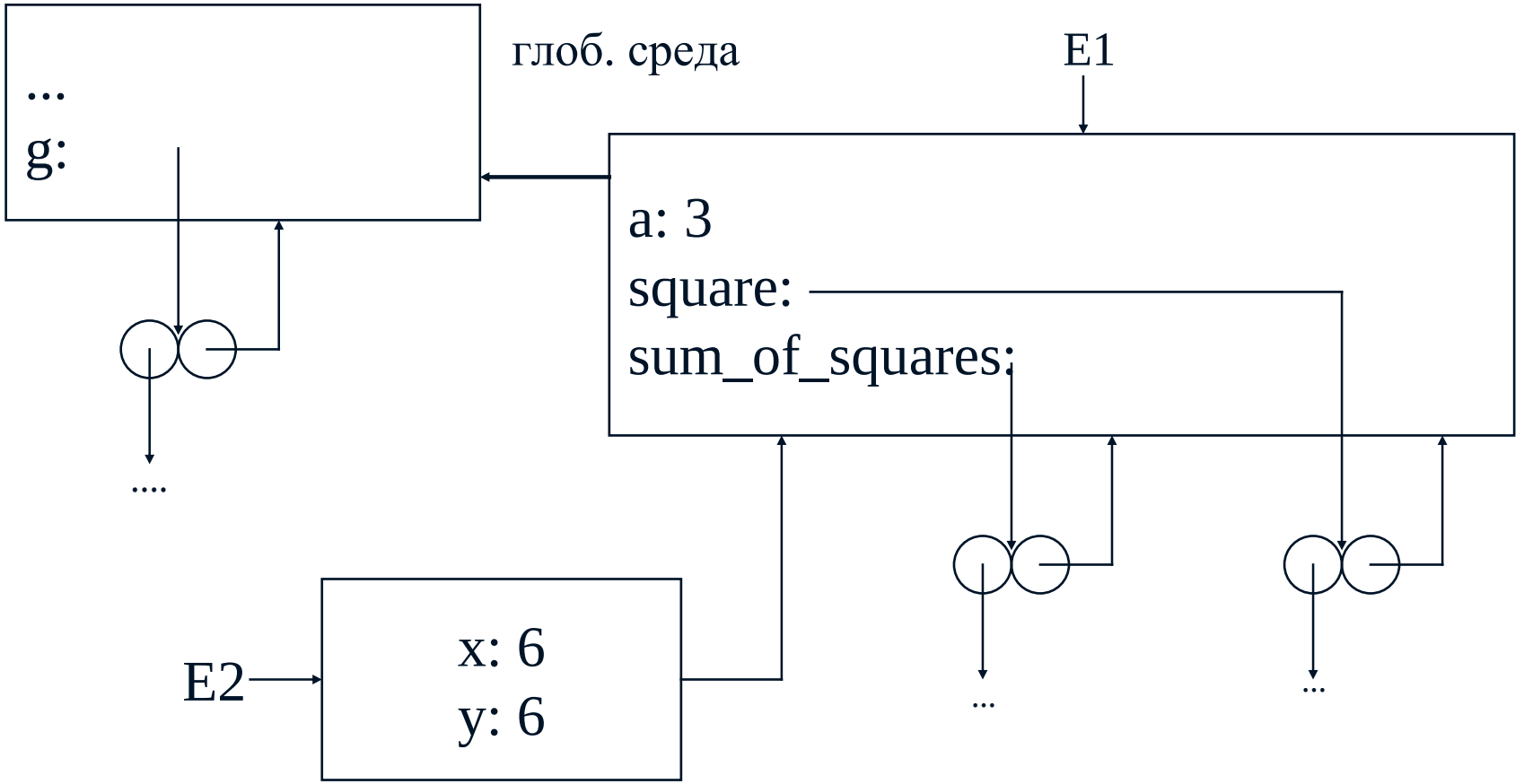


(sum_of_squares (+ a 3) (* a 2)) в средата E1

1. Вложени дефиниции и блокова структура



1. Вложени дефиниции и блокова структура



Следствия:

1. Областта на формалните параметри на процедура, която има блокова структура, е блокът на процедурата.
2. Областта на вложените дефиниции в процедура, която има блокова структура, е блокът на процедурата.
3. Редът на дефинирането на вложените дефиниции в процедура, която има блокова структура, не е от значение.
4. Възможно е в дефиниция на процедура с блокова структура да се вложи процедура със същото име (записват се в различни среди).
5. Възможно е формален параметър на вложена процедура да има име, съвпадащо с име на формален параметър на външната процедура.

