



# Linguagens de Programação

## Aula 13

**Celso Olivete Júnior**

**`olivete@fct.unesp.br`**

## Na aula passada

- ❑ **Suporte para a programação orientada a objetos**

## Na aula de hoje

# Programação Funcional

## Linguagem Haskell



# Roteiro

## □ Linguagem *Haskell*

- Funções
- Tipos básicos
- Expressões

# Linguagem Funcional

## Introdução

- ❑ 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.

# Linguagem *Haskell*

## Introdução

- ❑ *Haskell* é uma **linguagem funcional** projetada com objetivo de ser utilizada no ensino, pesquisa e construção de sistemas computacionais.
- ❑ *Haskell* deve seu nome ao matemático Haskell B. Curry, conhecido por seu trabalho em lógica combinatória e pioneiro no desenvolvimento do Cálculo Lambda → inspiração aos projetistas da maioria das LPs funcionais.

# Linguagem *Haskell*

## Exemplo

- ❑ Para somar os números inteiros de 1 a 10 podemos escrever em linguagem Java:

```
total = 0;
for (i = 1; i <= 10; i++)
    total = total + i;
```

- ❑ O método da computação é baseado em atribuição de valores à variáveis

# Linguagem *Haskell*

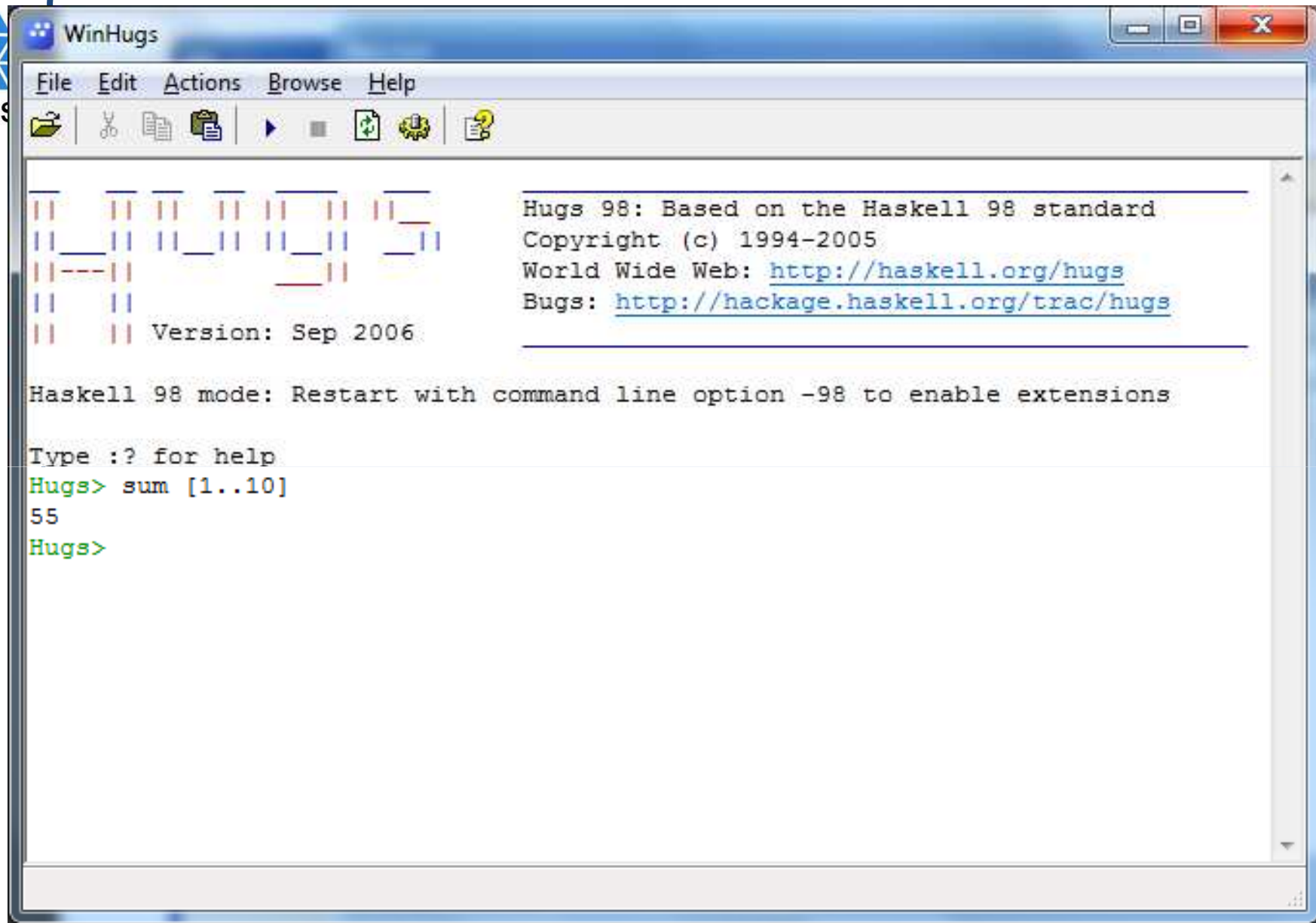
## Exemplo

- ❑ A soma dos números inteiros de 1 a 10 em linguagem *Haskell* pode ser feita como:

```
sum[1..10]
```

