Introdução

João Marcelo Uchôa de Alencar

Universidade Federal do Ceará - Quixadá

15 de Maio de 2017

O Professor

- João Marcelo Uchôa de Alencar
- E-mail: joao.marcelo@ufc.br
- Sala no Bloco II
- Disponibilidade para tirar dúvidas: Quinta e Sexta, a partir das 14:00

O Conteúdo

- O conteúdo programático está cadastrado no SI3
- Os dias previstos para as avaliações também estão lá
- **O SI3 é o ambiente oficial**
- **https://progfuncional20171.slack.com/**
- O SIPPA será atualizado com frequência menor, os dados completos somente estarão lá ao fim da disciplina
- Se você quiser conversar com o professor, use o e-mail ou o SI3.

Avaliações

- Duas provas escritas
- Um trabalho de implementação
- A nota final é a média das duas melhores avaliações
- A frequência é importante, ausências em número maior do que o permitido levarão a reprovação

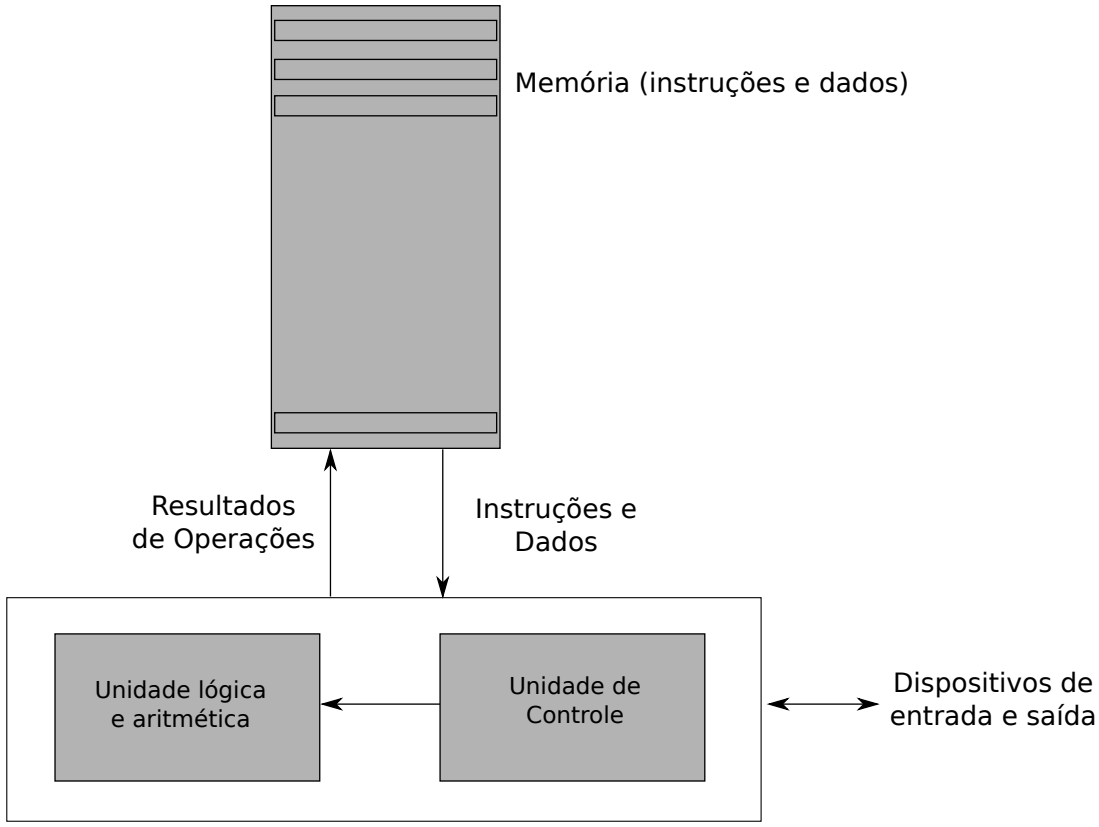
Bibliografia

- Haskell: Uma Abordagem Prática. Cláudio César de Sá e Márcio Ferreira da Silva. Editora Novatec. (tem na biblioteca).
- Real World Haskell. Bryan O'Sullivan, Don Stewart e John Goerzen. Editora O'Reilly. (disponível gratuito na Internet: `http://book.realworldhaskell.org/read/`.

