Linguagem de Programação
Introdução à Linguagem Hugs98

TURMA A
PROF. MARCELO LADEIRA CIC/UNB

Visão de Mundo

Programar em uma LP exige pensar com os significados das suas construções.

Cada paradigma (visão) tem construções que lhe são peculiares:

- Procedural (ou imperativo)
  - Solução algorítmica (passo a passo).
- Funcional
  - Declara a solução como valores a retornar.
- Relacional (ou lógico)
  - Declara a solução como relação entre entidades do discurso.
- Orientado a objetos
  - Descreve o problema em termos do próprio problema, ao invés de descrevê-lo em termos de um algoritmo que o computador vai rodar.

Linguagens Declarativas

Realizam o processamento simbólico

Processam listas (ES)

Bloco básico de construção: funções

Declaram o problema fazendo sua especificação (não resolvendo passo a passo)

implementam funções complexas,

implementam metas programas

Tratam símbolos e operações da lógica matemática com relativa facilidade.

Introdução à Linguagem Hugs98

Referência

- *Hugs98 User Manual*

Autores

- Mark P. Jones
- John C. Peterson

Distribuição e informações

- http://haskell.org/hugs/
- http://www.haskell.org/hugs/

hugs online

- Latest News
- Downloading
- Frequently Asked Questions
- Hugs Bugs & Features
- Documentation
- Developers' page

Note: *Hugs is no longer in development. The content on these pages is provided as a historical reference and as a resource for those who are still interested in experimenting with or otherwise exploring the system.*

Ambiente HUGS

É um interpretador com o ciclo lê-avalia-exibe resultados.

Funções primitivas:

- lidas do Prelude

Funções definidas pelo usuário:

- usando expressões let ou where
- carregadas de um arquivo de texto.

Tela inicial do interpretador Hugs

Comandos no Prelude

Main> :?

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

:project <filename> use project file

: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 defined in <modules>

:find <name> edit module containing definition of name

:! command shell escape

:cd dir change directory

:gc force garbage collection

:version print Hugs version

:quit exit Hugs interpreter

MDC

Funcional

```
mdc a 0 = a
mdc a b = mdc b (a `mod` b)
```

Imperativo

```
function mdc(a,b:integer):integer
    var t:integer;
     begin
              while b<> 0 do begin
          t := b;  b:=a mod b;  a:= t;
              end;
         mdc :=a ;
      end;
```

Lógico

```
mdc(A,0,A).
mdc(A,B,X) :-
                    BB is A mod B, mdc(B,
BB, X).
```

Expressões let e where

let <definição> in <expressão>

let {d_1; d_2; ....; d_n} in expressão

```
Prelude> let soma a b = a+b in soma 12 15
27 :: Integer
```

<expressão> where <definição>

expressão where {d_1; d_2; ....; d_n}

```
Prelude> fat 5 where fat n = product [1..n]
120 :: Integer
```

Re-escrita

Permite transformar (re-escrever) termos em outros

É a base do processo de avaliação de expressões

Programas funcionais são executados usando a redução ou re-escrita de termos.

Exemplo:

```
append   [ ]     ys  =  ys
append (x:xs)  ys   = x:append xs ys
```

```
append [1,3,5] [4,6] =1:append [3,5] [4,6]
                     = 1:3:append [5] [4,6]
                     = 1:3:5:append [ ] [4,6]
                     = 1:3:5:[4,6]
                     = [1,3,5,4,6]
```