- ❑ O método da computação é baseado em aplicação de argumentos à funções







# Expressões em *Haskell*

```
WinHugs
File Edit Actions Browse Help
[Icons]
Hugs> 13 `div` 3
4
Hugs> 13 `mod` 5
3
Hugs> sqrt(8)
2.82842712474619
Hugs> 2+5*2
12
Hugs> 2*5+2
12
Hugs> 1+1
2
Hugs> 1*4
4
Hugs> 1
1
Hugs> 9*3
27
Hugs> 2*(3+6)
18
Hugs>
```

Hugs é uma implementação da linguagem *Haskell* que pode ser executada em PCs e sistemas Unix, incluindo Linux. Pode ser obtido gratuitamente em [www.haskell.org/hugs](http://www.haskell.org/hugs)

# Funções

- ❑ Uma função pode ser representada da seguinte maneira



- ❑ A partir dos valores de entrada é obtido o valor de saída

# Funções

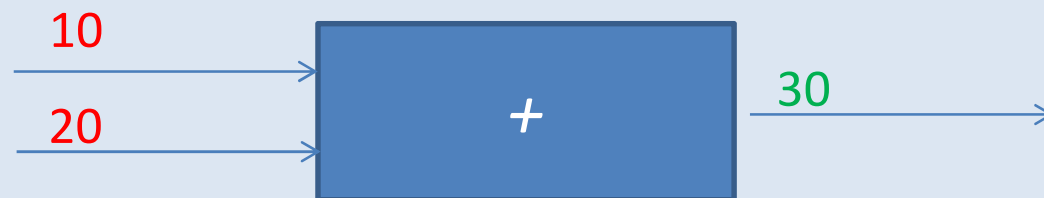
- ❑ Funções em *Haskell* são normalmente **definidas** pelo uso de **equações**. Por exemplo, a função soma pode ser escrita:

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

- ❑ Chamada da seguinte maneira:

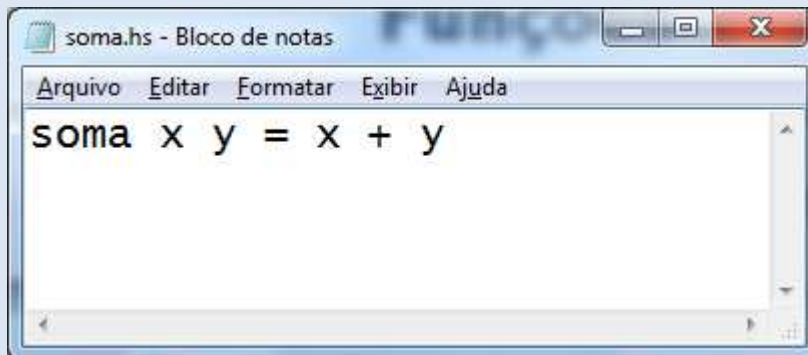
```
>soma 10 20
```

```
30
```



# Funções

- ❑ A função soma pode ser implementada em um arquivo (bloco de notas) e salvo com a extensão .hs



