**დეკლარაციული და კომპოზიციური სტილი**

Haskell-ში არსებობს რამდენიმე ჩადგმული გამოსახულება, რომელიც აადვილებს ფუნქციის აგებას და კოდს ხდის უფრო გასაგებს. ეს გამოსახულებები შეიძლება დაიყოს ორ ჯგუფად: გამოსახულებები, რომლებიც მხარს უჭერს ფუნქციის განსაზღვრის *დეკლარაციულ სტილს* (declarative style) და გამოსახულებები, რომლებიც მხარს უჭერს *კომპოზიციურ სტილს* (expression style). დეკლარაციული სტილით ფუნქციის განსაზღვრა უფრო წააგავს მათემატიკულ ნოტაციას. კომპოზიციური სტილის დროს ვაგებთ მარტივი გამოსახულებებიდან უფრო რთულ გამოსახულებებს, ვიყენებთ ამ გამოსახულებებთან სხვა გამოსახულებებს და ვაგებთ კიდევ უფრო დიდ გამოსახულებებს.

Haskell სრულად უჭერს მხარს ორივე სტილს. სტილის არჩევა, საზოგადოდ, გემოვნებაზეა დამოკოდებული.

**ლოკალური ცვლადები**

გავიხსენოთ სამკუთხედის ფართობის გამოთვლის ფორმულა სამკუთხედის სამი გვერდის მიხედვით, რომელიც ჰერონის ფორმულის სახელითაა ცნობილი:

*S* =

სადაც *a*, *b* და *c* – სამკუთხედის გვერდებია, ხოლო *p* არის ნახევარპერიმეტრი.

როგორ განვსაზღვროთ ეს ფუნქცია? ყველაზე მარტივად, შეგვიძლია დავწეროთ ასე:

square a b c **=** sqrt(p a b c **\*** (p a b c **-** a) **\*** (p a b c **-** b) **\*** (p a b c **-** c))

p a b c **=** (a **+** b **+** c) **/** 2

ეს ჩანაწერი ნამდვილად სჯობს შემდეგ ჩანაწერს:

square a b c **=** sqrt ((a**+**b**+**c)**/**2 **\*** ((a**+**b**+**c)**/**2 **-** a) **\*** ((a**+**b**+**c)**/**2 **-** b) **\*** ((a**+**b**+**c)**/**2 **-** c))

ორივე ჩანაწერში ხდება გამოსახულების გამოთვლის დუბლირება. სასურველია განსაზღვრება ისევე გამოიყურებოდეს, როგორც მათემატიკური განსაზღვრება:

square a b c **=** sqrt (p **\*** (p **-** a) **\*** (p **-** b) **\*** (p **-** c))

p **=** (a **+** b **+** c) **/** 2

საჭიროა p-მ იცოდეს, რომ a, b და c აიღება ფუნქციის არგუმენტებიდან. ასეთ შემთხვევებში გამოიყენება ლოკალური ცვლადები.

**where-გამოსახულება**

დეკლარატიულ სტილისას გამოიყენება **where**-გამოსახულება, რომელიც ასე იწერება:

square a b c **=** sqrt (p **\*** (p **-** a) **\*** (p **-** b) **\*** (p **-** c))

**where** p **=** (a **+** b **+** c) **/** 2

ან ასე:

square a b c **=** sqrt (p **\*** (p **-** a) **\*** (p **-** b) **\*** (p **-** c)) **where**

p **=** (a **+** b **+** c) **/** 2

ფუნქციის განსაზღვრას მოსდევს სიტყვა **where**, რომელსაცშემოყავს ლოკალური ცვლადების სინონიმები. ამასთან, ფუნქციის არგუმენტები იმყოფებიან სახელთა ხილვადობის არეში. სინონიმები შეიძლება იყოს რამდენიმე:

square a b c **=** sqrt (p **\*** pa **\*** pb **\*** pc)

**where** p **=** (a **+** b **+** c) **/** 2

pa **=** p **-** a

pb **=** p **-** b

pc **=** p **–** c

ლოკალური გამოსახულებების რიგი **where** გამოსახულებაში არ არის მნიშვნელოვანი.

