რეკურსია სიებზე

რეკურსია არ შემოიფარგლება ფუნქციებით მთელ რიცხვებზე, იგი შეიძლება ასევე გამოყენებული იქნეს ფუნქციათა განსაზღვრისათვის სიებზე. მაგალითად, *product* საბიბლიოთეკო ფუნქცია, რომელიც წინა პუნქტში ვიხმარეთ, შემდეგი სახით შეიძლება განისაზღვროს:

```
product          ::    Num a ⇒ [a] → a
product []       =     1
product (n : ns) =     n * product ns
```

პირველი განტოლება ამტკიცებს, რომ ცარიელი სიის ნამრავლი ერთიანია, რაც სავსებით ადეკვატურია, რადგან ერთიანი გამრავლების ოპერაციაში უნარჩუნებს გამოსახულებას იგივეობას. მეორე განტოლება კი გვეუბნება, რომ ნებისმიერი არაცარიელი სიის ნამრავლი მიიღება პირველი რიცხვისა და რიცხვთა დარჩენილი სიის ნამრავლის ერთმანეთზე გამრავლების ოპერაციით. მაგალითად:

```
        product [2, 3, 4]
=               { product ფუნქციის გამოყენება }
        2 * product [3, 4]
=               { product ფუნქციის გამოყენება }
        2 * (3 * product [4])
=               { product ფუნქციის გამოყენება }
        2 * (3 * (4 * product []))
=               { product ფუნქციის გამოყენება }
        2 * (3 * (4 * 1))
=               { * ოპერატორის გამოყენება }
        24
```

გავიხსენოთ, რომ რეალურად სიას ჰასკელში აქვს ერთი ელემენტის ლოგიკური სტრუქტურა, რომელიც ამავე დროს cons ოპერატორს იყენებს. მაშასადამე, [2, 3, 4] ჩანაწერი მხოლოდ აბრევიატურაა 2 : (3 : (4 : [])) გამოსახულებისათვის და მეტი არაფერი. განვიხილოთ სიებზე რეკურსიის კიდევ ერთი მარტივი მაგალითი, რისთვისაც მივმართოთ *length* საბიბლიოთეკო ფუნქციას, რომელიც შეიძლება განისაზღვროს რეკურსიის ამავე შაბლონის გამოყენებით *product* ფუნქციის მსგავსად:

```
length          ::    [a] → Int
length []       =     0
length ( _ : xs) =    1 + length xs
```

მაშასადამე, ცარიელი სიის სიგრძე ნულია, ხოლო ნებისმიერი არაცარიელი სიის სიგრძე მისი კუდის სიგრძის მომდევნო მნიშვნელობაა. ყურადღება მიაქციეთ (_) ჩასმის შაბლონის გამოყენებას რეკურსიულ გამოსახულებაში. ეს შაბლონი ასახავს იმ ფაქტს, რომ სიის სიგრძე დამოკიდებული არ არის მისი ელემენტების მნიშვნელობაზე.

ახლა განვიხილოთ საბიბლიოთეკო ფუნქცია, რომელიც სიის შებრუნებას, შექცევას (რევერსს) ახორციელებს. რეკურსიის საშუალებით ეს ფუნქცია შეიძლება შემდეგნაირად განვსაზღვროთ:

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

მაშასადამე, ცარიელი სიის რევერსი კვლავ ცარიელი სიაა, ხოლო ნებისმიერი არაცარიელი სიის რევერსი ხორციელდება მიბმით მისი კუდის რევერსთან ერთელემენტიანი სიის, რომელიც საწყისი სიის თავს წარმოადგენს. მაგალითად:

```
        reverse [1, 2, 3]
=               { reverse ფუნქციის  გამოყენება }
        reverse [2, 3] ++ [1]
=               { reverse ფუნქციის  გამოყენება }
        (reverse [3] ++ [2])++ [1]
=               { reverse ფუნქციის  გამოყენება }
        ((reverse []++ [3])++ [2])++ [1]
=               { reverse ფუნქციის  გამოყენება }
        (([] ++ [3])++ [2]) ++ [1]
=               { ++ ოპერატორის  გამოყენება }
        [3, 2, 1]
```

თავის მხრივ, დამატების ++ ოპერატორი, რომელიც *reverse* ფუნქციის ზემოთ მოცემულ განსაზღვრებაში გამოიყენება, თავად შეიძლება იქნეს აღწერილი რეკურსიით თავის პირველ არგუმენტზე:

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

მაგალითად:

```
        [1, 2, 3] ++ [4, 5]
=               { ++ ოპერატორის  გამოყენება }
        1 : ([2, 3] ++ [4, 5])
=               { ++ ოპერატორის  გამოყენება }
        1 : (2 : ([3]++ [4, 5]))
=               { ++ ოპერატორის  გამოყენება }
        1 : (2 : (3 : ([] ++ [4, 5])))
=               { ++ ოპერატორის  გამოყენება }
        1 : (2 : (3 : [4, 5]))
=               { ჩაწერა სიის სახით }
        [1, 2, 3, 4, 5]
```

ამრიგად, რეკურსიული განსაზღვრება ++ ოპერატორისათვის ფორმალურად ასახავს იმ იდეას, რომ ორი სიის გადაბმა შეიძლება პირველი სიიდან ელემენტების პირის (ასლის) გადაღებით, ვიდრე ეს სია არ ამოიწურება, რის შემდეგ მეორე სია დაემატება ამ ასლს ბოლოში.

ამ ნაწილს ვამთავრებთ ორი მაგალითით, რომელიც ეხება რაკურსიას დახარისხებულ სიებზე. უპირველეს ყოვლისა, განვიხილოთ ფუნქცია, რომელიც დახარისხებულ სიაში ახორციელებს ნებისმიერი მოწესრიგებული ტიპის ახალი ელემენტის ჩასმას კიდევ ერთი ახალი დახარისხებული სიის მისაღებად. ეს ფუნქცია შემდეგი სახით შეიძლება განისაზღვროს:

```
insert                          ::      Ord a ⇒ a → [a] → [a]
insert x []                     =       [x]
insert x (y : ys) | x ≤ y       =       x : y : ys
                  | otherwise   =       y : insert x ys
```

მაშასადამე, ახალი ელემენტის ჩასმა ცარიელ სიაში იძლევა ერთელემენტიან სიას, მაშინ როცა არაცარიელი სიისათვის შედეგი დამოკიდებულია ახალი x ელემენტისა და სიის y თავის ურთიერთგანლაგებაზე. სახელდობრ, თუ $x \leq y$, მაშინ ახალი x ელემენტი მხოლოდ თავსდება სიის დასაწყისში და მეტი არაფერი, წინააღმდეგ შემთხვევაში y თავი საბოლოო სიის პირველი ელემენტი ხდება და შემდეგ იწყებენ ახალი ელემენტის ჩასმას მოცემული სიის კუდში. მაგალითად:

```
        insert 3 [1, 2, 4, 5]
=               { insert ფუნქციის გამოყენება }
        1 : insert 3 [2, 4, 5]
=               { insert ფუნქციის გამოყენება }
        1 : 2 : insert 3 [4, 5]
=               { insert ფუნქციის გამოყენება }
        1 : 2 : 3 : [4, 5]
=               { ჩაწერა სიის სახით }
        [1, 2, 3, 4, 5]
```

insert ფუნქციის გამოყენებით შესაძლებელია ახალი ფუნქციის განსაზღვრა, რომელიც ახორციელებს *დახარისხებას ჩასმით*[1] (ინგლ. *insertion sort*). სახელდობრ, ამ განსაზღვრებაში უნდა იქნეს გათვალისწინებული, რომ ცარიელი სია უკვე დახარისხებულია, ხოლო ნებისმიერი არაცარიელი სიის დახარისხება ხდება მისი თავის ჩასმით კუდის დახარისხების შედეგად მიღებულ სიაში:

```
isort           ::      Ord a ⇒ [a] → [a]
isort []        =       []
isort (x : xs)  =       insert x (isort xs)
```

მაგალითად:

```
        isort [3, 2, 1, 4]
=               { isort ფუნქციის გამოყენება }
        insert 3 (insert 2 (insert 1 (insert 4 [])))
=               { insert ფუნქციის გამოყენება }
        insert 3 (insert 2 (insert 1 [4]))
=               { insert ფუნქციის გამოყენება }
        insert 3 (insert 2 [1, 4])
=               { insert ფუნქციის გამოყენება }
        insert 3 [1, 2, 4]
=               { insert ფუნქციის გამოყენება }
        [1, 2, 3, 4]
```

[1] ხელით დახარისხების მარტივი და ძალიან არაეფექტური მეთოდი, როცა მონაცემთა მორიგი ელემენტი თავსდება სიის საჭირო ადგილზე სიის არსებულ ელემენტებთან შედარების შემდეგ.