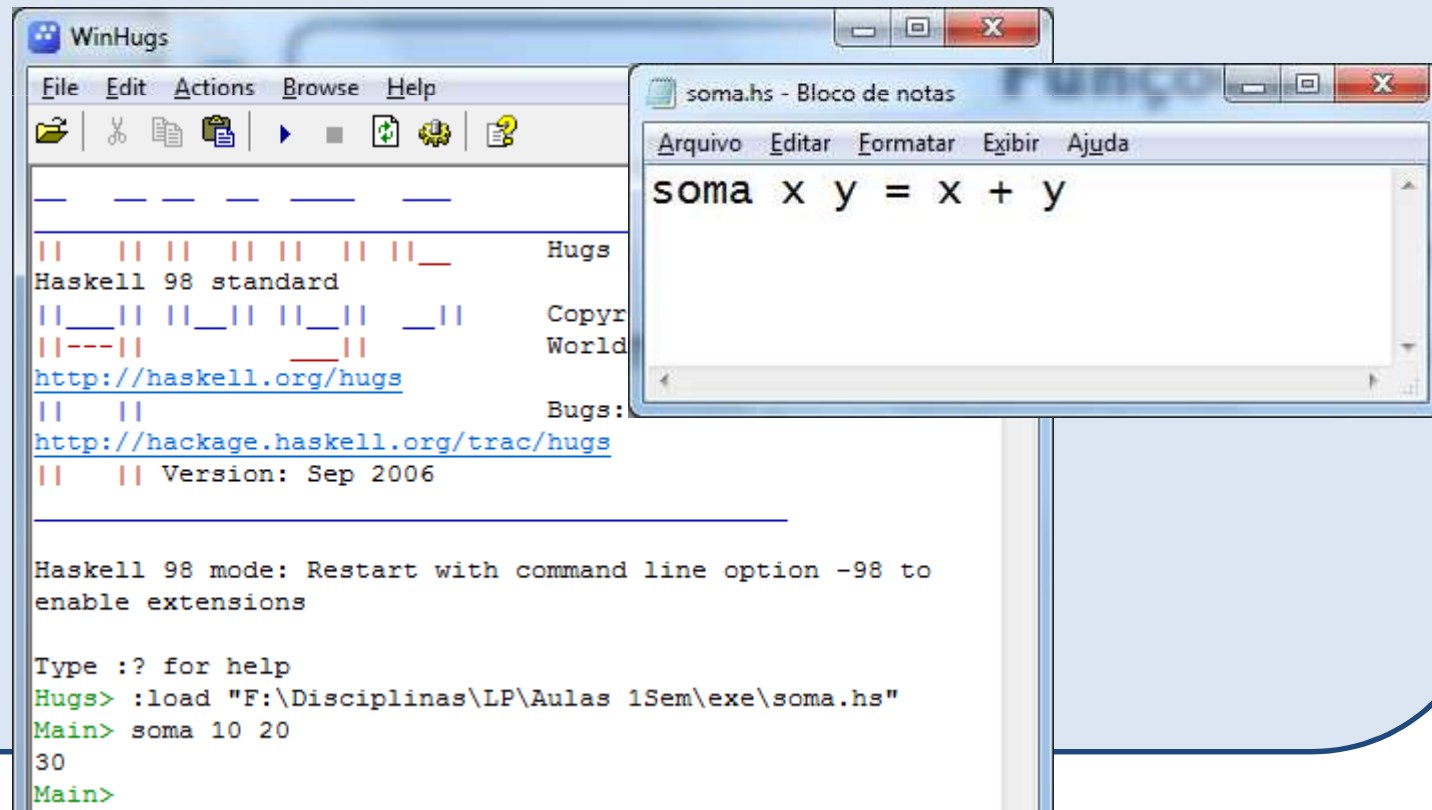
```
soma.hs - Bloco de notas
Arquivo  E_ditar  F_ormatar  E_xibir  A_juda
soma x y = x + y
```



# Funções

## □ Chamando a função soma

```
Main> soma 10 20
```

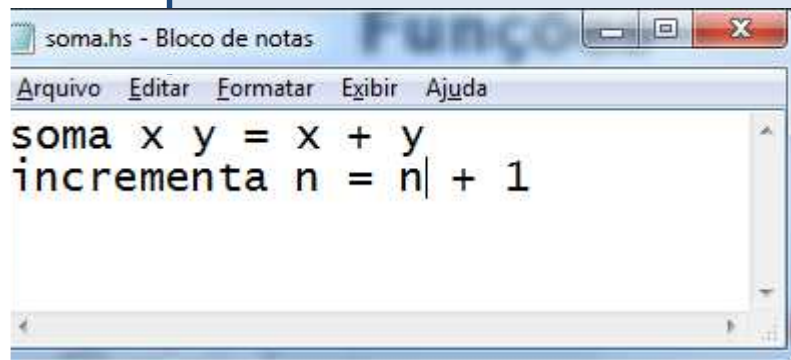


The screenshot shows two windows. The main window is WinHugs, displaying the Haskell 98 standard and the command prompt. The command prompt shows the command `:load "F:\Disciplinas\LP\Aulas 1Sem\exe\soma.hs"` and the result `30`. The Notepad window, titled "soma.hs - Bloco de notas", contains the definition of the `soma` function: `soma x y = x + y`.

