

Основни понятия в програмирането на езика Haskell

Общи сведения за езика Haskell

Haskell е език за строго функционално програмиране. Създаден е в края на 80-те години на 20-ти век. Носи името на Haskell B. Curry – един от пионерите на λ -смятането (математическа теория на функциите, дала тласък в развитието на множество езици за функционално програмиране).

В близкото минало най-популярната среда за програмиране на Haskell беше **Hugs 98** (<http://cvs.haskell.org/Hugs/>, последна версия – от 2006 г.). Тя предоставя много добри средства за обучение и се разпространява безплатно за множество платформи.

От 2009 г. се предлага безплатно нова среда за програмиране и разработка на софтуерни приложения на Haskell, която постепенно придобива статута на стандарт – Haskell Platform (<http://hackage.haskell.org/platform/>, последна версия – от 2012 г.).

Haskell home page:

<http://www.haskell.org/haskellwiki/Haskell>

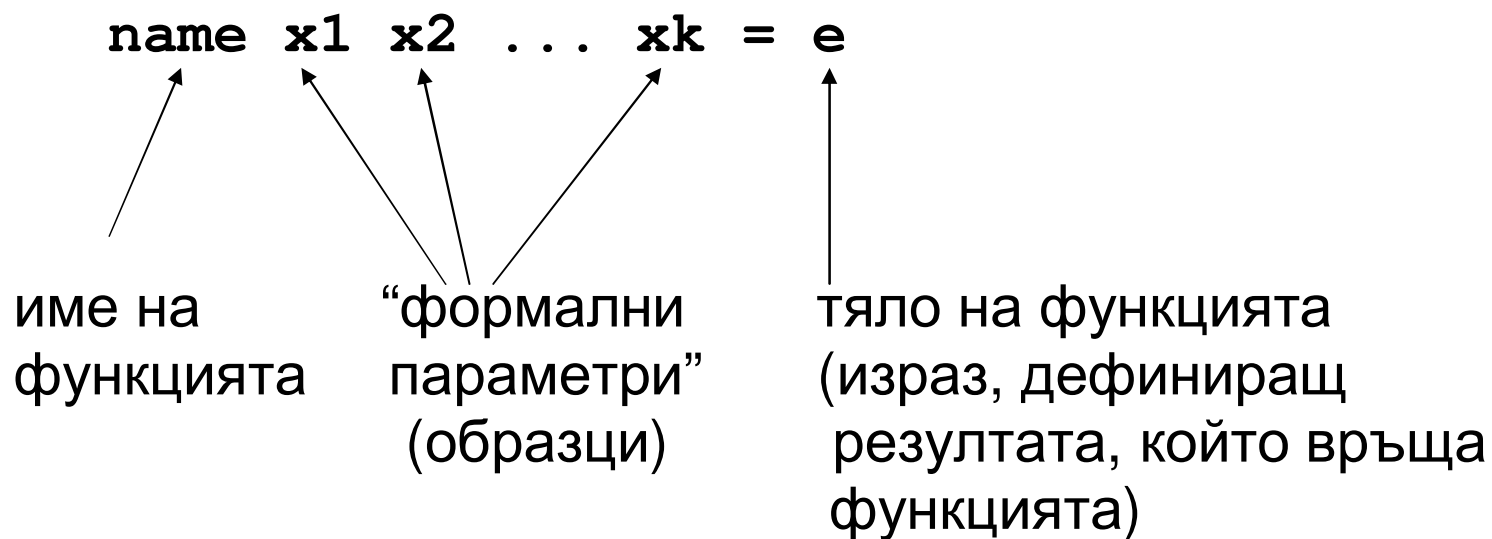
Дефиниране на “променливи” (нелитерални константи)