Linguagens Imperativas

- **Variáveis** modelando células de memória.
- Ciclo de obtenção e execução.
- Existe um **contador de programa** que indica qual a próxima instrução a ser executada.
- **Operandos** representando instruções ou sequências de instruções da arquitetura.

Linguagens Imperativas



O que é Programação Funcional?

- Programação Funcional é um método de construção de programas que enfatiza **funções** e suas aplicações ao invés de comandos e sua execução.
- Programação Funcional usa uma notação matemática simples que permite descrever problemas de maneira **clara e concisa**.
- Programação Funcional tem uma base matemática simples que suporta racionalizar sobre as **propriedades** de um programa.
- **Haskell** é uma linguagem de programação funcional com tipos de dados bem definidos.

O Paradigma Funcional

- Vantagens
  - Construção Eficiente de Programas.
  - Prova de Propriedades.
  - Transformação de Programas.
  - Concorrência Natural.
- Desvantagens
  - O mundo Não é Funcional?
  - Implementações Ineficientes?
  - Mecanismos Primitivos de Entrade e Saída.

O que é uma Função Matemática?

- Domínio.
- Imagem.
- Relação para elementos do domínio em direção à elementos da imagem.

Características do Haskell

- Haskell também é conhecida como uma *calculadora funcional*.
- Estilo declarativo.
- Polimorfismo.
- Fácil aprendizado.
- Baseada no Lambda Cálculo.

Funções e Tipos

O conceito básico de Haskell é a função.

```
f :: X -> Y
```

f é uma função que recebe argumentos do **tipo** X e retorna resultados do **tipo** Y.

```
sin :: Float -> Float
age :: Person -> Int
add :: (Integer, Integer) -> Integer
logBase :: Float -> (Float -> Float)
```

Na matemática escrevemos $f(x)$. Em Haskell, podemos dispensar os parênteses: $f\ x$. Apenas um espaço separa a função do argumento.

A função também é um tipo!

A aplicação de funções é **associativa a esquerda** e tem **maior prioridade** do que as outras operações.

Composição Funcional

Considere duas funções:

```
f :: Y -> Z
g :: X -> Y
```

Podemos combiná-las em uma única:

```
f . g :: X -> Z
```

Primeiro g é aplicado para um argumento do tipo X, dando um resultado do tipo Y, então f é aplicada, dando um resultado final do tipo Z.

```
(f . g) x = f (g x)
```

O Poder de Haskell

- Somente com tipos e composição funcional, podemos construir praticamente todo tipo de programa.
- Sim, utilizamos recursão.
- Podemos utilizar estruturas de controle e repetição quando necessário, mas mesmo assim elas tem uma estrutura *funcional*.

Ambiente de Programação

- Instalação no Ubuntu 16.04:
  - Devemos instalar o *Glasgow Haskell Compiler*.
  - *sudo apt-get install ghc*
  - Existe um compilador, como em C, *ghc*.
  - Existe um interpretador iterativo, como em Python, *ghci*.
  - Existe um interpretador em lote, como em Python, *runghc*.
  - A sintaxe varia um pouco de um para outro.
- Como IDE, podemos utilizar qualquer qualquer editor de texto. Sugestão: gedit.
- Por enquanto, vamos ficar no interpretador iterativo.

```
jmhal@joao:~$ ghci
GHCi, version 7.10.3: http://www.haskell.org/ghc/  :? for help
Prelude>
```

O **Prelude** é como se fosse o stdio.h do C ou a biblioteca padrão do Python.