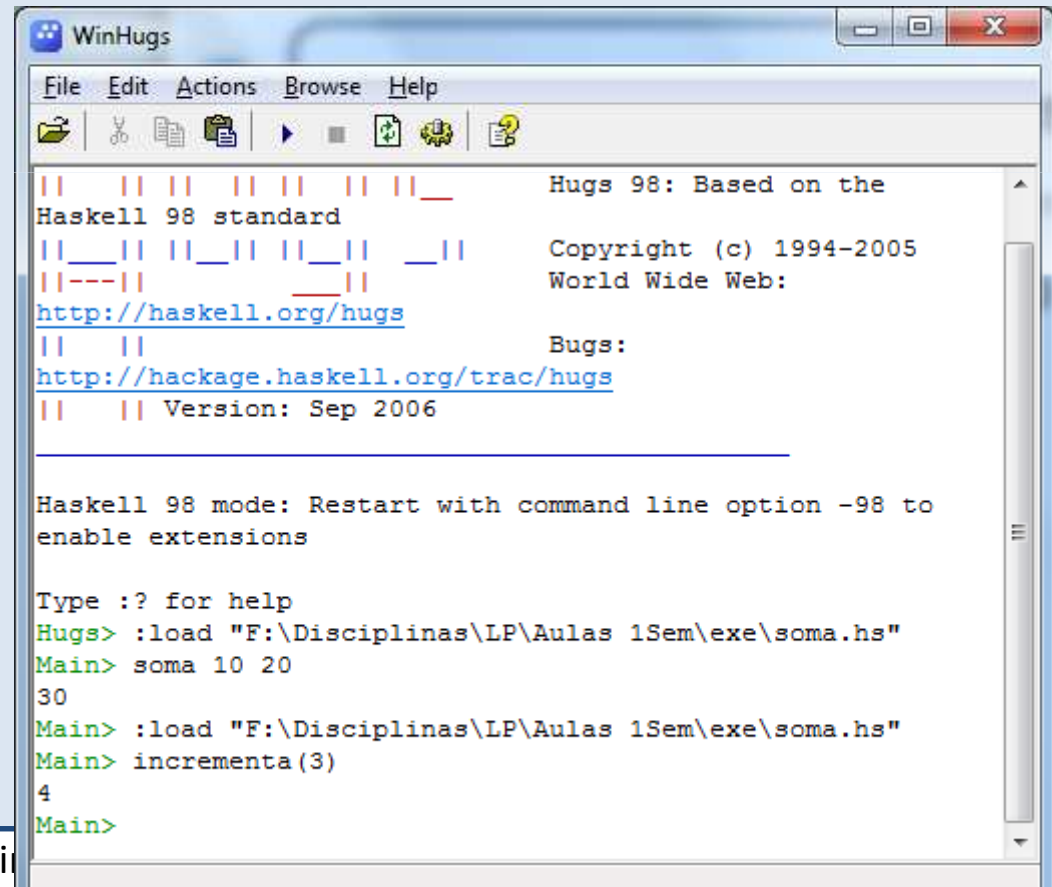
# Funções

- ❑ Incluindo mais funções no arquivo

```
incrementa n = n + 1
```



```
soma.hs - Bloco de notas
Arquivo  Editar  Formatar  Exibir  Ajuda
soma x y = x + y
incrementa n = n + 1
```



```
WinHugs
File Edit Actions Browse Help
|| || || || || || || ||_ 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

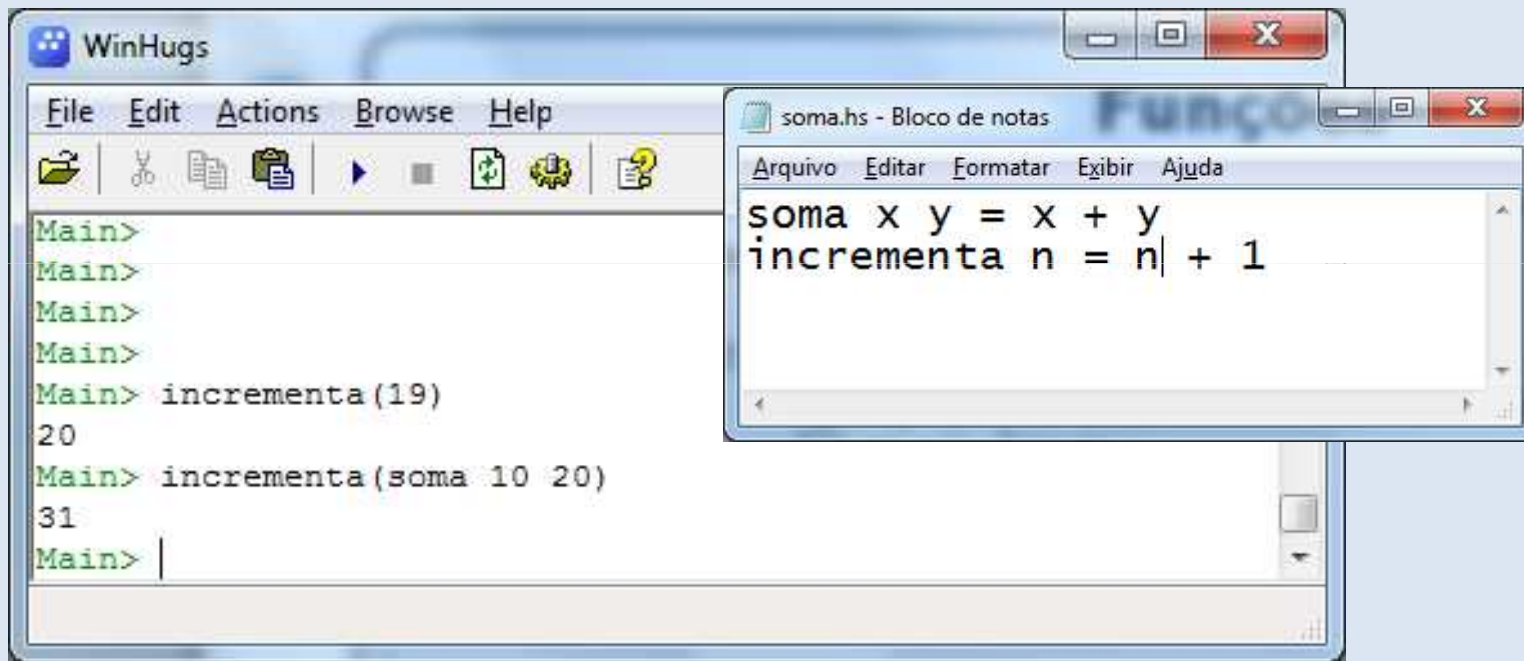
Haskell 98 mode: Restart with command line option -98 to
enable extensions

Type :? for help
Hugs> :load "F:\Disciplinas\LP\Aulas 1Sem\exe\soma.hs"