2. Процедури и процесите, които те генерират

Процес

Процесът е абстрактно понятие, описващо развитието във времето на пресмятанията върху определени входни данни до получаване на крайния резултат.

2. Процедури и процесите, които те генерират

Линейна рекурсия и итерация

Задача. Да се състави програма за пресмятане на **n!**.

***Първи начин*:** Чрез дефиницията на **n!**, според която:

n! = n.(n-1)!, n > 1,

1! = 1.

```
(define (fact n)
    (if (= n 1) 1
        (* n (fact (- n 1)))))
```

2. Процедури и процесите, които те генерират

Проследяване на процеса

(fact 4)
(* 4 (fact 3))
(* 4 (* 3 (fact 2)))
(* 4 (* 3 (* 2 (fact 1))))
(* 4 (* 3 (* 2 1)))
(* 4 (* 3 2))
(* 4 6)
24

разгъване на оценяването

сгъване на оценяването

Такъв процес се нарича **рекурсивен**. Тъй като дължината на веригата от отложени умножения е от порядъка на n, процесът се нарича **линейно рекурсивен**.

Втори начин: Използва се дефиницията на **n!**, съгласно която **n! = 1.2.3. ... .n.**

Според тази дефиниция n! може да се намери по следния начин:

```
f = 1;
for (i = 1; i <= n; i++)
    f = f * i;
```

```
f = 1;
for(i = 1; i <= n; i++)
    f = f * i;
```

Запис на Scheme

```
(define (fact n)
    (define (fact-iter f i)
        (if (> i n) f
            (fact-iter (* f i)  (+ i 1))))
    (fact-iter 1 1))
```

2. Процедури и процесите, които те генерират

Проследяване на процеса

(fact 4)

(fact-iter 1 1)

(fact-iter 1 2)

(fact-iter 2 3)

(fact-iter 6 4)

(fact-iter 24 5)

24

Такъв процес се нарича **итеративен**. И тъй като времето за оценяване е от порядъка на n, процесът се нарича още **линейно итеративен**.

2. Процедури и процесите, които те генерират

Сравнение на двете решения

Прилики:

- реализират пресмятането на една и съща математическа функция;
- броят на стъпките е пропорционален на **n**, т.е. и двата процеса са линейни;
- извършва се една и съща редица от умножения (1.2, 1.2.3, 1.2.3.4, ...) и съответно се получават еднакви междинни резултати.

2. Процедури и процесите, които те генерират

Разлики:

- при линейно рекурсивния процес има фаза на разгъване (отложени операции) и след това - фаза на сгъване (изпълнение на отложените операции). При линейно итеративния процес няма разгъване, няма свиване; броят на участващите операции е постоянен.

- при изпълнението на линейно рекурсивния процес интерпретаторът трябва да запазва формираната верига от умножения, за да може по-късно да ги изпълни. При изпълнението на линейно итеративния процес, текущите стойности на променливите **f** и **i** дават пълна информация за текущото състояние на изчислителния процес.

2. Процедури и процесите, които те генерират

В програмирането рекурсията се определя като:

- ***акумулираща рекурсия;***
- ***опашкова рекурсия***;
- ***пряка рекурсия;***
- ***косвена рекурсия;***
- ***споделена рекурсия;***
- ***нелинейна рекурсия*** и др.

2. Процедури и процесите, които те генерират

При **акумулиращата (линейна) рекурсия** дефиницията включва (поражда) само едно рекурсивно обръщение към **същата** процедура с опростени аргументи.

Пример (акумулираща рекурсия):

```
(define (fact n)
        (if (= n 1) 1
            (* n (fact (- n 1)))))
```

2. Процедури и процесите, които те генерират

При опашковата рекурсия (tail recursion) **общата задача се трансформира до нова**, по-проста, като решението на общата задача съвпада с решението на по-простата, а не се получава от него с помощта на допълнителни (отложени) операции.

Пример (опашкова рекурсия):

```
(define (fact n)
    (define (fact-iter f i)
        (if (> i n) f
            (fact-iter (* f i)  (+ i 1))))
    (fact-iter 1 1))
```

2. Процедури и процесите, които те генерират

При опашковата рекурсия общата задача директно се редуцира до по-проста и няма нужда от обратен ход за получаването на решението на общата задача.

Използването на опашкова рекурсия повишава ефективност на съответните процедури. Това е така, тъй като интерпретаторите на Scheme се реализират така, че при изпълнението на опашково рекурсивните процедури не използват допълнителна памет при изпълнението на обръщенията към процедурите.

Затова се казва, че интерпретаторите на езика Scheme имат опашково рекурсивна семантика.

