Синтаксис за дефиниране на функции

Трифон Трифонов

Функционално програмиране, спец. Информатика, 2016/17 г.

1 декември 2016 г.

Разглеждане на случаи

Можем да дефинниране на функции с разглеждане на случаи по параметрите.
Условието на всеки случай се нарича **пазач**.

- <име> {<параметър>}
  { | <пазач> = <израз> }$^{+}$

Разглеждане на случаи

Можем да дефинниране на функции с разглеждане на случаи по параметрите.
Условието на всеки случай се нарича **пазач**.

- <име> {<параметър>}
  { | <пазач> = <израз> }$^+$
- <име> <параметър$_1$> <параметър$_2$> ... <параметър$_k$>
  | <пазач$_1$> = <израз$_1$>
  ...
  | <пазач$_n$> = <израз$_n$>

Разглеждане на случаи

Можем да дефинниране на функции с разглеждане на случаи по параметрите.
Условието на всеки случай се нарича **пазач**.

- <име> {<параметър>}
  { | <пазач> = <израз> }$^+$
- <име> <параметър$_1$> <параметър$_2$> ... <параметър$_k$>
  | <пазач$_1$> = <израз$_1$>
  ...
  | <пазач$_n$> = <израз$_n$>
- ако <пазач$_1$> е `True` връща <израз$_1$>, а ако е `False`:
- ...
- ако <пазач$_n$> е `True` връща <израз$_n$>, а ако е `False`:
- грешка!

Разглеждане на случаи

Можем да дефинниране на функции с разглеждане на случаи по параметрите.
Условието на всеки случай се нарича **пазач**.

- <име> {<параметър>}
  { | <пазач> = <израз> }$^+$
- <име> <параметър$_1$> <параметър$_2$> ... <параметър$_k$>
  | <пазач$_1$> = <израз$_1$>
  ...
  | <пазач$_n$> = <израз$_n$>
- ако <пазач$_1$> е `True` връща <израз$_1$>, а ако е `False`:
- ...
- ако <пазач$_n$> е `True` връща <израз$_n$>, а ако е `False`:
- грешка!
- За удобство `Prelude` дефинира `otherwise = True`

Разглеждане на случаи — примери

```
fact n
  | n == 0  = 1
  | n > 0   = n * fact (n - 1)
```

Разглеждане на случаи — примери

```
fact n
  | n == 0  = 1
  | n > 0   = n * fact (n - 1)
```

- `fact (-5)` $\longrightarrow$?

Разглеждане на случаи — примери

```
fact n
  | n == 0  = 1
  | n > 0   = n * fact (n - 1)
```

- `fact (-5)` $\longrightarrow$ Грешка!

Разглеждане на случаи — примери

```
fact n
  | n == 0  = 1
  | n > 0   = n * fact (n - 1)
```

- `fact (-5)` $\longrightarrow$ Грешка!
- добра практика е да имаме изчерпателни случаи

Разглеждане на случаи — примери

```
fact n
  | n == 0  = 1
  | n > 0   = n * fact (n - 1)
```

- `fact (-5)` $\longrightarrow$ Грешка!
- добра практика е да имаме изчерпателни случаи
- можем да използваме стандартната функция `error`

Разглеждане на случаи — примери

```
fact n
  | n == 0  = 1
  | n > 0   = n * fact (n - 1)
  | n < 0   = error "подадено отрицателно число"
```

- `fact (-5)` $\longrightarrow$ Грешка!
- добра практика е да имаме изчерпателни случаи
- можем да използваме стандартната функция `error`

Разглеждане на случаи — примери

```
fact n
  | n == 0  = 1
  | n > 0   = n * fact (n - 1)
  | n < 0   = error "подадено отрицателно число"
```

- `fact (-5)` $\longrightarrow$ Грешка!
- добра практика е да имаме изчерпателни случаи
- можем да използваме стандартната функция `error`

```
grade x
  | x >= 5.5     = "Отличен"
  | x >= 4.5     = "Много добър"
  | x >= 3.5     = "Добър"
  | x >= 3       = "Среден"
  | otherwise    = "Слаб"
```

Локални дефиниции с `let`

- `let` { <дефиниция> }$^{+}$
  `in` <тяло>

Локални дефиниции с `let`

- `let` { <дефиниция> }$^+$
  `in` <тяло>
- `let` <дефиниция$_1$>
  <дефиниция$_2$>
  ...
  <дефиниция$_n$>
  `in` <тяло>

Локални дефиниции с `let`

- `let` { <дефиниция> }$^+$
  `in` <тяло>
- `let` <дефиниция$_1$>
  <дефиниция$_2$>
  ...
  <дефиниция$_n$>
  `in` <тяло>
- <дефиниция$_i$> се въвеждат едновременно
- областта на действие на дефинициите е само в рамките на `let` конструкцията
- може да са взаимно рекурсивни

Примери за `let`

- `let x = 5 in x + 3` $\longrightarrow$ 8

Примери за `let`

- `let x = 5 in x + 3` $\longrightarrow$ `8`
- ```
  let f x = y + x   ⟶ ?
      y = 7
  in  f 2 * y
  ```
  ```

Примери за `let`

- `let x = 5 in x + 3` $\longrightarrow$ 8
- ```
  let f x = y + x   ⟶ 63
      y = 7
  in  f 2 * y
  ```
  ```

Примери за let

- `let x = 5 in x + 3` $\longrightarrow$ `8`
- ```
  let f x = y + x  ⟶ 63
      y = 7
  in  f 2 * y
  ```
- ```
  fact2 n = let fact n =  if n == 0 then 1
                          else n * fact (n-1)
            in (fact n)^2
  ```

Примери за let

- `let x = 5 in x + 3` $\longrightarrow$ `8`
- ```
  let f x = y + x ⟶ 63
      y = 7
  in  f 2 * y
  ```
- ```
  fact2 n = let fact n =  if n == 0 then 1
                          else n * fact (n-1)
            in (fact n)^2
  ```
- В интерактивен режим (GHCi) `let` може да се използва без `in` за въвеждане на нови дефиниции