Main> soma 10 20
30
Main> :load "F:\Disciplinas\LP\Aulas 1Sem\exe\soma.hs"
Main> incrementa (3)
4
Main>
```



# Funções

- ❑ Função que chama uma função



The screenshot shows two windows. The main window is WinHugs, a Haskell interpreter, with a menu bar (File, Edit, Actions, Browse, Help) and a toolbar. The console shows the following interactions:

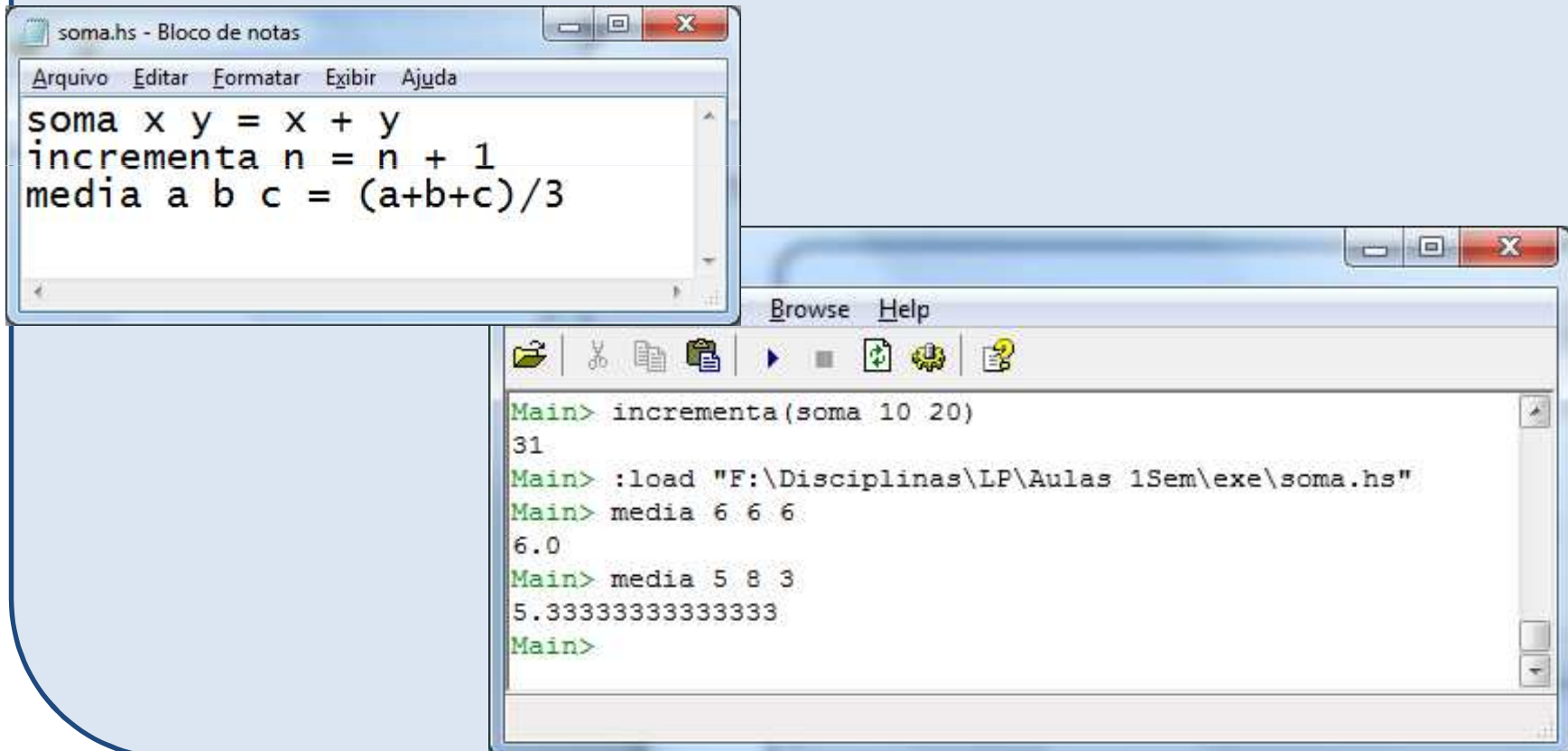
```
Main>  
Main>  
Main>  
Main>  
Main> incrementa (19)  
20  
Main> incrementa (soma 10 20)  
31  
Main> |
```

