UFSC / CTC / INE
Disciplina: Paradigmas de Programação
(Programação Funcional)

Curso de Ciências da Computação: INE5416

Prof. Dr. João Dovicchi*

1 Aula Prática 1 - Unix / Haskell

Nesta aula vamos compreender alguns conceitos do ambiente a ser utilizado nas aulas de Programação Funcional. Em primeiro lugar, vamos compreender alguns princípios básicos do sistema Unix e, posteriormente, alguns aplicativos para interpretar compilar programas em Haskell e compilar programas em C.

Embora existam diversas interfaces de programação ou *Integrated Development Environment* (IDE), é fundamental que o aluno use um editor simples, uma vez que ele deve compreender aspectos fundamentais da estrutura da linguagem.

1.1 O sistema Unix

O sistema Unix, diferentemente do que muitos acreditam, tem todas as ferramentas de desenvolvimento e interface gráfica (anterior ao sistema de janelas do DOS, hoje conhecido como Microsoft Windows). Muito embora existam diversos GUI (*Graphical User Interface*), o conceito do Unix é bem diferente de outros sistemas operacionais. Enquanto o MS Windows é um sistema operado por disco (DOS), os sistemas Unix (FreeBSD, Sun OS, AIX, HPUX etc.) e os *Unices-like* (Linux, BeOS, Hurd, Inferno etc.) são baseados no paradigma preemptivo de paginação de memória.

*`http://www.inf.ufsc.br/~dovicchi --- dovicchi@inf.ufsc.br`

O aluno deve saber como utilizar um terminal para se conectar a um sistema Unix disponível. Para as aulas práticas o aluno deve familiarizar-se com o ambiente do terminal do Unix (*Shell*), bem como com os comandos básicos para criar e editar arquivos de programa, manipular arquivos e diretórios, compliar programas em C e Haskell e executá-los.

1.1.1 A linha de comandos (bash)

O Bourne Again Shell (bash) é um dos mais poderosos interpretadores de comandos do terminal Unix. Geralmente, sistemas como Linux, FreeBSD, SunOS e outros implementam este tipo de interpretador por *default*.

Normalmente, o sinal de pronto (*prompt*) do bash é algo semelhante a:

```
bash-2.05b$
```

ele indica que o sistema está pronto, esperando a entrada de uma linha de comando. Nestes roteiros, o sinal de pronto será representado pelo sinal "$". No caso da máquina haskell, o *prompt* tem o nome do usuário e da máquina seguido do diretório corrente:

```
[jcd@haskell ~]$
```

onde o sinal de til representa o diretório `$HOME` (digite "pwd" para ver o diretório corrente).

Uma das melhores facilidades do bash é o fato de ele completar a linha de comandos quando usamos a tecla `<tab>`. Por exemplo, ao digitarmos:

```
$ a<tab><tab>
```

o sistema retornará todos os comandos disponíveis no `$PATH` do usuário que comecem com a letra "a".

Para as atividades práticas o aluno deve conhecer alguns comandos básicos para lidar com arquivos e executar algumas tarefas. a tabela 1 apresenta alguns comandos básicos.

Um fato importante é que o Unix sempre considera que o comando deve ser executado. Raramente, o sistema pede a confirmação. Alguns comandos têm uma opção para pedir confirmação (p. ex. `rm -i` sempre pedirá confirmação antes de remover o arquivo).

Comando	**Descrição** (use `man <comando>` para mais informações)
cd	Muda diretório.
cp	Copia arquivo.
ls	Lista os arquivos do diretório corrente.
man	exibe a página de manual.
mkdir	cria um diretório.
mv	Move (ou renomeia) arquivo.
pwd	exibe o path do diretório corrente.
rm	remove arquivo.

Tabela 1: Comandos básicos do Unix

1.1.2 Edição arquivos

Existem vários editores de texto no Unix. O mais usado, certamente é o Vi. Outros editores como o pico, mcedit, joe e emacs podem ser usados. O aluno deve escolher o que melhor lhe servir e utilizá-lo na edição de seus textos de programas.

Embora o Vi seja um tanto diferente de outros editores, ele é um dos mais poderosos e, além disso, é universalmente encontrado nas máquinas Unix. No entanto, os outros são mais amigáveis. O pico é o editor de textos do pine e o mcedit é o editor do "midnight commander". O joe é um tipo de wordstar.

