Conceitos de Linguagens de Programação

Aula 09 – Programação Funcional (Haskell)

Edirlei Soares de Lima
<edirlei@iprj.uerj.br>

Programação Funcional

- A programação funcional modela um problema computacional como uma **coleção de funções** matemáticas, cada uma com um domínio de entrada e um resultado.

- As funções interagem e combinam entre si usando **composição funcional**, **condições** e **recursão**.

- Linguagens importantes de programação funcional são:
  - Lisp, Scheme, **Haskell** e ML.

Haskell vs C

- **Exemplo:** distancia entre dois pontos em C

```
#include <stdio.h>
#include <math.h>
float dist(float PX1, float PY1, float PX2, float PY2)
{
  float res = sqrt((PX2 - PX1)*(PX2 - PX1) +
                   (PY2 - PY1)*(PY2 - PY1));
  return res;
}
int main()
{
  float f = dist(2.0, 4.0, 3.0, 1.0);
  printf("Distance = %f", f);
  return 0;
}
```

Haskell vs C

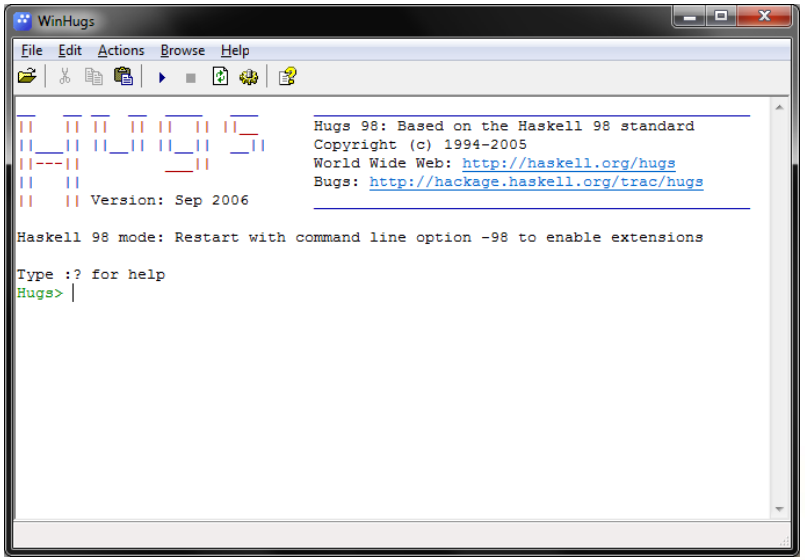
- **Exemplo:** distancia entre dois pontos em Haskell

```
dist x1 y1 x2 y2 = sqrt(((x2 - x1)^2) + ((y2 - y1)^2))
```

```
> dist 2.0 4.0 3.0 1.0
```

