PROGRAMOWANIE FUNKCYJNE HASKELL

WPROWADZENIE

HASKELL

- **Nazwa pochodzi od imienia znanego logika Haskell'a Brooks'a Curry'ego (1900-1982)**

- Jest jednym z wielu funkcyjnych języków programowania, do których należą m.in. Lisp, Erlang i inne.

- Jest językiem czysto funkcyjnym, dzięki czemu nie pozwala na efekty uboczne. Głównym założeniem języków czysto funkcyjnych jest to, że wynik działania funkcji jest uzależniony wyłącznie od przekazywanych jej argumentów. Za każdym razem, gdy wywołamy funkcję z takimi samymi parametrami, zwróci ona taką samą wartość.

- Jest językiem „leniwym" (ang. „lazy", „non strict"), gdyż wyrażenia, które nie są potrzebne, by ustalić odpowiedź na dany problem nie są wyznaczane. Leniwe wartościowanie gwarantuje, że nic nie zostanie policzone niepotrzebnie.

- Jest językiem stosującym silnie typowanie . Niemożliwa jest więc przypadkowa (niejawna) konwersja, np. Double do Int.

- Jest zwięzły. Programy są krótsze, dzięki czemu łatwiej jest analizować poszczególne funkcje i lokalizować błędy.

- Jest językiem wysokiego poziomu – programy napisane w Haskellu często są bardzo podobne do opisu algorytmu. Pisząc funkcje na wyższym poziomie abstrakcji i zostawiając szczegóły kompilatorowi, zmniejszamy szanse pojawienia się błędu.

- Zarządza pamięcią – programista zajmuje się jedynie implementacją algorytmu.

- Jest modularny – Haskell oferuje bardzo wiele metod łączenia modułów.

- Język Haskell, podobnie jak języki obiektowe, umożliwia tworzenie abstrakcyjnych struktur danych, polimorfizm i enkapsulację. Enkapsulacja jest realizowana poprzez umieszczenie każdego typu danych w osobnym module.

NAJBARDZIEJ POPULARNE KOMPILATORY

GHC the Glasgow Haskell Compiler
Dostępny pod adresem http://www.haskell.org/ghc/

The Haskell Interpreter Hugs
Dostępny pod adresem http://haskell.org/hugs/

Polecane tutoriale i wykłady:

Ze strony głównej interpretera Hugs wchodzimy kolejno:

Haskell 98 – przejście do strony http://www.haskell.org

U dołu strony link do strony z materiałami
wiki.haskell.org.
Wybieramy
Books && tutorials
Learn You a Haskell for Great Good!
Haskell Wikibook

Graham Hutton „Programming in Haskell”
Strona Grahama Huttona: www.cs.nott.ac.uk/~pszgmh
Wykłady w postaci slajdów i video na podstawie wymienionego podręcznika

SPOSÓB PRACY Z HASKELLEM

• Haskell umożliwia pracę interaktywną. Praca taka zwana jest sesją.

Interpreter Hugs:

```
||   || ||  || ||  || ||__       Hugs 98: Based on the Haskell 98 standard
||___|| ||__|| ||__||  __||      Copyright (c) 1994-2005
||---||         ___||            World Wide Web: http://haskell.org/hugs
||   ||                          Bugs: http://hackage.haskell.org/trac/hugs
||   || Version: Sep 2006
_____________________________________________

Haskell 98 mode: Restart with command line option -98 to enable extensions

Type :? for help
```

Interpreter Hugs w linii komend wyznacza wartości wyrażeń podane przez użytkownika. Po znaku zachęty wpisujemy wyrażenie, którego wartość chcemy wyznaczyć i naciskamy Enter. W nowej linii otrzymujemy odpowiedź. Jest to wartość wpisanego wyrażenia lub komunikat o błędzie.

Hugs > Wyrażenie
Wartość wyrażenia

```
Hugs> "Hello!"
"Hello!"

Hugs> reverse "Hello!"
"!olleH"


Hugs> "Witaj Haskell"
"Witaj Haskell"


Hugs> 5*3+4
19

Hugs> sqrt 9
3.0

Hugs> pi^2*5
49.3480220054468
Hugs>
```

Skrót “$$”

Możemy użyć tego skrótu, aby odwołać się do ostatnio wyznaczonej wartości, co pozwala na wygodne użycie interpretera.

```
Hugs> 2+3
5
Hugs> 2^$$
ERROR - Syntax error in expression (unexpected end of input)
Hugs> 2 ^ $$
32
Hugs> mod $$ 3
2
Hugs> $$ 'mod' 3
ERROR - Improperly terminated character constant
Hugs> $$ `mod` 3
2

Hugs> replivate 3 "hello"
ERROR - Undefined variable "replivate"

Hugs> replicate 3 "hello"
["hello","hello","hello"]

Hugs> concat $$
"hellohellohello"

Hugs> length $$
15
Hugs>
```

Odpowiedź interpretera a wersja drukowalna

Odpowiedź otrzymana z interpretera jest w postaci umożliwiającej jej ponowne użycie, ale możemy ją otrzymać w bardziej tradycyjnej postaci

```
Hugs> "Dzien Dobry"
"Dzien Dobry"

Hugs> putStr $$
Dzien Dobry

Hugs>
```

Polecenia interpretera

W interpreterze dostępne są polecenia wyższego poziomu (meta-level commands)

```
Hugs> :?
LIST OF COMMANDS:  Any command may be abbreviated to :c where
c is the first character in the full name.

:load <filenames>   load modules from specified files
:load               clear all files except prelude
:also <filenames>   read additional modules
:reload             repeat last load command
:edit <filename>    edit file
:edit               edit last module
:module <module>    set module for evaluating expressions
<expr>              evaluate expression
:type <expr>        print type of expression
:?                  display this list of commands
:set <options>      set command line options
:set                help on command line options
:names [pat]        list names currently in scope
:info <names>       describe named objects
:browse <modules>   browse names exported by <modules>
:main <aruments>    run the main function with the given arguments
:find <name>        edit module containing definition of name
:cd dir             change directory
:gc                 force garbage collection
:version            print Hugs version
:quit               exit Hugs interpreter
```

Wyznaczanie wartości wyrażeń arytmetycznych.