Usando o ghci como uma calculadora

```
:set prompt "ghci> "
ghci>
ghci> 2 + 2
4
ghci> 31337 * 101
3165037
ghci> 7.0 / 2.0
3.5
ghci> 2 + 2
4
ghci> (+) 2 2
4
ghci> 2 + -3
<interactive>:1:0:
    precedence parsing error
        cannot mix `(+)' [infixl 6] and prefix `-' [infixl 6] in the same infix
ghci> 2 + (-3)
-1
```

Valores booleanos, operadores e comparações

```
ghci> True && False
False
ghci> False || True
True
ghci> True && 1
<interactive>:1:8:
    No instance for (Num Bool)
      arising from the literal ‘1’ at <interactive>:1:8
    Possible fix: add an instance declaration for (Num Bool)
    In the second argument of ‘(&&)’, namely ‘1’
    In the expression: True && 1
    In the definition of ‘it’: it = True && 1
ghci> 1 == 1
True
ghci> 2 < 3
True
ghci> 4 >= 3.99
True
ghci> 2 /= 3
True
ghci> not True
False
```

Listas

```
ghci>[1,2,3]
[1,2,3]
ghci>[]
[]
ghci>["ufc", "quixada", "computacao"]
["ufc","quixada","computacao"]
ghci>[True, False, "palavra"]
<interactive>:6:15:
    Couldn't match expected type `Bool' with actual type `[Char]'
    In the expression: "palavra"
    In the expression: [True, False, "palavra"]
    In an equation for `it': it = [True, False, "palavra"]
ghci>[1..10]
[1,2,3,4,5,6,7,8,9,10]
ghci> [1.0,1.25..2.0]
[1.0,1.25,1.5,1.75,2.0]
ghci> [1,4..15]
[1,4,7,10,13]
ghci> [10,9..1]
[10,9,8,7,6,5,4,3,2,1]
ghci>[1,2] ++ [3,4]
[1,2,3,4]
ghci> 1 : [2,3]
[1,2,3]
```

Strings e Caracteres

```
ghci>"Campus de Quixadá."
"Campus de Quixad\225."
ghci>"Campus de Quixada."
"Campus de Quixada."
ghci>putStrLn "Aqui está uma nova linha --> \n <-- Viram?"
Aqui está uma nova linha -->
 <-- Viram?
ghci>'a'
'a'
ghci>let a = [ 'q', 'u', 'e', 'r', 'o',' ', 'p', 'r', 'o', 'g', 'r', 'a', 'm',
ghci>a
"quero programar"
ghci>a == "quero programar"
True
ghci> "" == []
True
ghci>'a':"bc"
"abc"
```

Verificando os Tipos das Expressões

- Assim como as funções, toda expressão também tem um **Tipo** associado.
- Os tipos em Haskell desempenham papel importante na verificação e composição de funções.
- Por padrão, os nomes de tipos começam com maiúsculas, enquanto os nomes de variáveis começam por minúsculas.

```
ghci> :set +t
ghci> 'c'
'c'
it :: Char
ghci> "ufc"
"foo"
it :: [Char]
ghci>:unset +t
ghci>:type "Joao"
"Joao" :: [Char]
```

A configuração $+t$ faz com que o *ghci* emita o tipo da expressão após a avaliação. A variável *it* armazena o resultado da última expressão avaliada.

Olá Mundo

```
ghci>putStrLn "Olá Mundo"
Olá Mundo
<CTRL-D>
Leaving GHCi.
$ cat OlaMundo.hs
main = putStrLn "Olá Mundo"
$ runghc OlaMundo
Olá Mundo
$ ghc OlaMundo.hs -o OlaMundo
[1 of 1] Compiling Main             ( OlaMundo.hs, OlaMundo.o )
Linking OlaMundo ...
$ ./OlaMundo
Olá Mundo
```

Exercício

Descubra o tipo de cada uma das expressões abaixo no *ghci*:

1. 5 + 8
2. 3 * 5 + 8
3. 2 + 4
4. (+) 2 4
5. sqrt 16
6. succ 6
7. succ 7
8. pred 9
9. pred 8
10. sin (pi / 2)
11. truncate pi
12. round 3.5
13. round 3.4
14. floor 3.7
15. ceiling 3.3