Общ вид:

```
name :: type  
name = expression
```

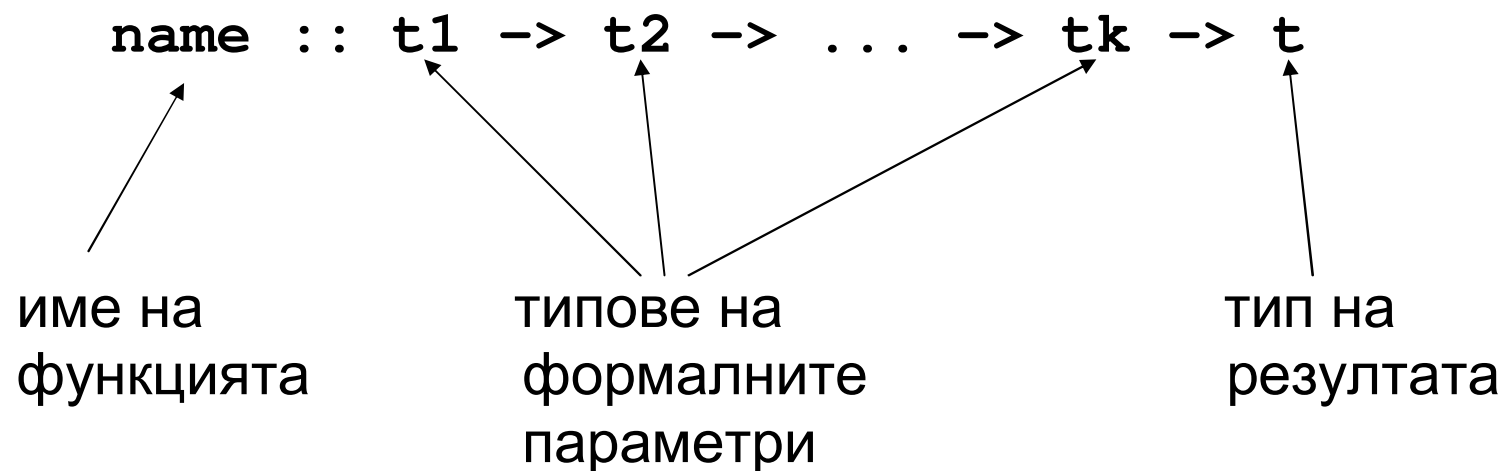
Дефиниране на функции

Общ вид на дефиниция на функция (най-прост вариант):



Дефиницията на функция трябва да бъде предшествана от декларация на нейния тип (на типовете на аргументите и типа на резултата, който връща функцията).

Общ вид на декларация на типа на функция:



Примери

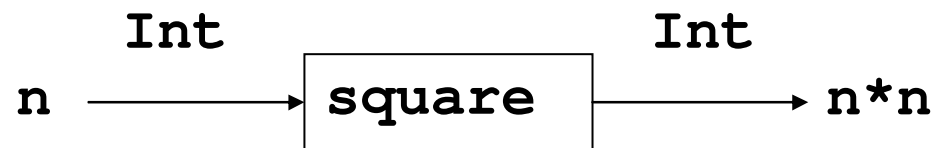
```
square :: Int -> Int  
square n = n*n
```

```
> square 3
```

```
9
```

```
> square 5
```

```
25
```



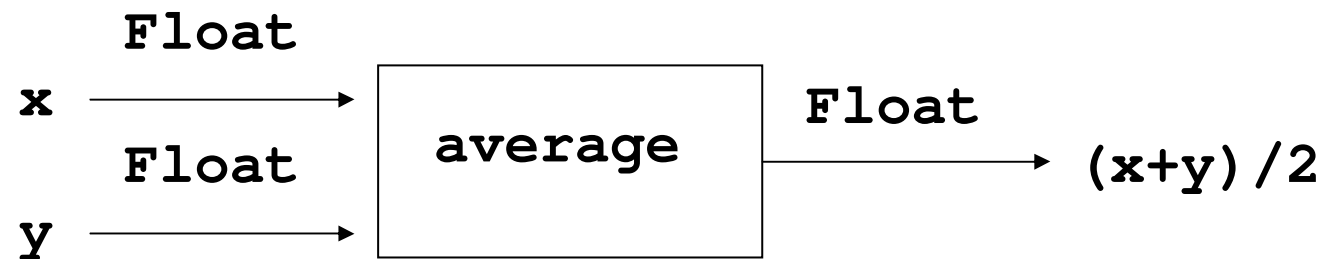
```
average :: Float -> Float -> Float
average x y = (x+y)/2
```

```
> average 3.4 5.6
```

```
4.5
```

```
> average 3 4
```

```
3.5
```



Общ вид на програмата на Haskell

Програмите на Haskell обикновено се наричат **скриптове** (**scripts**). Освен програмния код (поредица от дефиниции на функции) един скрипт може да съдържа и коментари.

Има два различни стила на писане на скриптове, които съответстват на две различни философии на програмиране.

Традиционно всичко в един програмен файл (файл с изходния код на програма на Haskell) се интерпретира като програмен текст (код), освен ако за нещо е отбелязано специално, че представлява коментар. Скриптовете, написани в такъв (traditional) стил, се съхраняват във файлове с разширение “.hs”.

Традиционно коментари се означават по два начина. Символът “--” означава начало на коментар, който продължава от съответната позиция до края на текущия ред. Коментари, които съдържат произволен брой знакове и евентуално заемат повече от един ред, могат да бъдат заключени между символите “{-“ и “-}”.

Алтернативният (literate) подход предполага, че всичко във файла е коментар освен частите от текста, специално означени като програмен код. В Пример 2 програмният текст е само в редовете, започващи с “>” и отделени от останалия текст с празни редове.

