ФУНКЦИОНАЛНО ПРОГРАМИРАНЕ

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

Процедури от по-висок ред

(Процедурите като параметри)

Процедурите като параметри – дефиниция и примери

Дефиниция.

Процедури, някой формални параметри на които са процедури, се наричат **процедури от по-висок ред**.

Ще покажем дефинирането им чрез дефиниране на:

$$\sum_{\substack{i=a,\\ next}}^{b} term(i) \;,\; \prod_{\substack{i=a,\\ next}}^{b} term(i)$$

и тяхно обобщение, акумулиращо елементи.

Процедурите като параметри

Пример 1. Пресмятане на сумата на целите числа от *a* до *b* със стъпка 1 (*a*, *b* са дадени цели числа).

***Решение*:**

$$\sum_{\substack{k=a,\\ 1+}}^{b} k = a + (a+1) + (a+2) + \ldots + b = a + \sum_{\substack{k=a+1,\\ 1+}}^{b} k$$

$$Sum(a,b) = \begin{cases} 0, & \text{a} > \text{b} \\ a + Sum(a+1, b), & \text{a} \le \text{b} \end{cases}$$

Процедурите като параметри

Пример 1. Пресмятане на сумата на целите числа от *a* до *b* със стъпка 1 (*a*, *b* са дадени цели числа).

***Решение*:**

```
(define (id x) x)
(define (1+ x) (+ x 1))

(define (sum-int a b)
  (if (> a b)
      0
      (+ (id a) (sum-int (1+ a) b))))
```

Процедурите като параметри

Пример 2. Пресмятане на сумата от кубовете на целите числа от *a* до *b* със стъпка 1 (*a* и *b* са дадени цели числа).

***Решение*:**

$$\sum_{\substack{k=a,\\ 1+}}^{b} k^{3} = a^3 + (a+1)^3 + (a+2)^3 + \ldots + b^3 = a + \sum_{\substack{k=a+1,\\ 1+}}^{b} k^{3}$$

$$Sum(a,b) = \begin{cases} 0, & \mathrm{a} > \mathrm{b} \\ a^{3} + Sum(a+1, b), & \mathrm{a} \le \mathrm{b} \end{cases}$$

Пример 2. Пресмятане на сумата от кубовете на целите числа от *a* до *b* със стъпка 1 (*a* и *b* са дадени цели числа).

***Решение*:**

```
(define (cube x) (* x x x))
(define (1+ x) (+ x 1))

  (define (sum-cub a b)
      (if (> a b)
          0
          (+ (cube a) (sum-cub (1+ a) b))))
```

Процедурите като параметри

Пример 3. Пресмятане на част от сумата по-долу, която според известната формула на Лайбниц клони към $\pi/8$:

$$\frac{\pi}{8} = \frac{1}{1.3} + \frac{1}{5.7} + \frac{1}{9.11} + \ldots + \frac{1}{a(a+2)} + \frac{1}{(a+4)(a+6)} + \ldots$$

a е цяло положително число от вида *4k+1*.

***Решение*:**

$$\frac{1}{a(a+2)} + \frac{1}{(a+4)(a+6)} + \ldots + \frac{1}{b(b+2)}$$

Пример 3.

$$\frac{1}{a(a+2)}+\frac{1}{(a+4)(a+6)}+\ldots+\frac{1}{b(b+2)}$$

$$Sum(a,b)=\begin{cases} 0, & a > b \\ \dfrac{1}{a(a+2)}+Sum(a+4,b), & a \le b \end{cases}$$

Пример 3.

```
(define (first x) (/ 1 (* x (+ x 2))))
(define (4+ x) (+ x 4))

(define (sum-pi a b)
   (if (> a b)
       0
       (+ (first a) (sum-pi (4+  a) b))))
```

Пресмятане на приближения на числото π.

```
> (* 8.0 (sum-pi 1 40000))
3.1415426535898243
> (* 8.0 (sum-pi 1 60000))
3.1415593202564693
> (* 8.0 (sum-pi 1 100000))
3.141572653589795
> (* 8.0 (sum-pi 1 150000))
3.1415793202564606
```

Процедурите като параметри

Коментар. Трите процедури от горните примери използват един и същ образец (шаблон) за дефиниране, имащ вида:

```
(define (<name> a b)
  (if (> a b)
      0
      (+ (<term> a)
         (<name> (<next> a) b)) ))
```

където:

- **<term>** е име на процедура (задава правилото за пресмятане на поредния член на сумата);

- **<next>** също е име на процедура (задава правилото за получаване на следващата стойност на аргумента).

Процедурите като параметри

Този образец отразява идеята за намиране на сума от вида:

$$\sum_{\substack{i=a,\\ next}}^{b} f(i) = f(a) + f((\text{next a})) + \text{f}((\text{next}(\text{next a}))) + \ldots + f(b)$$

В тази формула *Σ* съответства на **<name>**, *f* – **<term>**, *next* – на <next>.