**where** გამოსახულებაში შეიძლება განისაზღვროს ახალი ფუნქციები და ასევე, აღიწეროს მათი ტიპები:

add2 x **=** succ (succ x)

**where** succ **:: Int -> Int**

succ x **=** x **+** 1

ფუნქციის ტიპი შეიძლება არც აღიწეროს, ინტერპრეტატორი თვითონ მიხვდება:

add2 x **=** succ (succ x)

**where** succ x **=** x **+** 1

თუმცა, ზოგჯერ სასარგებლოა ფუნქციის ტიპის მითითება, როცა გამოიყენება ტიპების კლასები. განვიხილოთ კიდევ ერთი მაგალითი-სიის ფილტრაციის ფუნქცია, რომელიც განსაზღვრულია **Prelude-ში**:

filter **::** (a **-> Bool**) **->** [a] **->** [a]

filter p **[] = []**

filter p (x**:**xs) **= if** p x **then** x **:** rest **else** rest

**where** rest **=** filter p xs

აქ განსაზღვრულია ლოკალური ცვლადი rest, რომელიც მიუთითებს ფუნქციის რეკურსიულ გამოძახებას სიის დარჩენილ ნაწილით.

**where**-გამოსახულებები განისაზღვრება ფუნქციის განსაზღვრაში თითოეული განტოლებისთვის:

even **:: Int -> Bool**

even **Zero =** res

**where** res **= True**

even (**Succ Zero**) **=** res

**where** res **= False**

even x **=** even res

**where** (**Succ** (**Succ** res)) **=** x

რა თქმა უნდა, ამ მაგალითში **where** არ არის საჭირო, მაგრამ მოყვანილია იმის საჩვენებლად, თუ როგორ შეიძლება **where** განმარტება მიებას მოცემულ განტოლებას. მაგალითში განსაზღვრულია სამი ლოკალური ცვლადი ერთიდაიგივე სახელით. **where** გამოსახულებები შეიძლება მოიცეს **where**-ს შიგნითვე, მაგრამ სჯობს ღრმად ჩადგმულ გამოსახულებებს მოვერიდოთ.

**let-გამოსახულება**

კომპოზიციურ სტილში სამკუთხედის ფართობის გამოთვლის ფუნქციას აქვს შემდეგი სახე: გამოთვლის

square a b c **= let** p **=** (a **+** b **+** c) **/** 2

**in** sqrt (p **\*** (p **-** a) **\*** (p **-** b) **\*** (p **-** c))

სიტყვები **let** და **in** – გასაღები სიტყვებია. ისინი შეიძლება გამოყენებული იყოს გამოსახულების ნებისმიერ ადგილას:

square a b c **= let** p **=** (a **+** b **+** c) **/** 2

**in** sqrt ((**let** pa **=** p **-** a **in** p **\*** pa) **\***

(**let** pb **=** p **-** b

pc **=** p **-** c

**in** pb **\*** pc))

ამით ჩანს, რომ ისინი მიეკუთვნებიან კომპოზიციურ სტილს. **let** გამოსახულებები შეიძლება მონაწილეობდეს ნებისმიერ ქვეგამოსახულებაში, შეიძლება დაჯგუდეს ფრჩხილებით. ხოლო **where** გამოსახულებები მიბმულია ფუნქციის განმარტების განტოლებებთან.

განვსაზღვროთ ფილტრაციის ფუნქცია **let** -ის მეშვეობით:

filter **::** (a **-> Bool**) **->** [a] **->** [a]

filter p **[] = []**

filter p (x**:**xs) **=**

**let** rest **=** filter p xs

**in if** p x **then** x **:** rest **else** rest

**დეკომპოზიცია**

დეკომპოზიცია ანუ ნიმუშთან შედარება საშუალებას იძლევა შემადგენელი მნიშვნელობებიდან გამოიყოს ის მარტივი მნიშვნელობები, რომელთა საშუალებით იყვნენ ისინი შედგენილები და ამასთან, ორგანიზება გაუკეთდეს პირობით გამოთვლებს, რომელიც დამოკდებულია ფუნქციის შემავალ მნიშვნელობაზე:

not **True = False**

not **False = True**