Hugs

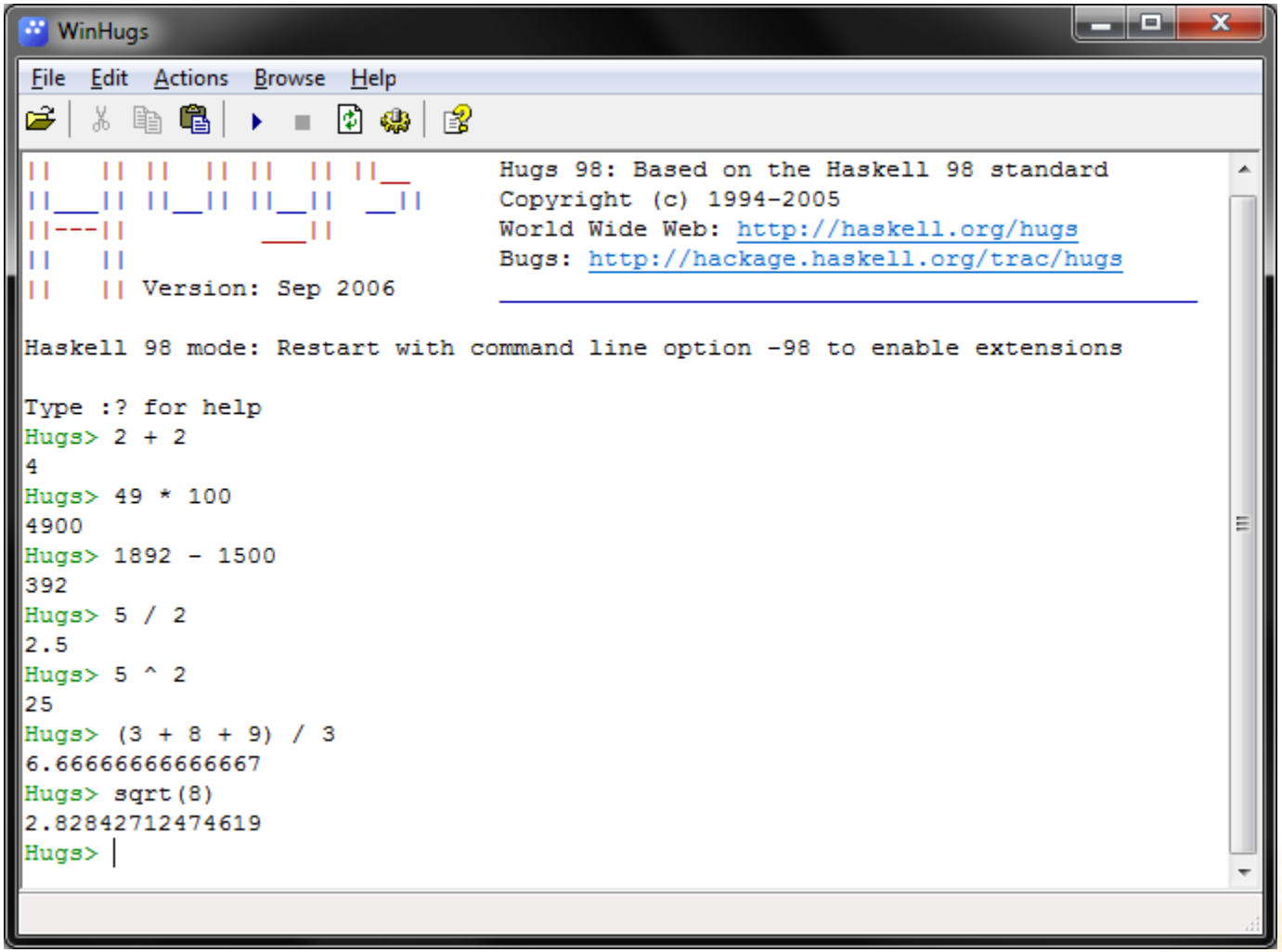
- **Hugs** é uma implementação da linguagem Haskell que pode ser