Дефиниция на функцията от по-висок ред sum

```
(define (sum term a next b)
   (if (> a b)
       0
       (+ (term a)
          (sum term (next a) next b))))
```

функция

Процедурите като параметри

Прилагане на sum :

а) Дефиниране на sum-int чрез sum

```
(define (sum-int a b)
    (sum id a 1+ b))
```

където

```
(define (1+ x) (+ x 1))
(define (id x) x)
```

б) *Дефиниране на sum-cube чрез sum*

```
(define (sum-cub a b)
    (sum cube a 1+ b))
```

където

```
(define (1+ x) (+ x 1))
(define (cube x) (* x x x))
```

Процедурите като параметри

в) *Дефиниране на sum-pi чрез sum (чрез блокова структура)*

```
(define (sum-pi a b)
    (define (pi-term x) (/ 1 (* x (+ x 2))))
    (define (4+ x) (+ x 4))
    (sum pi-term a 4+ b))
```

Процедурите като параметри

***По-сложен пример за използване на процедурата sum*:**

Дадени са: f, a, b и h. Да се намери

$$\int_a^b f(x)dx$$

За целта да се изполва формулата:

$$\int_a^b f(x)dx \approx h.\{f(a+h/2) + f(a+h+h/2) + f(a+2h+h/2) + \ldots +f(b)\}$$

$$= h.\sum_{i=a+h/2}^{b} f(i)$$

***По-сложен пример за използване на процедурата sum*:**

```
(define (integral f a b h)
    (define (next x) (+ x h))
    (* (sum f (+ a (/ h 2)) next b) h))
```

Пример за използване на тази процедура:

```
> (integral cube 0 1 0.001)
0.249999875000001
```

Точната стойност на този интеграл е 0.25.

Процедурите като параметри

Забележка. Дефинираната процедура *sum* генерира линейно рекурсивен процес. За да дефинираме аналогична по резултат процедура, която генерира линеен итеративен процес, преминаваме през аналогични стъпки. Получаваме:

```
(define (sum term a next b)
   (define (sum_iter i s)
      (if (> i b) s
         (sum_iter (next i) (+ s (term i)))))
   (sum_iter a 0))
```

Процедурите като параметри

Да се дефинира функция от по-висок ред product, аналогична на функцията sum, която намира произведението

$$\prod_{a}^{b} f(n) \qquad = f(a). \ldots .f(b).$$

```
(define (product term a next b)
   (if (> a b) 1
        (* (term a)
            (product term (next a) next b))))
```

Процедурите като параметри

Задача: Като се ползва процедурата от по-висок ред **product**, да се дефинира функция, която пресмята *n!* (*n* е дадено цяло неотрицателно число).

Решение: $$n! = \prod_{i=1;1+}^{n} i$$

```
(define (fact n)
    (define (id x) x)
    (define (1+ x) (+ x 1))
    (product id 1 1+ n) )
```

Процедурите като параметри

Обобщение:

Да се дефинира процедура от по-висок ред, която реализира натрупване на редица от стойности на дадена функция, като за целта се използва акумулиране чрез зададена операция **op**. Известна е и началната стойност на операцията.

Процедурите като параметри

```
(define (accumulate op null-value term a next b)
  (if (> a b) null-value
      (op (term a)
          (accumulate op null-value term (next a) next b))))
```

- **op** е функцията, определяща правилото, по което текущата стойност на **term** се комбинира с резултата от натрупването на предишните стойности;
- **null-value** определя началната стойност на натрупването.

***Примери*:**

А) Дефиниция на sum чрез accumulate

```
(define (sum term a next b)
    (accumulate + 0 term a next b))
```

Б) Дефиниция на product чрез accumulate

```
(define (product term a next b)
    (accumulate * 1 term a next b))
```

Процедурите като параметри

Сума на членовете на редица (итеративен вариант)

```
(define (sum term a next b)
  (define (iter i s)
    (if (> i b) s
      (iter (next i) (+ s (term i)))))
 (iter a 0))
```

Произведение на членовете на редица (итеративен вариант)

```
(define (product term a next b)
  (define (iter i s)
    (if (> i b) s
      (iter (next i) (* s (term i)))))
 (iter a 1))
```

Процедурите като параметри

accumulate за членовете на редица (итеративен вариант)

```
(define (accumulate op null_value term a next b)
   (define (iter i s)
      (if (> i b) s
         (iter (next i) (op s (term i)))))
  (iter a null_value))
```

Процедурите като параметри

Да се дефинира процедура от по-висок ред, която обобщава действието на процедурата **accumulate**, като използва в ролята на допълнителен параметър още една процедура – филтър на членовете, които да бъдат акумулирани (натрупани).

Т.е. търсената процедура трябва да акумулира само тези стойности на **term**, които удовлетворяват филтриращата функция.

Процедурите като параметри

Рекурсивен вариант

```
(define (filter-accum pred? op null-value term a next b)
  (cond ((> a b) null-value)
        ((pred? (term a))
         (op (term a) (filter-accum pred? op null-value term
                                    (next a) next b)))
        (else (filter-accum pred? op null-value term (next a) next b))))
```

Итеративен вариант

```
(define (filter-accum pred? op null-value term a next b)
  (define (iter i s)
    (cond ((> i b) s)
          ((pred? (term i)) (iter (next i) (op s (term i))))
          (else (iter (next i) s))))
  (iter a null-value))
```

Тест 1

```
(define (f1 x)
  (define (f1-help a)
    (if (= a 0) 0 (- a 1)))
  (f1-help (f2 (- x 1))))

(define (f2 x)
  (define (f2-help a)
    (if (= a 2) 1 (+ a 1)))
  (if (= x 0) 0
      (f2-help (f1 (- x 1)))))
```

Като се използва моделът на средите да се оценят:

а) (f2 6)

б) (f1 5)