ფუნქციების განსაზღვრა დამგროვებელი პარამეტრებით

ენები, რომლებიც ფუნქციონალურ პარადიგმას მიეკუთვნებიან, ბუნებრივია ფლობენ მრავალფეროვან საშუალებებს ფუნქციების აღსაწერად. გამონაკლისი არც ენა MicroSoft Haskell-ია და მასში განსაზღვრულია ფუნქციების წარმოდგენის რამდენიმე საშუალება.

რეკურსიული ფუნქციების შესრულებისას შეიძლება დადგეს მეხსიერების ხარჯვის სერიოზული პრობლემა. განვიხილოთ reverse - სიის შექცევის ფუნქცია Prelude ბიბლიოთეკიდან:

```
reverse :: [a] → [a]
reverse [] = []
reverse (x : xs) = reverse xs ++ [x]
```

++ ოპერაციის ხანგრძლივობა წრფივად არის დამოკიდებული პირველი არგუმენტის სიგრძეზე. n სიგრძის სიისთვის reverse xs მოთხოვნილი ბიჯების რაოდენობაა 1-დან (n+1)-მდე რიცხვების ჯამი:

$$(n+1)(n+2)/2=(n2+ 3n + 2)/2$$

ათი ათასი ელემენტის შემცველი სიის შექცევა მოითხოვს დაახლოებით, რედუქციის 50 მლნ. ბიჯს.

დამატების ოპერაციის (++) გამოსარიცხად განვიხილოთ ე.წ. დამატებითი პარამეტრის შემოტანის მექანიზმი. ხშირად ასეთ პარამეტრს დამგვროვებელ პარამეტრს, იგივე აკუმულატორს უწოდებენ.

განვიხილოთ ფაქტორიალის გამოთვლის ფუნქცია დამატებითი პარამეტრის გამოყენებით:

```
Factorial_A N = F N 1
F 0A = A--მეორე არგუმენტი–აკუმულატორი
F N A = F (N –1) (N * A)
```

- ✓ აკუმულატორი შეიცავს შედეგს, რომელიც ბრუნდება რეკურსიის დამთავრებისას.
- ✓ თვითონ რეკურსიას ამ დროს აქვს „კუდური“ რეკურსიის სახე
- ✓ მეხსიერება გამოიყენება მხოლოდ ფუნქციის მნიშვნელობების დასაბრუნებელი მისამართების შენახვისათვის.

კუდური რეკურსია წარმოადგენს რეკურსიის სპეციალურ სახეს, რომლის დროსაც მხოლოდ ერთხელ ხდება რეკურსიული ფუნქციის გამოძახება და ეს გამოძახებაც სრულდება ყველა გამოთვლის შემდეგ.

კუდური რეკურსიის რეალიზაცია შეიძლება შესრულდეს იტერაციის პროცესის საშუალებით. პრაქტიკაში ეს ნიშნავს, რომ ფუნქციონალური ენის „კარგ“ ტრანსლიატორს უნდა „შეეძლოს“ აღმოაჩინოს კუდური რეკურსია და მოახდინოს მისი რეალიზება ციკლის სახით. თავის მხრივ, პარამეტრის დაგროვების მეთოდს ყოველთვის არ მოვყავართ კუდურ რეკურსიამდე, თუმცა იგი ცალსახად გვეხმარება საერთო მეხსიერების მოცულობის შემცირებაში.

განვიხილოთ reverse - სიის შექცევის ფუნქცია დამატებითი პარამეტრის გამოყენებით:

```
reverse' :: [a] → [a] → [a]
reverse' [] ys=ys
reverse' (x : xs) ys=reverse' xs (x : ys)

reverse :: [a] → [a]
reverse xs=reverse' xs []
```

მაგალითი: *reverse* [1, 2, 3]
={*reverse*-ისგამოყენება}
reverse' [1, 2, 3] []
={*reverse'*-ისგამოყენება}
reverse' [2, 3] (1 : [])
={*reverse'*-ისგამოყენება}
reverse' [3] (2 : (1 : []))
={*reverse'*-ისგამოყენება}
reverse' [] (3 : (2 : (1 : [])))
={*reverse'*-ისგამოყენება}
3 : (2 : (1 : []))

10 000 ელემენტიანი სიის შექცევა ახლა დაიკავებს დაახლოებით 10000 ბიჯს, ნაცვლად 50 მლნ. ბიჯისა.

განვიხილოთ შემდეგი მაგალითი, სადაც საჭიროა ორი დამატებითი პარამეტრი. ესაა სიის ელემენტების საშუალო არითმეტიკულის გამოთვლის ფუნქცია:

```
sasharit l = sasharit'l 0 0
sasharit'[] s n =s/n
sasharit'(x:xs) s n = sasharit'xs (x+s) (n+1)
```

აქ გამოიყენება დამხმარე ფუნქცია sasharit', რომელსაც აქვს დამატებით აკუმულატორად ორი ცვლადი და მისი გამოძახება ხდება ამ ცვლადების საწყისი მნიშვნელობებით: 0 და 0. პირველი ცვლადი აგროვებს სიის ელემენტების ჯამს, ხოლო მეორე ცვლადი - წევრების რაოდენობას. ამ ფუნქციის შედეგია სიის წევრთა ჯამისა და რაოდენობის განაყოფი.

შეიძლება დავადგინოთ დამგროვებელი პარამეტრებით განსაზღვრებების აგების პრინციპები:

1. შემოდის ახალი ფუნქცია დამატებითი არგუმენტით (არგუმენტებით), რომელშიც გროვდება გამოთვლების შედეგები.
2. აკუმულატორი არგუმენტის საწყისი მნიშვნელობა მოიცემა ტოლობით, რომელიც აკავშირებს ძველ და ახალ ფუნქციებს.
3. საწყისი ფუნქციის ის ტოლობა, რომელიც შეესაბამება რეკურსიიდან გამოსავალს, იცვლება აკუმულატორით დაბრუნების გამოსახულებით.
4. ტოლობა, რომელიც შეესაბამება რეკურსიულ განსაზღვრებას, გამოიხატება როგორც ახალ ფუნქციაზე მიმართვა, რომელშიც აკუმულატორი იღებს იმ მნიშვნელობას, რომელიც ბრუნდება საწყისი ფუნქციით.

ნებისმიერი ფუნქცია შეიძლება გარდავქმნათ აკუმულატორის გამოთვლის ფორმით**?**

ამ შეკითხვის პასუხი არის უარყოფითი. დამგროვებელი პარამეტრიანი ფუნქციის აგების ხერხი არ არის უნივერსალური და იგი არ არის კუდური რეკურსიის აგების გარანტია.

განსაზღვრების აგება დამგროვებელი პარამეტრით არის შემოქმედებითი საქმე. ამ პროცესში აუცილებელია ზოგიერთი ევრისტიკის გამოყენება.