executada em **múltiplas plataformas**.



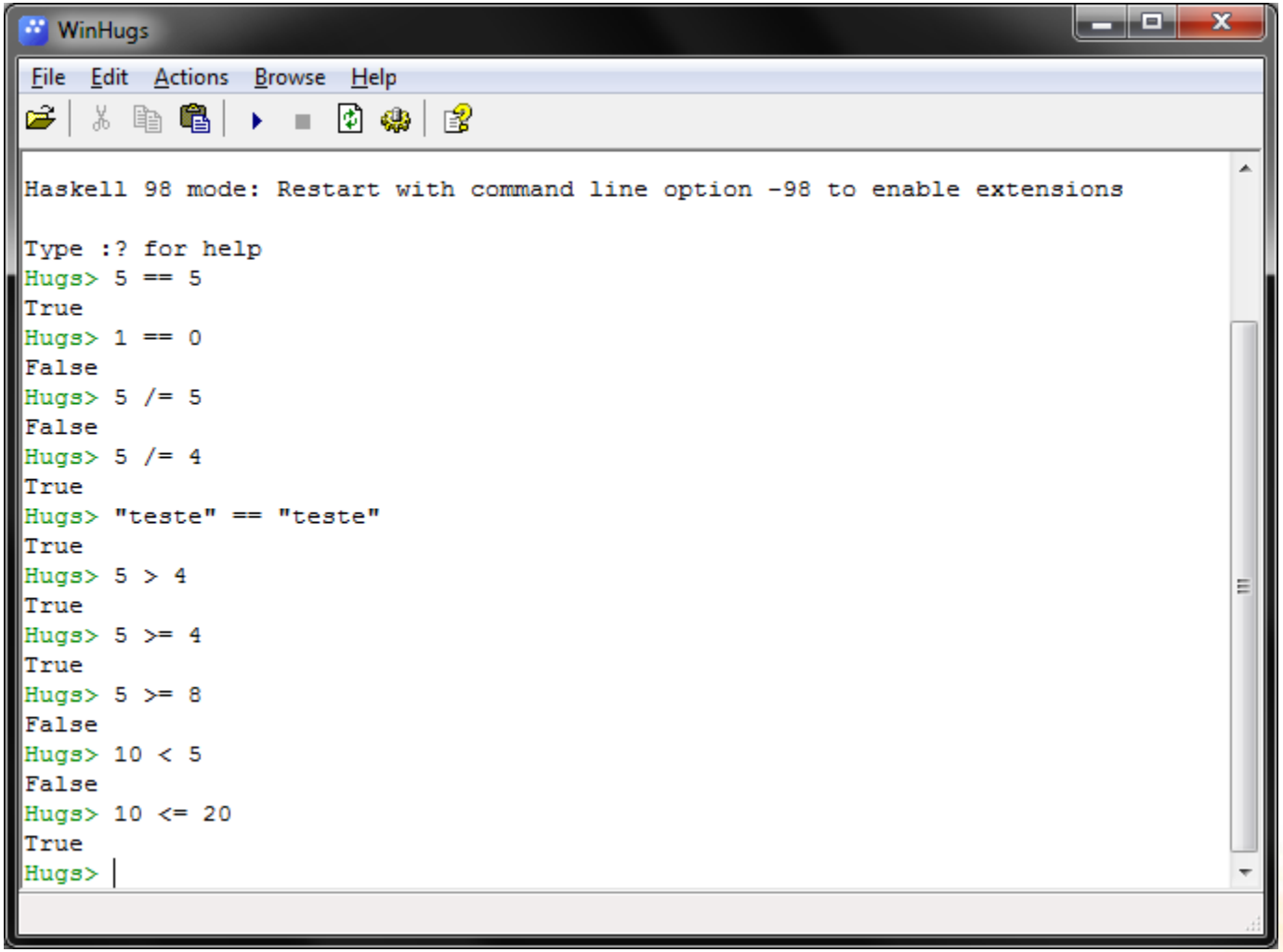
- Baseado no padrão Haskell 98.