The second window is a Notepad window titled "soma.hs - Bloco de notas" with a menu bar (Arquivo, Editar, Formatar, Exibir, Ajuda). It contains the following Haskell code:

```
soma x y = x + y  
incrementa n = n | + 1
```

# Funções

- ❑ Calcular a média entre três valores



The image shows two windows. The top window is a text editor titled 'soma.hs - Bloco de notas' with a menu bar containing 'Arquivo', 'Editar', 'Formatar', 'Exibir', and 'Ajuda'. The code in the editor is:

```
soma x y = x + y
incrementa n = n + 1
media a b c = (a+b+c)/3
```

The bottom window is a REPL (Read-Eval-Print Loop) window titled 'Browse Help' with a toolbar. It shows the following interactions:

```
Main> incrementa (soma 10 20)
31
Main> :load "F:\Disciplinas\LP\Aulas 1Sem\exe\soma.hs"
Main> media 6 6 6
6.0
Main> media 5 8 3
5.333333333333333
Main>
```

# Hugs – alguns comandos importantes

Comando	Significado
:load “arq.ext”	carregar o arquivo
:reload	recarregar o arquivo atual
:edit “arq.ext”	editar o arquivo pedido
:edit	editar o arquivo atual
:type expr	mostrar o tipo de uma expressão
:?	mostrar todos os comandos
:quit	encerrar o Hugs

# Exercícios

1. Escreva uma função para calcular o dobro de um número.
2. Escreva uma função para quadruplicar um número, usando a função definida no item anterior.

# Tipos básicos

- ❑ 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.
- ❑ A partir dos tipos pré-definidos na linguagem *Haskell*, novos tipos podem ser construídos pelo programador.

# Tipos primitivos em *Haskell*

Tipo	Descrição	Exemplos
Caracter	Char	'a', 'z', 'A', '3'
Booleano	Bool	True, False
Reais	Double Float	-18412.356626, 12.54 0.0, 456.235,
Inteiros	Integer Int	4537687898, -7 3, 21475
Cadeia de caracteres	String	"Haskell"

- ❑ Para definir que uma expressão  $E$  tem o tipo  $T$  ( $E$  é do tipo  $T$ ) escreve-se:

$E :: T$

# 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 :: Tipoarg1 -> Tipoarg2 ... Tipoargsaida
```

# Prototipação de tipos

## Exemplos

Exemplo de comentário

-- Função que verifica se um número inteiro é par

`par :: Int -> Bool`

Recebe um inteiro e  
retorna *true* ou *false*

`par x = if mod x 2 == 0 then True else False`

-- Função para converter um valor Fahrenheit em Celsius

`converteFC :: Float -> Float`

`converteFC x = (x - 32) / 1.8`

*nome\_da\_funcao :: Tipo<sub>arg1</sub> -> Tipo<sub>arg2</sub> ... Tipo<sub>argsaida</sub>*



# Tipo booleano

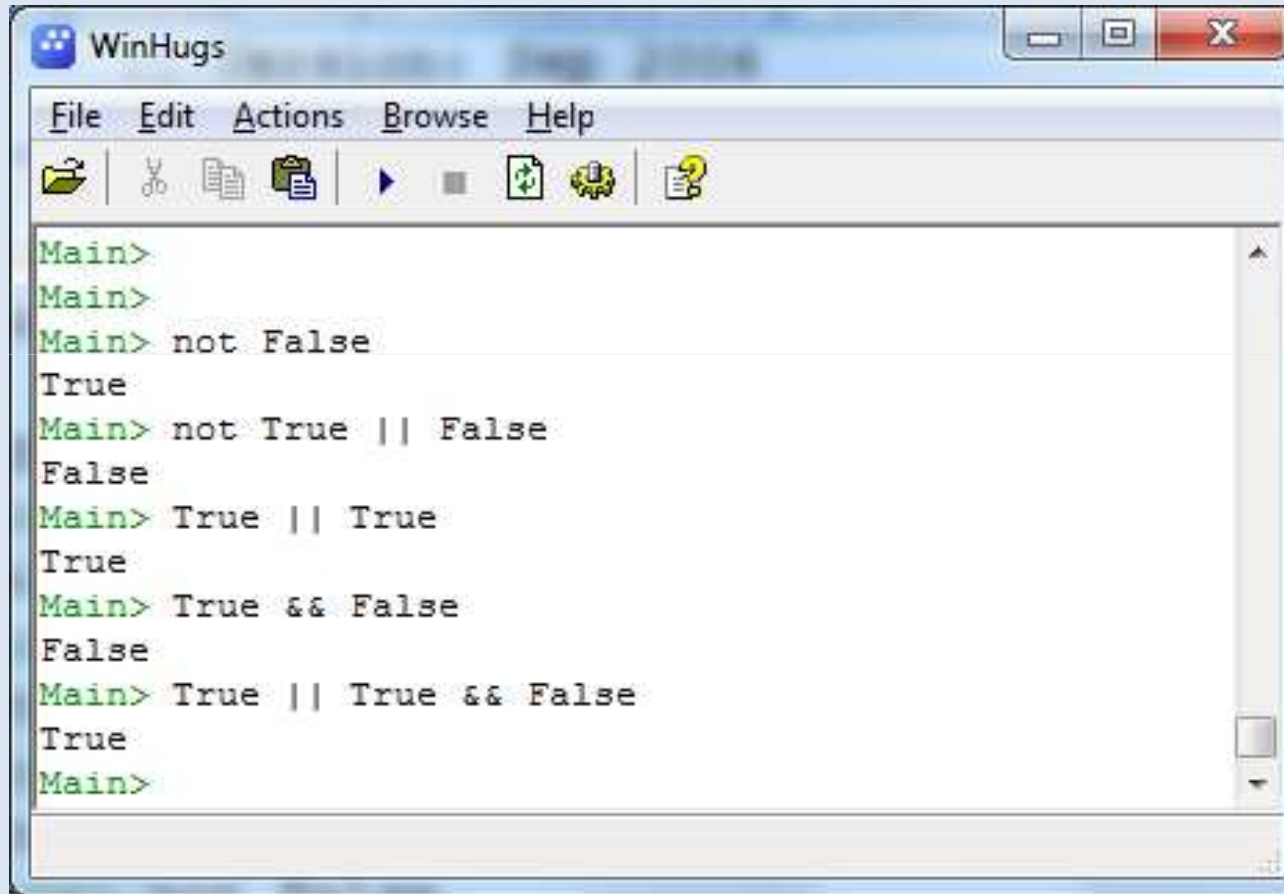
- ❑ O tipo booleano é representado pela abreviatura *Bool* e possui os valores: *True* e *False*.
- ❑ Os valores booleanos resultam de testes condicionais de comparações. As operações definidas sobre booleanos são: e (&&), ou (||) e negação (not).
- ❑ As prototipações são dadas por:

```
&& :: Bool -> Bool -> Bool
```

```
|| :: Bool -> Bool -> Bool
```

```
not :: Bool -> Bool
```

# Combinações entre *True* e *False*



```
WinHugs
File Edit Actions Browse Help
Main>
Main>
Main> not False
True
Main> not True || False
False
Main> True || True
True
Main> True && False
False
Main> True || True && False
True
Main>
```

# Tipo inteiro

- ❑ Os inteiros são referenciados por `Int`, o domínio matemático dos inteiros (até 2147483647).

Operador	Descrição	Exemplos
<code>+</code>	Soma entre dois inteiros	$1 + 2 = 3$
<code>-</code>	Subtração entre dois inteiros	$(-) 5 - 4 = 1$
<code>*</code>	Multiplicação entre dois inteiros	$2 * 2 = 4$
<code>^</code>	Exponenciação	$3.2^3 = 32.768$
<code>div</code>	Parte inteira da divisão	<code>div 20 3 = 6</code>
<code>mod</code>	Resto da divisão	<code>mod 20 3 = 2</code>
<code>abs</code>	Valor absoluto de um inteiro	<code>abs (-89) = 89</code>
<code>negate</code>	Inverte o sinal de um inteiro	<code>negate (13) = -13</code>

# Expressões

- ❑ A definição de uma função deve obedecer à sintaxe seguinte:

```
nome_da_funcao :: Tipoarg1 -> Tipoarg2 ... Tipoargsaida  
nome_funcao arg1 ... argN = <expressão_resultado>
```

- ❑ Sendo `tipo1, ..., tipoN` os tipos dos parâmetros de entrada da função.
- ❑ Considere uma função que retorna o menor entre dois números:

```
menor 5 4 = 4
```

```
menor 0 0 = 0
```

```
menor 3 7 = 3
```

## Exemplo

- ❑ A declaração dos tipos dos valores de entrada e saída da função é definida por:

```
menor :: Int -> Int -> Int
```

- ❑ Estratégia para a definição do problema: uso de uma expressão de seleção

```
menor x y =
```

```
  se x <= y então o resultado da expressão é x
```

```
  senão o resultado da expressão é y
```

# Expressão de seleção

- ❑ Sintaxe de uma expressão de seleção bidirecional:

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

- ❑ Função Menor:

```
menor :: Int -> Int -> Int  
menor x y = if x <= y then x  
           else y
```

# Expressão de seleção

- ❑ Sintaxe de uma expressão de seleção multidirecional (estilo *guardas*):

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

- ❑ As *guardas* permitem estabelecer uma distinção entre casos diferentes da definição de uma função.

# Expressão de seleção

## Exemplo

- ❑ Função para retornar o maior entre três números:

```
maxTres :: Int -> Int -> Int -> Int
```

```
maxTres x y z
```

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

```
| y >= z              = y
```

```
| otherwise           = z
```

- ❑ Avaliação de uma aplicação da função maxTres

```
> maxTres 4 3 2
```

```
?? 4 >= 3 && 4 >= 2
```

```
?? True && True
```

```
?? True
```

```
4
```



# Expressão de seleção

## Exemplo

- ❑ Avaliação de uma aplicação da função maxTres

```
> maxTres 6 (4+3) 5
?? 6 >= (4+3) && 6 >= 5
?? 6 >= 7 && 6 >= 5
?? False && True
?? False
?? 7 >= 5
?? True
7
```

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

- ❑ Neste exemplo avaliamos inicialmente a primeira condição,  $6 \geq (4 + 3) \ \&\& \ 6 \geq 5$  que resultou em False; e em seguida avaliamos  $7 \geq 5$  que é verdadeira. Assim, o resultado é 7.

## Funções do módulo `Prelude`

- `even` - verifica se um valor dado é par
- `odd` - verifica se um valor dado é ímpar
- `rem` - resto da divisão inteira
- `mod` - resto da divisão inteira
- `ceiling` - arredondamento para cima
- `floor` - arredondamento para baixo
- `round` - arredondamento para cima e para baixo
- `truncate` - parte inteira do número
- `fromIntegral` converte um inteiro em real.

# Funções do módulo Prelude (cont...)

- sin - seno de ângulo em radianos
- cos - cosseno
- tan - tangente
- asin - arco seno
- acos - arco cosseno
- atan - arco tangente
- abs - valor absoluto
- sqrt - raiz quadrada, valor positivo apenas
- exp - exponencial base e
- log - logaritmo natural (base e)
- logBase - logaritmo na base dada
- min - menor de 2 objetos dados
- max - maior de 2 objetos dados

# Exercícios

1. Mostre manipulações sobre todos os operadores dos inteiros vistos anteriormente
2. Escreva uma função que recebe um valor numérico e devolva "o número é primo" ou "o número não é primo"
3. Escreva uma função que recebe um valor numérico e devolva o valor 1 se o valor for maior que zero, -1 se for negativo, 0 se for zero.
4. Faça uma função que, dados três parâmetros de entrada, se o primeiro for um asterisco, os outros dois serão multiplicados; se for uma barra, o segundo deve ser dividido pelo terceiro; se não for nenhum dos dois, imprima uma mensagem de erro.
5. A sequência de *Fibonacci* é dada pela seguinte série:

0 1 1 2 3 5 8 13 ...

Construa uma função para retornar o n-ésimo termo da sequência. Exemplo:

```
main> fibonacci 6
```

8