Wartości wyrażeń arytmetycznych możemy wyznaczać bezpośrednio w interpreterze.

Uwaga. Funkcje i operatory arytmetyczne mogą być używane zarówno w postaci prefiksowej, jak i infiksowej.

Postać prefiksowa: div x y mod x y (+) 5 2 (*) 5 2

Postać infiksowa: x `div` y x `mod` y 5 + 2 5 * 2

Postać infiksowa jest bardziej naturalna dla operatorów arytmetycznych takich jak + - * / a postać prefiksowa dla funkcji arytmetycznych.

```
Hugs> 5+2
7
Hugs> 5-2
3
Hugs> 5*2
10
Hugs> 5/2
2.5

Hugs> (+) 5 2
7
Hugs> (-) 5 2
3
Hugs> (*) 5 2
10
Hugs> (/) 5 2
2.5

Hugs> div 5 2
2
Hugs> mod 5 2
1

Hugs> 5 `div` 2
2
Hugs> 5 `mod` 2
1
```

Priorytet i łączność operatorów.

Operatory arytmetyczne i logiczne

Operator	Działanie	Priorytet	Łączność
+	dodawanie	6	L
-	odejmowanie	6	L
*	mnożenie	7	L
/	dzielenie	7	L
^, ^^, **	potęgowanie	8	P
`mod`	dzielenie modulo	7	L
`div`	dzielenie całkowite	7	L
<	mniejszy	4	N
<=	mniejszy lub równy	4	N
>	większy	4	N
>=	większy lub równy	4	N
==	równy	4	N
/=	różny	4	N
&&	AND	3	P
||	OR	2	P

Priorytety operatorów decydują o tym, który operator ma pierwszeństwo wykonania przed innym. Priorytet operatora jest liczbą całkowitą z zakresu od 0 do 9 włącznie (im wyższy tym silniejszy). Priorytet 10 jest zarezerwowany dla funkcji.

O kolejności wykonywania operatorów oprócz priorytetu decyduje ponadto własność „fixity", która określa czy operator wiąże w lewo - L („left-associative" – łączny lewostronnie), w prawo - P („right-associative" – łączny prawostronnie), czy jest niełączny - N („non-associative").

Polecenie
Sprawdź typ łączności operatorów: + i -, * i /, ^ , `mod`, `div` wyznaczając wartości wyrażeń:

3 + 4 - 5 + 1
((3 + 4) – 5) +1
3 + (4 – (5 + 1))

3 * 4 / 2 * 5
((3 * 4) / 2) * 5
3 *(4 / (2 * 5))

3 ^ 2
3 ^ 2 ^ 3
(3 ^ 2) ^ 3
3 ^ (2 ^ 3)

10 `mod` 6 `mod` 4
(10 `mod` 6) `mod` 4
10 `mod` (6 `mod` 4)

16 `div` 3 `div` 2
(16 `div` 3) `div` 2
16 `div` (3 `div` 2)

Sprawdź, że mod jako funkcja ma wyższy priorytet niż operator `mod' , wyznaczając wartości wyrażeń:

5 `mod` 2 ^ 3
mod 5 2 ^ 3

Definiowanie i stosowanie funkcji w interpreterze.

Jeżeli np. chcemy w interpreterze zdefiniować funkcję $kwadrat(x) = x^2$ i zastosować do argumentu 3 możemy to zrobić na dwa sposoby:

kwadrat 3 **where** kwadrat x = x * x

lub

let kwadrat x = x * x **in** kwadrat 3

```
Hugs> kwadrat 3 where kwadrat x = x*x
9

Hugs> let kwadrat x = x*x in kwadrat 3
9

Hugs> kwadrat 5
ERROR - Undefined variable "kwadrat"

Hugs> kwadrat(kwadrat 3) where kwadrat x = x*x
81

Hugs>
```

Definiując funkcję w interpreterze można ją wykorzystać tylko w linii, w której została zdefiniowana.

Skrypty

Haskell daje możliwość tworzenia definicji złożonych funkcji, które wykorzystujemy w obliczeniach. Zbiór definicji funkcji nazywamy skryptem.
Aby definicje funkcji mogły zostać użyte wielokrotnie należy je zapisać za pomocą dowolnego edytora tekstowego i umieścić w pliku z rozszerzeniem **.hs**.

Przykład definicji:

```
suma_ciagu :: Integer -> Integer
suma_ciagu  x = x * (x+1) `div` 2
```

Definicja funkcji składa się z dwóch części:
1) opisu typu funkcji, który może być pominięty, jeżeli nie prowadzi to do niejednoznaczności,
2) sposobu wyliczenia wartości funkcji

Składnia Haskella, w wielu przypadkach, pozwala obejść się bez znaków średników i przecinków ; ich rolę spełnia odpowiedni układ tekstu.