Този вид скриптове се съхраняват във файлове с разширение “.lhs”.

Пример 1. A traditional script

```
{- #####  
    MyFirstScript.hs  
##### -}  
  
-- The value size is an integer (Int), defined to be  
-- the sum of 12 and 13.  
  
size :: Int  
size = 12+13  
  
-- The function to square an integer.  
  
square :: Int -> Int  
square n = n*n
```

```
-- The function to double an integer.
```

```
double :: Int -> Int  
double n = 2*n
```

```
-- An example using double, square and size.
```

```
example :: Int  
example = double (size - square (2+2))
```

Пример 2. A literate script

```
{- #####  
    MyFirstLiterate.lhs  
##### -}
```

The value `size` is an integer (`Int`), defined to be the sum of 12 and 13.

```
> size :: Int  
> size = 12+13
```

The function to square an integer.

```
> square :: Int -> Int  
> square n = n*n
```

The function to double an integer.

```
> double :: Int -> Int
> double n = 2*n
```

An example using double, square and size.

```
> example :: Int
> example = double (size - square (2+2))
```

Библиотеки на Haskell

Haskell поддържа множество вградени типове данни: цели и реални числа, булеви стойности, низове, списъци и др., както и предлага вградени функции за работа с данни от тези типове.

Дефинициите на основните вградени функции в езика се съдържат във файл (**the standard prelude**) с името `Prelude.hs`. По подразбиране при стартиране на съответната среда за програмиране на Haskell най-напред се зарежда `the standard prelude`, след което потребителят може да започне своята работа.

Напоследък, с цел намаляване на обема на `the standard prelude`, дефинициите на част от вградените функции се преместват от `the standard prelude` в множество **стандартни библиотеки**, които могат да бъдат включени от потребителя в средата на Haskell при необходимост.

Модули

Възможно е текстът на една програма на Haskell да бъде разделен на множество компоненти, наречени **модули**.

Всеки модул има свое **име** и може да съдържа множество от дефиниции на Haskell. За да се дефинира даден модул, например `Aut`, е необходимо в началото на програмния текст в съответния файл да се включи ред от типа на

```
module Aut where
```

```
.....
```

Един модул може да **импортира** дефиниции от други модули. Например модулът `Вее` ще може да импортира дефиниции от модула `Aut` чрез включване на оператор `import` както следва:

```
module Bee where
import Aut
. . . . .
```

В случая операторът `import` означава, че при дефинирането на функции в `Bee` могат да се използват всички (**видими**) дефиниции от `Aut`.

Механизмът на модулите поддържа споменатите по-горе библиотеки.

Механизмът на модулите позволява да се определи кои дефиниции да бъдат достъпни чрез **експортиране** от даден модул за употреба от други модули.

Примитивни типове данни в Haskell

Булеви стойности (Bool)

Булевият тип в Haskell се нарича Bool. Булеви константи са True и False, а вградените Булеви оператори, поддържани от езика, са:

&&	and
	or
not	not

Цели числа (числа с фиксирана точка: Int и Integer)

Целите числа (числата с фиксирана точка) в Haskell са от тип Int или Integer. За представяне на целите числа от тип Int се използва фиксирано пространство (32 бита), което означава, че типът Int съдържа краен брой елементи. Константата **maxBound** има за стойност максималното число от тип Int ($2^{31}-1 = 2147483647$).

За работа с цели числа с неограничена точност може да бъде използван типът Integer.

Haskell поддържа следните вградени оператори за работа с цели числа:

+	Сума на две цели числа.
*	Произведение на две цели числа.
^	Повдигане на степен; 2^3 е 8.
-	Разлика на две цели числа (при инфиксна употреба: $a-b$) или унарен минус (при префиксна употреба: $-a$).
div	Частно при целочислено деление, например <code>div 14 3</code> е 4. Може да се запише и инфиксно: <code>14 `div` 3</code> .
mod	Остатък при целочислено деление, например <code>mod 14 3</code> или <code>14 `mod` 3</code> .
abs	Абсолютната стойност на дадено цяло число (числото без неговия знак).
negate	Функция, която променя знака на дадено цяло число.

Забележка. Чрез заграждане на името на всяка двуаргументна функция в обратни апострофи (backquotes) е възможно записът на обръщението към тази функция да стане инфиксен.

Вградени оператори за сравнения:

>	greater than
>=	greater than or equal to
==	equal to (може да се използва и при аргументи от други типове)
/=	not equal to
<=	less than or equal to
<	less than

Реални числа (числа с плаваща точка: Float, Double и Rational)

За представянето на числата с плаваща точка от тип Float в Haskell се използва фиксирано пространство, което рефлектира върху точността на работата с този тип числа.