Дървовидна рекурсия

Задача: Да се дефинира процедура, която намира n-тото число на Фибоначи ($n \geq 0$), т.е.

$$fib(n) = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ fib(n-1) + fib(n-2), & n > 1 \end{cases}$$

2. Процедури и процесите, които те генерират

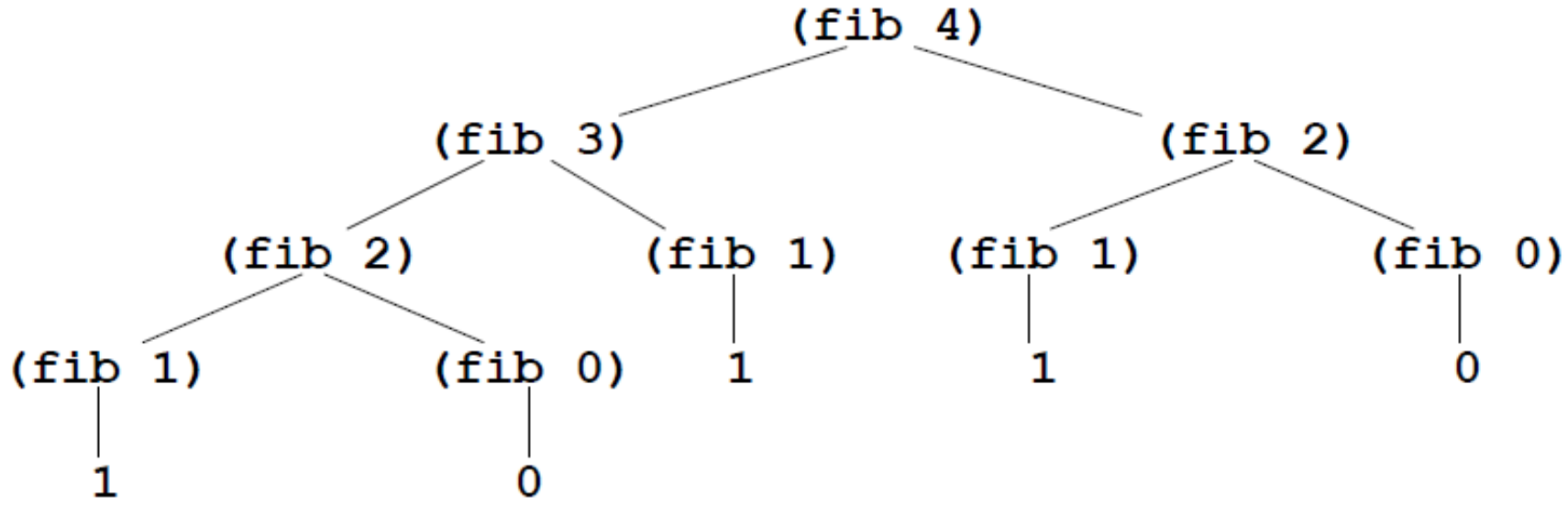
Дървовидна рекурсия

I-ви начин - чрез рекурсивна процедура, която реализира рекурсивен процес.

```
(define (fib n)
  (cond ((= n 0) 0)
        ((= n 1) 1)
        (else (+ (fib (- n 1)) (fib (- n 2))))))
```

Дървовидна рекурсия

Оценяването на обръщението (fib 4) може да се разглежда като процес на обхождане в дълбочина на дървото:



Затова процесът, породен от тази процедура, се нарича **дървовидно рекурсивен**.

2. Процедури и процесите, които те генерират

Анализ:

1) Функцията fib(n) расте експоненциално с нарастването на n. По-точно, fib(n) е най-близкото цяло число до $\Phi^n/\sqrt{5}$, където $\Phi = (1+\sqrt{5})/2 \approx 1.6180$ е положителният корен на уравнението $\Phi^2 = \Phi+1$;

2) времето за намиране на fib(n) е пропорционално на броя на генерираните рекурсивни обръщения (броя на стъпките, за които се извършва пресмятането), който е равен на броя на върховете на дървото и е от порядъка на 2^{n-1};

3) обемът на необходимата памет е пропорционален на дълбочината на дървото, която е O(n).

2. Процедури и процесите, които те генерират

II-ри начин - чрез рекурсивна процедура, която реализира линейно итеративен процес.

Реализира се програмния фрагмент:

```
(a, b) = (1, 0)
for(i = n; i >= 1; i--)
    (a, b) = (a+b, a);
```

```
(define (fib n)
    (define (fib-iter a b i)
        (if (= i 0) b
            (fib-iter (+ a b) a (- i 1))))
    (fib-iter 1 0 n))
```