Entretanto, nada impede que os arquivos sejam editados no seu editor de preferência (inclusive em outro sistema) e depois copiado para a máquina onde serão feitas as práticas, desde que estes estejam no formato de texto puro. No entanto, é importante lembrar que os sistemas Unix não usam dois caracteres de fim de linha, mas apenas o caractere ASCII "10" (`\x0A` ou `LF`). Assim, ao tentarem abrir um arquivo texto editado no Unix em um sistema DOS, o arquivo poderá não estar corretamente formatado. O DOS utiliza dois caracteres ASCII como final de linha "10 13" (`\x0A\x0D` ou `LF CR`).

2 Roteiro 1

- O aluno deve acessar e aplicar, na prática, o que foi aprendido sobre o sistema operacional Unix.
  - Uso do comando `ls` para listar arquivos;
  - Criar, renomear e apagar diretórios; e

  - Aplicar os comandos `cat, echo, tail, head` e `more`.

- Usar um editor de texto (vi, pico, mcedit etc.) para editar um texto e selecionar um de sua preferência.

3 Roteiro 2

1. Crie um diretório "haskell"em seu `$HOME` e entre nele. No shell do Unix faça:

```
$ mkdir haskell
$ cd haskell
```

2. Crie um arquivo "Exemplo.hs", usando seu editor de preferência [1], com o código:

Listagem 1: Programa Pratica02.hs

```
module Exemplo (
    inc
    ) where

main = print ()

inc :: Integer -> Integer
inc n = n + 1
```

3. Para entender a tipagem de dados vamos usar o ghci que é um interpretador da linguagem Haskell. Para isso, digite "ghci" na linha de comando e ENTER:

```
$ ghci

   ___         ___ _
  / _ \ /\  /\/ __(_)
 / /_\// /_/ / /  | |      GHC Interactive, version 6.6.1, for Haskell 98.
/ /_\\/ __  / /___| |      http://www.haskell.org/ghc/
\____/\/ /_/\____/|_|      Type :? for help.

Loading package base ... linking ... done.
Prelude>
```

[1] Você pode criar este arquivo no seu computador e enviá-lo para o seu diretório $HOME/haskell/ ou editá-lo no próprio diretório usando o vi ou seu editor de preferência.

Ao carregar o GHCi, o programa lhe apresenta um novo shell com uma linha de comando. Digite `:?` para ver os comandos possíveis.

3.1 Verificando tipos de dados

4. Carrege o módulo "Exemplo.hs":

```
Prelude> :l Exemplo
Exemplo*>
```

5. O módulo "Exemplo" tem apenas uma função, que se chama `inc` que pode ser expressa, matematicamente, como:

$$f(x) = x + 1 | x \in \mathbb{Z}$$

ou seja, a função de x incrementa x tal que x pertencente ao conjunto dos números inteiros.

1. Na linha de comando do GHCi:
   (a) Digite "inc 2", analise o resultado;
   (b) Digite "inc 2.5", analise o resultado;
   (c) Digite ":type inc", explique o que aconteceu.
2. Edite seu arquivo "Exemplo.hs" e adicione mais uma função: A função a ser acrescentada será `incInt`. Ao acrescentar mais uma função, temos que explicitar que o módulo Exemplo exporta esta função (ver listagem 2).

Listagem 2: Programa Exemplo.hs

```
module Exemplo (
         inc, incInt
         ) where

main = print ()

inc :: Integer -> Integer
inc n = n + 1

incInt :: Int -> Int
incInt n = n + 1
```

Se você editar dentro do GHCi, basta digitar ":reload" para recarregar o módulo. Caso contrário, carreque o GHCi e, novamente, carregue o módulo.

3. Na linha de comando do GCHi, digite:

   (a) inc 1000000000000

   (b) incInt 1000000000000

   (c) incInt 100

   Explique o resultado.

4. Vamos, então, criar uma outra função `incReal` que possa aceitar quaisquer números. Neste caso, nossa função passa a operar no domínio de todos os números reais:

   $$f(x) = x + 1 | x \in \Re$$

   para isso, inclua (na definição do módulo) a função `incReal` e defina-a como:

   ```
   incReal :: Double -> Double
   incReal n = n + 1.0
   ```

   Uma outra forma de instanciar a função é:

   ```
   incReal :: Fractional a => a -> a
   incReal n = n + 1.0
   ```

5. Teste a função "incReal" com vários tipos de números.

6. Agora voce deve estar apto a criar outras funções que atuem sobre números. É claro que o módulo "Prelude" já tem diversas funções mais usadas.

7. Crie uma função chamada `metade` que recebe um valor e retorna a sua metade, ou seja:

   $$f(x) = x/2 | x \in \Re$$