Допустим запис:

- като десетични дроби, например

0.31426

-23.12

567.345

4513.0

- scientific notation (запис с мантика и порядък), например

231.61e7 $231.61 \times 10^7 = 2316100000$

231.61e-2 $231.61 \times 10^{-2} = 2.3161$

-3.412e03 $-3.412 \times 10^3 = -3412$

Типът Double се използва за работа с числа с плаваща точка с по-голяма (двойна) точност, а типът Rational се използва за представяне на реални (по-точно, рационални) числа с пълна (неограничена) точност.

Някои вградени аритметични оператори:

+	-	*	Float -> Float -> Float
/			Float -> Float -> Float
^			Float -> Int -> Float (x^n за неотрицателно цяло n)
**			Float -> Float -> Float (x^y)
==	/=	<	Float -> Float -> Bool
>	<=	>=	Float -> Float -> Bool
signum			Float -> Float (връща резултат 1.0, 0.0 или -1.0)
sqrt			Float -> Float

Знакове (characters, Char)

Представяват отделни знакове, заградени в единични кавички (апострофи), например 'd' или '3'.

Връзка между знаковете и техните ASCII кодове:

```
ord :: Char -> Int
```

```
chr :: Int -> Char
```

Конвертирането на малки букви към главни изисква към съответния код да бъде прибавено определено отместване:

```
offset :: Int
```

```
offset = ord 'A' - ord 'a'
```

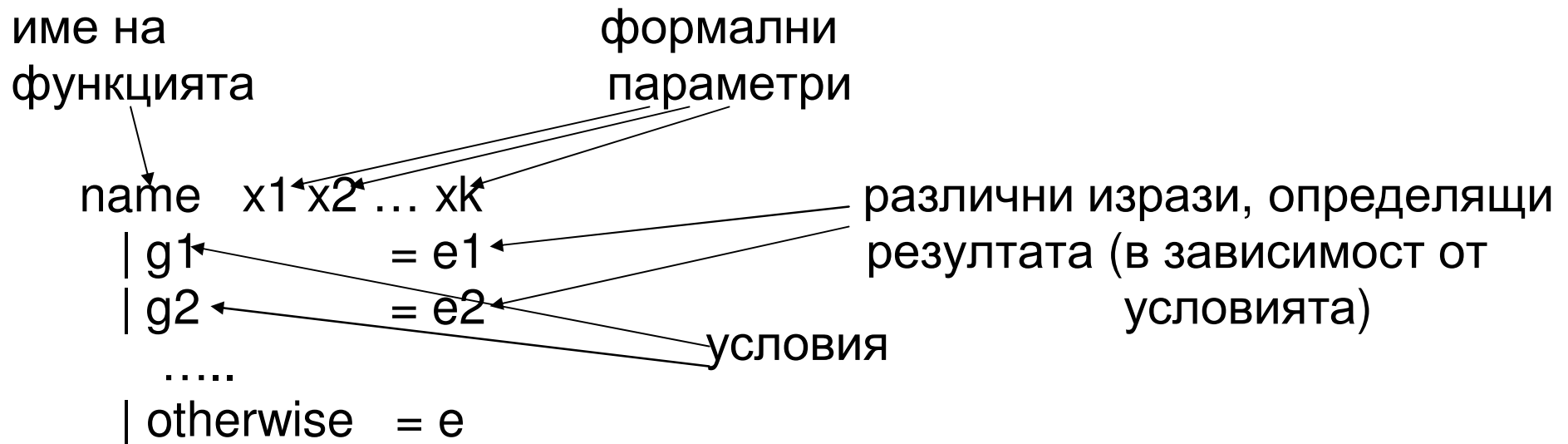
```
toUpper :: Char -> Char
```

```
toUpper ch = chr (ord ch + offset)
```

Програмиране на условия (guards)

Условието (guard) е Булев израз. Условия се използват, когато трябва да се опишат различни случаи в дефиницията на функция. Съществува аналогия между условията (guards) в Haskell и специалната форма cond в Scheme.

Общ вид на дефиниция на функция с условия:



Забележка. Клаузата otherwise не е задължителна.

Примери

```
max :: Int -> Int -> Int
```

```
max x y  
  | x >= y      = x  
  | otherwise   = y
```

```
maxThree :: Int -> Int -> Int -> Int
```

```
maxThree x y z  
  | x >= y && x >= z      = x  
  | y >= z                = y  
  | otherwise            = z
```

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

Условни изрази

Аналогично на Scheme и в езика Haskell е възможно да се описват условни изрази в термините на конструкцията `if ... then ... else`.

Общ вид на условен израз в Haskell:

```
if condition then m else n
```

Пример

```
max :: Int -> Int -> Int
max x y
    = if x >= y then x else y
```