Доклад

Доклад на тема функции в езика Haskell

На

Георги Горанов

12Б №9

1. Чисти функции - характеристики. Примери.
2. Чистите функционални функции (или изрази) нямат никакви странични ефекти (памет или I / O). Това означава, че чистите функции имат няколко полезни свойства, много от които могат да бъдат използвани за оптимизиране на кода:
   1. Ако не се използва в резултат на чистото изразяване, тя може да бъде отстранена без да се засягат други изрази.
   2. Ако чистата функция се нарича с аргументи, които не причиняват странични ефекти, резултатът е константа по отношение на този списък с аргументи (понякога се нарича референтна прозрачност), т.е. ако чистата функция отново е призована със същите аргументи, същият резултат ще бъде върнат.
   3. Ако няма зависимост от данни между два чисти израза, тяхната цел може да бъде обърната, или може да се извършва успоредно и те не могат да пречат един на друг (с други думи, оценката на всеки чист израз е безопасна).
   4. Ако пълния език не позволява странични ефекти, тогава всяка стратегия за оценка може да бъде използвана; това дава свободата на компилатора да преподреди или комбинира оценката на изразяване в програма.
3. Рекурсивни функции. Примери. Опашкова рекурсия.
   1. Рекурсивна реализация на цикли

* В Haskell няма цикли
* Циклите се реализират чрез рекурсия
* В Haskell всички функции са чисти (не могат да променят състояние)
* Итерациите се реализират с рекурсивни извиквания
* Итераторите се реализират като параметри и се променят при всяко рекурсивно извикване

repeatString str n =

if n == 0

then ""

else str ++ (repeatString str (n-1))

1. Опашкова рекурсия
   1. Опашковата рекурсия е рекурсия, при която последното извършвано действие е рекурсивно извикване
   2. Оптимизация наречена премахване на опашното извикване (tail call elimination)
   3. Вместо с последващо връщане рекурсивното обръщение се реализира със преход без връщане
   4. При тази рекурсия заделената в стека памет се преизползва вместо да се заделя нова
   5. Намалява разхода на памет и обикновено подобрява бързината на алгоритъма, но по-трудно се откриват грешки.

repeatStringLoop string result n =

if n == 0

then result

else repeatStringLoop string (result ++ string)(n-1)

repeatString string n = repeatStringLoop string string n

1. Функции от по-висок ред. Примери - map, filter
   1. За функционалните езици е характерно че функциите:

- имат тип;

- могат да са резултат от работата на други функции;

- могат да бъдат параметри на други функции.

- Функции, които са резултат от работата на други функции или са параметри на други функции, се наричат функции от по-висок ред.

По това те се различават от обикновените функции.

* 1. А) Функцията map.

map :: (a->b) -> [a] -> [b]

map f [ ] -> [ ]

map f (x:xs) -> f x : map f xs

doubleAll :: [Int] -> [Int]

doubleAll [ ] = [ ]

doubleAll list = map (\*2) list

* 1. Б) Функцията filter

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

filter p [ ] = [ ]

filter p (x:xs)

| p x = x : filter p xs

| otherwise = filter p xs

1. Списъци като параметри на функции. Примери. Празен списък, произволен списък, непразен списък.
   1. В Haskell списъците са хомогенна структура от данни. Той съхранява няколко елемента от един и същи тип. Може да са крайни или безкрайни. Елементите може да са изрази. Може да се дефинират списъци от елементи с конкретен тип ( например реални числа ) или с произволен.
   2. [Int] - списък от тип Int
   3. [Char] - списък от тип Char
   4. [a] - списък от произволен тип

* []- означава празен списък
* (x:xs)- означава списък, който не е празен, x е head на списъка, а xs - tail
* x - име на променлива, когато параметърът е произволен списък.