


Метод параметра нагромадження

Лекція 7



Рекурсія – досить ресурсоємкий процес.
Треба писати не лише правильні
програми, а й ефективні.
Як писати рекурсивні функції, щоб
обчислювальні ресурси
використовувалися мінімально.



Способи підвищення ефективності програм

1. Введення допоміжної функції
2. Метод параметра нагромадження (акумулятора)



Дослідимо математичну функцію
факторіал.

Функційне визначення. Варіант 1.

$\text{Factorial1}(0)=1$ //термінальна гілка

$\text{Factorial1}(N)=\text{Factorial1}(N-1) \cdot N$

Хід обчислень (10 кроків)

$N=4$

1. $\text{Factorial1}(4)$
2. $\text{Factorial1}(3) \cdot 4$
3. $\text{Factorial1}(2) \cdot 3 \cdot 4$
4. $\text{Factorial1}(1) \cdot 2 \cdot 3 \cdot 4$
5. $\text{Factorial1}(0) \cdot 1 \cdot 2 \cdot 3 \cdot 4$
6. $1 \cdot 1 \cdot 2 \cdot 3 \cdot 4$
7. $1 \cdot 2 \cdot 3 \cdot 4$
8. $2 \cdot 3 \cdot 4$
9. $6 \cdot 4$
10. 24

Функційне визначення. Варіант 2.

$\text{Factorial2}(0,a)=a$ //термінальна гілка

$\text{Factorial2}(N,a)=\text{Factorial2}(N-1,a \cdot N)$

a – акумулююча змінна, яка містить
результат.

$\text{FactorialA}(N)= \text{Factorial2}(N,1)$

Хід обчислень (6 кроків)

$N=4$, FactorialA(4)

1. Factorial2(4,1)

2. Factorial2(3,4)

3. Factorial2(2,12)

4. Factorial2(1,24)

5. Factorial2(0,24)

6. 24




Ми розглянули спеціальний тип рекурсії - хвостова рекурсія.

Рекурсія, в якій рекурсивний виклик функції є останньою після всіх обчислень її операцією, наз. хвостовою.


Інтерпретатори і компілятори функціональних мов програмування виконують хвостову рекурсію в обмеженому обсязі пам'яті за допомогою ітерацій. Пам'ять витрачається лише на збереження адрес повернення значень функції.

Принципи побудови функцій з папаметром нагромадження

1. Вводиться нова функція з додатковим аргументом – акумулятором для нагромадження результатів обчислень.
2. Початкове значення акумулятора задається у рівності, яка пов'язує вихідну функцію з новою.
3. Ті рівності вихідної функції, які відповідають виходу з рекурсії замінюють поверненням акумулятора.
4. Ті рівності вихідної функції, які відповідають рекурсивному виклику, виглядають як звернення до нової функції, в якому акумулятор отримує те значення, яке повертається вихідною функцією.



Чи будь-яку функцію можна перетворити
для обчислень з параметром
нагромадження?



Метод акумулятора використовується не лише в декларативному програмуванні.

Приклад

Визначити функцію, яка обертає список на верхньому рівні.

`(func '(e m e h c s)) – (s c h e m e)`

`(func '(e m e h c s (s p i l)) –
((s p i l) s c h e m e))`

Допоміжна функція

для об'єднання двох списків


(comb '(1 3 5) '(2 4 6 8)) - (1 3 5 2 4 6 8)

Ідея: беремо по черзі елемент першого списку і записуємо в другий список

```
(define comb (lambda (x y)
```

```
(cond ((eq? x ())) y)
```

```
(#t (cons (car x) (comb (cdr x) y))))))
```



```
(define rever (lambda (x) (cond ((eq? x ()) ())  
  (#t (comb (rever (cdr x)) (cons (car x) ()) )))  
)))
```

Метод акумулятора

```
(define rever_a (lambda (x a) (cond ((eq? x  
  ()) a) (#t (rever_a (cdr x) (cons (car x)  
  a))))))  
  
(define rever_end (lambda(x) (rever_a x () )  
))
```




Витрати на “жадібну” функцію

Аналізуємо скільки разів
використовується конструктор в
функціях

1. rever
2. rever_a

Задано список довжини N

1. `rever` викликає себе N разів. Отже, і `(cons (car x) ())` викликається N разів. `rever` викликає функцію `comb` N разів, для якої перший аргумент має відповідно довжину 0, 1, ..., N-1 елементів. Для кожного з цих викликів відповідну кількість разів використовується конструктор.