- Pode ser obtido em: https://www.haskell.org/hugs/

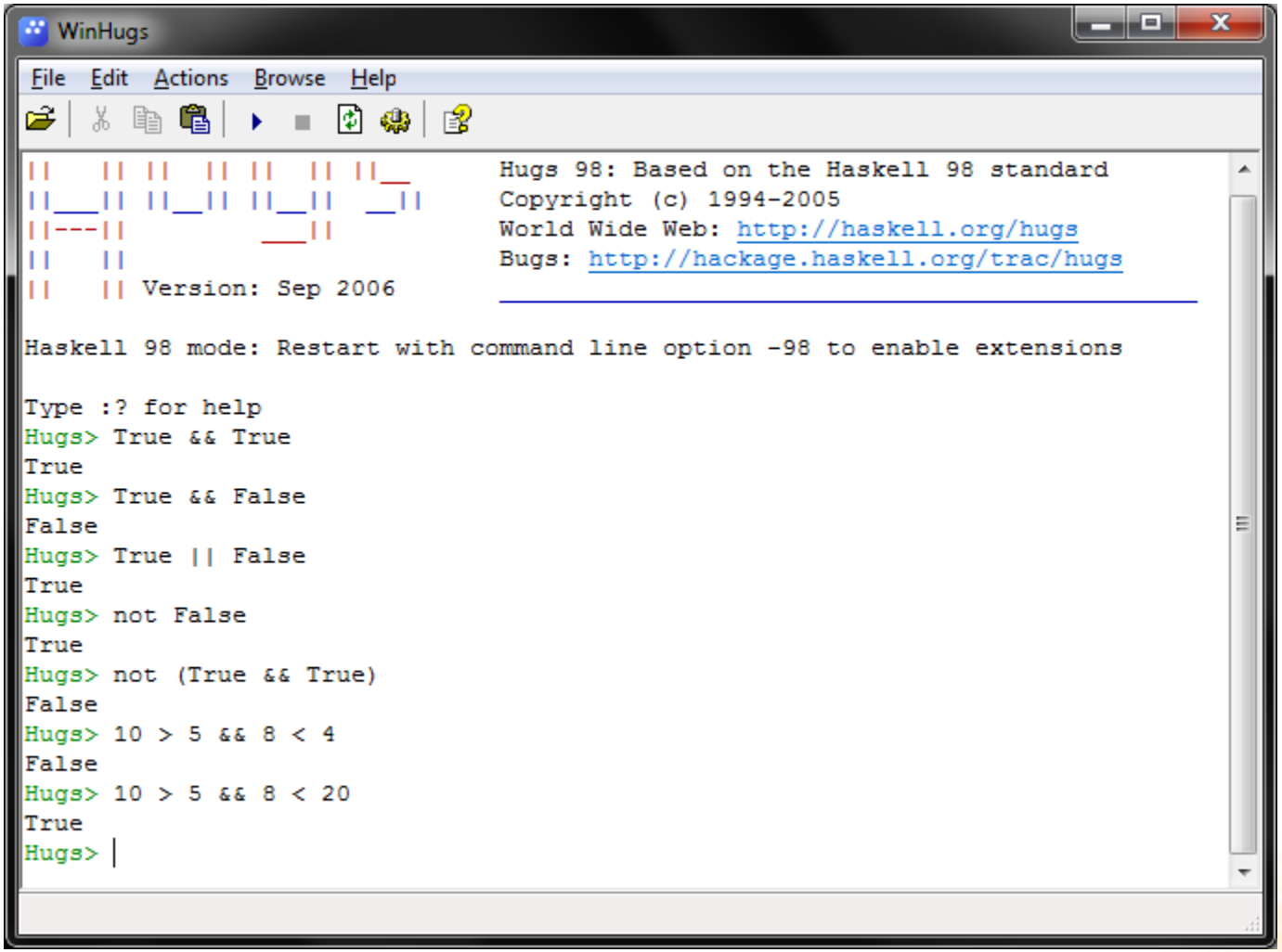
Haskell – Operações Aritméticas



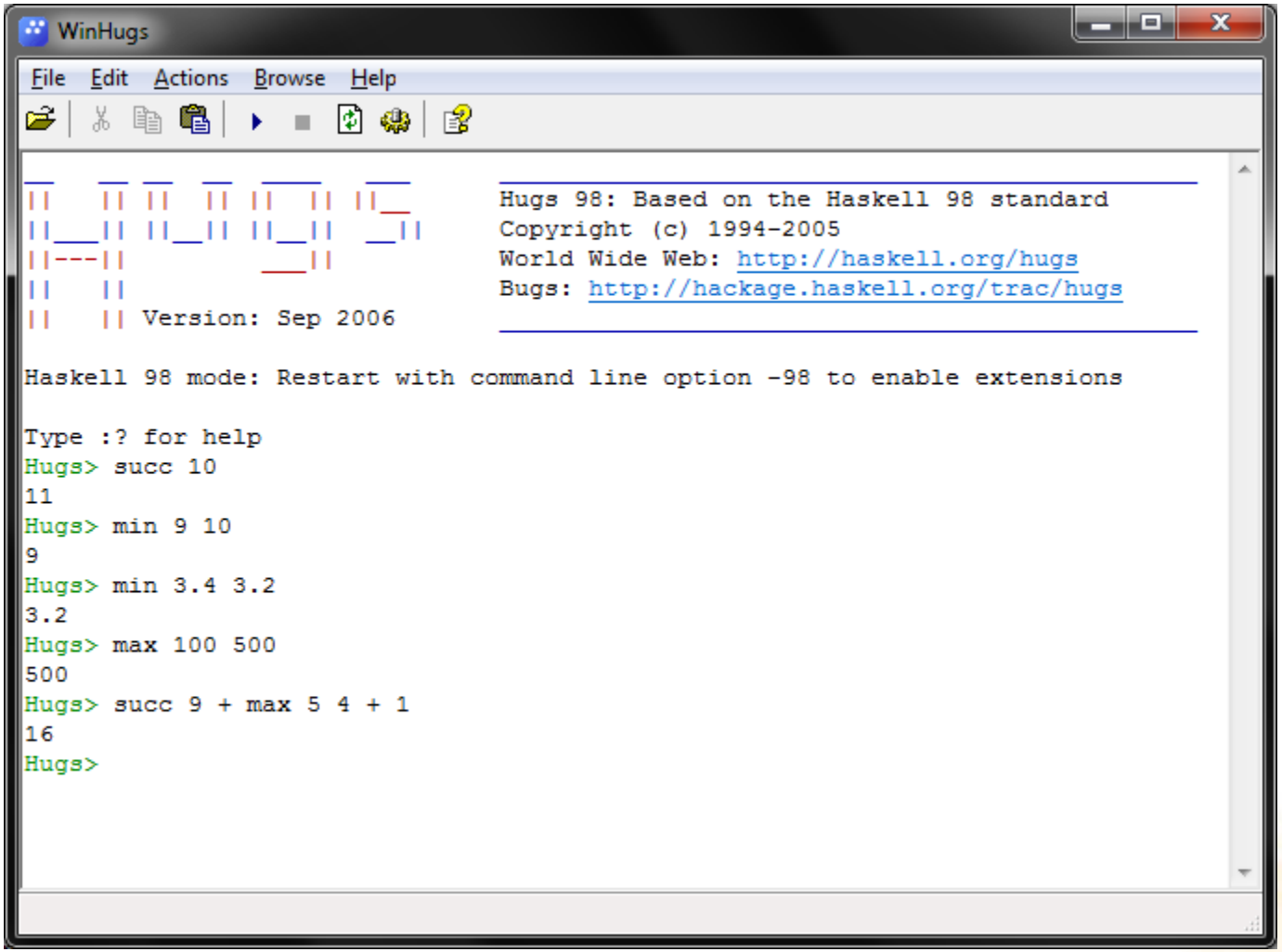
Haskell – Operadores Relacionais



Haskell – Operadores Lógicos

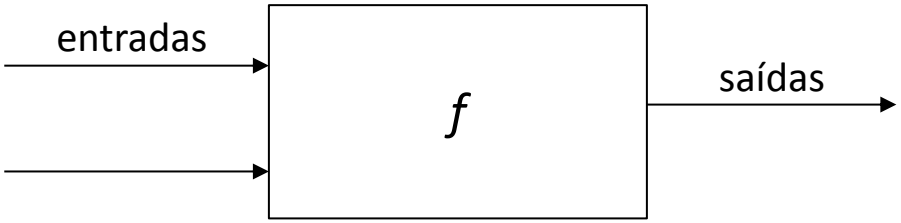


Haskell – Funções da Linguagem



Haskell – Funções

- Em Haskell, uma função pode ser representada da seguinte forma:



- A partir dos valores de entrada á obtido o valor de saída

Haskell – Funções

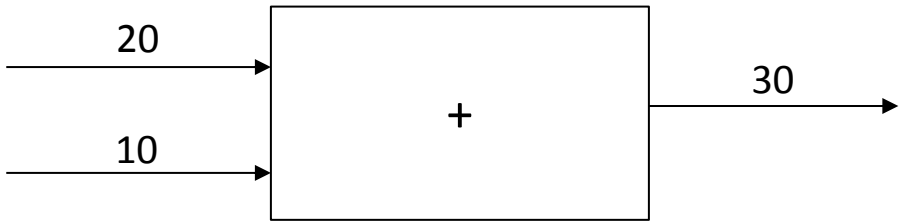
- Funções em Haskell são normalmente definidas pelo uso de equações.

- Por exemplo, a função soma pode ser escrita:

```
soma x y = x + y
```

- E chamada da seguinte maneira:

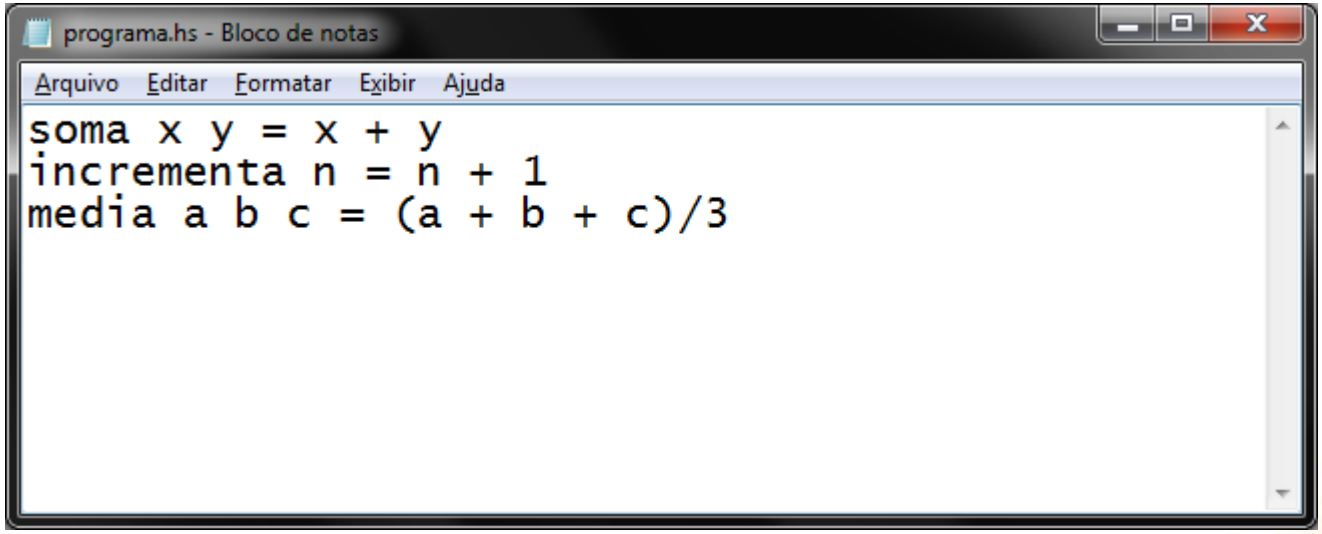
```
> soma 20 10
```



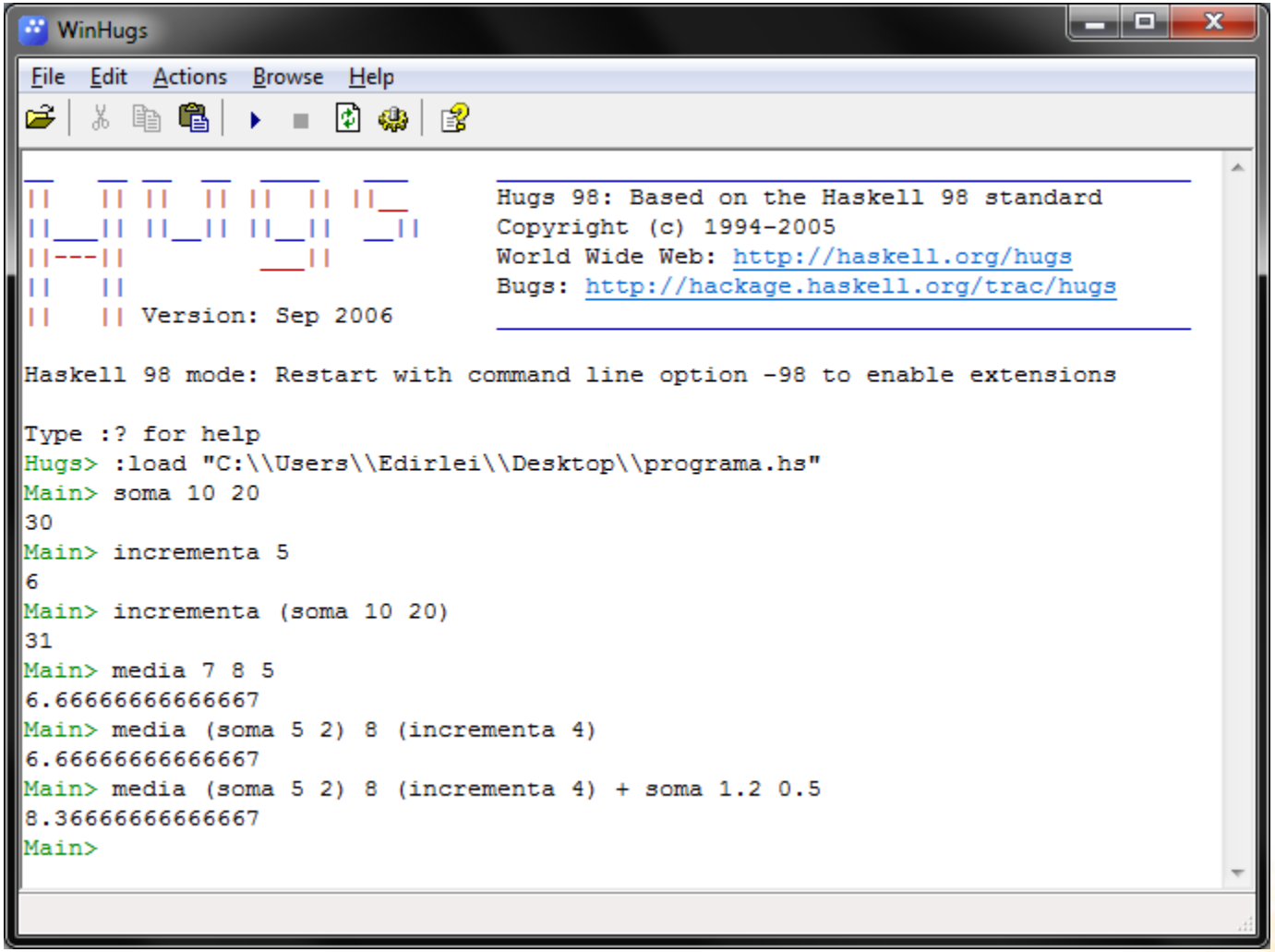
Haskell – Programas

- Os programas em Haskell podem ser implementados em um arquivo de texto salvo com a extensão .hs

- O programa pode conter a definição de várias funções:



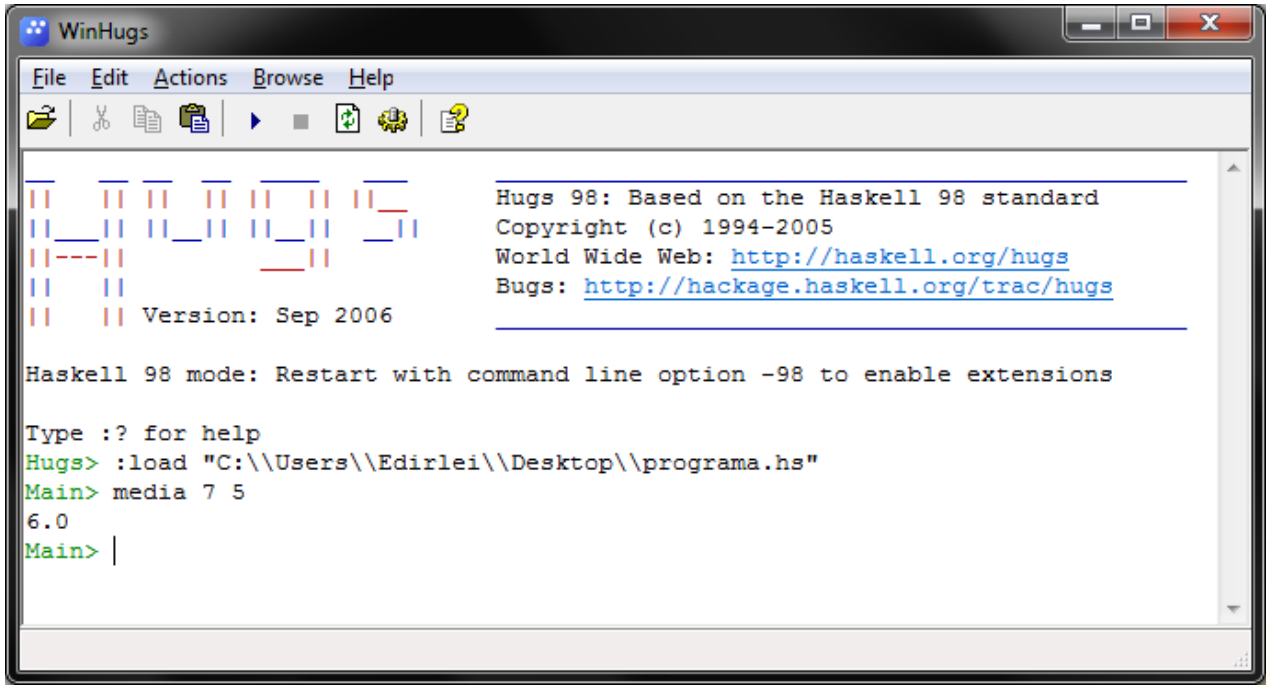
Haskell – Programas



Haskell – Programas

- Funções podem fazer chamadas a outras funções:

```
soma x y = x + y
media a b = (soma a b)/2
```



Haskell – Expressão de Seleção

- Expressão de seleção bidirecional:

```
if <condição> then <resultado1> else <resultado2>
```

- Exemplo:

```
menor x y = if x <= y then x else y
```

Haskell – Expressão de Seleção

- Expressão de seleção multidirecional (estilo *guards*):

```
nome_função par1 ... parN
       | <condição1> = <resultado1>
       | ...
       | <condiçãoN> = <resultadoN>
       | otherwise = <resultado>
```

- Exemplo:

```
menor_tres x y z
       | x <= y && x <= z = x
       | y <= z = y
       | otherwise = z
```

Haskell – Expressão de Seleção

- Expressão de seleção multidirecional (estilo *guards*) com clausula *where*:

```
nome_função par1 ... parN
       | <condição1> = <resultado1>
       | ...
       | <condiçãoN> = <resultadoN>
       | otherwise = <resultado>
       where var1 = val1
             ...
             varN = valN
```

Haskell – Expressão de Seleção

- Exemplo com clausula *where*:

```
calc_imc peso altura
    | imc <= 18.5 = "Abaixo do peso!"
    | imc <= 25.0 = "Peso ideal!"
    | imc <= 30 = "Acima do peso!"
    | otherwise = "Muito acima do peso!"
    where imc = peso / altura ^ 2
```

Haskell – Tipos de Dados

- A linguagem Haskell possui uma disciplina rigorosa de tipos de dados, sendo **fortemente tipada**. Neste caso, toda função, variável, constante tem apenas um tipo de dado, que sempre pode ser determinado.

- Embora fortemente tipada, a linguagem Haskell possui um sistema de **dedução automática de tipos** para funções cujos tipos não foram definidos.

Haskell – Tipos de Dados

Tipo	Descrição	Exemplos
Char	Caracter	'a', 'z', 'A', '3'
Bool	Booleano	True, False
Double Float	Reais	-18412.356626, 12.54, 0.0, 456.235
Integer Int	Inteiros	25, 35484, 1, -20
String	Cadeia de caracteres	“Haskell”

Haskell – Prototipação de Tipos

- Toda função definida em Haskell têm uma prototipação de tipos, que segue a sequência dos argumentos da função, sendo o último tipo o do valor de retorno da função.

```
nome_da_funcao :: Tipo_arg 1 -> Tipo_arg 2 ... Tipo_arg saida
```

- Exemplos:

```
fahrenheit_celsius :: Float -> Float
fahrenheit_celsius x = (x - 32)/ 1.8
```

```
num_par :: Int -> Bool
num_par x = if mod x 2 == 0 then True else False
```

Haskell – Recursividade

- Funções recursivas em Haskell são definidas de forma semelhante as suas definições matemáticas.

- Exemplo:

```
fatorial :: Integer -> Integer
fatorial 0 = 1
fatorial n = n * fatorial (n - 1)
```

Haskell – Listas e Tuplas

- A linguagem Haskell nos fornece dois mecanismos para a construção de dados compostos:
  - **Lista:** possibilita a união de vários elementos – todos do mesmo tipo - numa única estrutura.
  - **Tupla:** permite combinar os componentes de um dado em uma única estrutura, e os componentes podem ter tipos e propriedades distintas.

Haskell – Listas

- Em Haskell, listas vazias são representadas por []
- Lista não-vazias são representadas entre colchetes. Exemplos: [1, 2, 3, 4], ['a', 'b', 'c']
- Uma lista é composta sempre de dois segmentos: cabeça (head) e cauda (tail).
  - A cabeça da lista é sempre o primeiro elemento.

Haskell – Criação de Listas

- Listas podem ser definidas diretamente:

```
[1,2,3,4]
```

```
['a', 'b', 'c']
```

- Ou definidas através do limite inferior e superior de um conjunto conhecido, onde existe uma relação de ordem entre os elementos:

```
[1 .. 20]
```

```
[2, 4 .. 20]
```

```
['a' .. 'z']
```

Haskell – Operações em Listas

- Inserção em listas (:):

```
5:[1,2,3]
```

```
'H':"ello World!"
```

```
1:2:[3,4,5]
```

- Concatenação de listas (++):

```
[1,2,3] ++ [4,5,6]
```

- Acessar elemento pelo índice (!!):

```
[1,2,3,4,5] !! 3
```

```
"Hello World!" !! 2
```

Haskell – Operações em Listas

- Cabeça da lista (head):

```
head [5,4,3,2,1]
```

- Cauda da lista (tail):

```
tail [5,4,3,2,1]
```

- Ultimo elemento (last):

```
last [5,4,3,2,1]
```

- Lista sem o ultimo elemento (init):

```
init [5,4,3,2,1]
```

Haskell – Operações em Listas

- Tamanho da lista (length):

```
length [5,4,3,2,1]
```

- Verifica lista vazia (null):

```
null [1,2,3]
```

- Lista inversa (reverse):

```
reverse [5,4,3,2,1]
```

- Pegar parte da lista do inicio para o final (take):

```
take 3 [5,4,3,2,1]
```

Haskell – Operações em Listas

- Remove parte da lista do inicio para o final (drop):

```
drop 3 [8,4,2,1,5,6]
```

- Valor máximo da lista (maximum):

```
maximum [1,9,2,3,4]
```

- Valor minimo da lista (minimum):

```
minimum [8,4,2,1,5,6]
```

- Somatório dos valores da lista (sum):

```
sum [5,2,1,6,3,2,5,7]
```

- Produto dos valores da lista (product):

```
product [6,2,1,2]
```

Haskell – Exemplos com Listas

- Gerar palíndromo:

```
gera_palindromo n = n ++ reverse n
```

- Tamanho de uma lista:

```
tamanho :: [t] -> Int
tamanho [] = 0
tamanho (a:x) = 1 + tamanho x
```

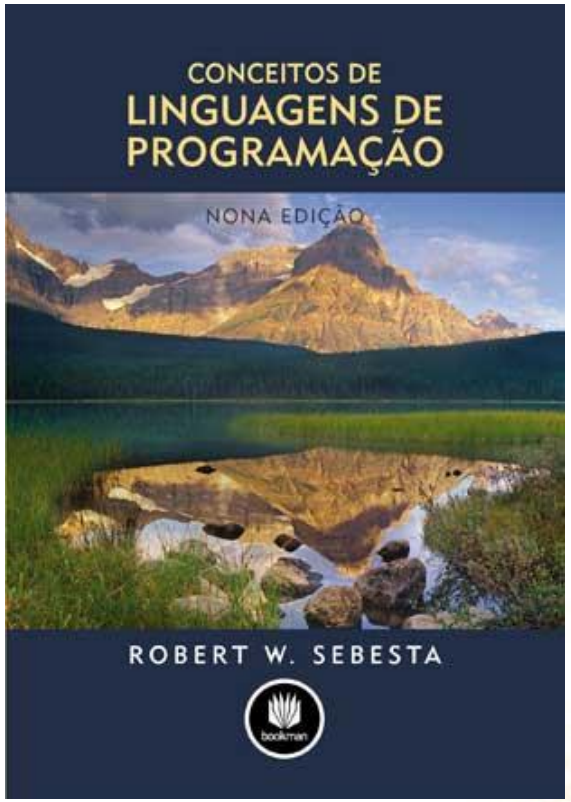
- Pertence:

```
pertence :: Eq t => t -> [t] -> Bool
pertence a [] = False
pertence a (x:z) = if (a == x) then True else pertence a z
```

Leitura Complementar

- Sebesta, Robert W. **Conceitos de Linguagens de Programação**. Editora Bookman, 2011.

- **Capítulo 15: Linguagens de Programação Funcionais**



Curso Web: http://learnyouahaskell.com/chapters