PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL - FACULDADE DE INFORMÁTICA
PARADIGMAS DE LINGUAGENS I - AULA PRÁTICA (MosML)

⇒ INSTRUÇÕES

- Para utilizar o MosML abra uma janela DOS e digite:
  set mosmllib=g:\mosml\lib
  set path=g:\mosml\bin;%path%
  h: {ou o diretório onde você armazenará os seus arquivos .sml}
  mosml
  Para facilitar a sua utilização é mais fácil criar um arquivo .bat com os comandos acima, e acrescentar um ícone para ele na área de trabalho.
- Para utilizar uma biblioteca, deve-se acrescentar os seguintes comandos no início do arquivo: load "<nome_biblioteca>" e open <nome_biblioteca>. Exemplo: *load "math"; open math;*
- Prioridade dos operadores: /, div, mod, *, +, -, =, <>, <, <=, >, >=.
- Cuidado ao digitar os comandos, pois ML diferencia letras maiúsculas de minúsculas.
- Para fazer os exercícios, consulte o material sobre a linguagem MosML que está disponível em http://www.inf.pucrs.br/~manssour/ManualMosML/index.html e http://www.inf.pucrs.br/~manssour/Paradigmasl/ml.pdf.
- Obs.: Como teremos duas aulas no laboratório para finalizar estes exercícios, traga esta folha nas próximas aulas.

⇒ EXERCÍCIOS

1. Inicialmente crie um arquivo *ascii* no *Bloco de Notas* ou no *Wordpad* com extensão ".sml", que contenha o exemplo abaixo já apresentado em aula:

```
datatype shape = point
               | circle of real
               | box of (real * real)
fun area (point) = 0.0
  | area (circle r) = 3.14 * r * r
val p = area(point);
val c = area(circle 2.3);
```

2. Após salvar o arquivo, teste o exemplo no ambiente interativo digitando a partir do *prompt* do DOS ***mosml <nome_arquivo>.sml***', ou digitando no *prompt* do DOS apenas ***mosml***' e no ambiente interativo ***use "<nome_arquivo>.sml";***. O ambiente interativo permite a entrada de declarações e a avaliação de expressões.

3. Observe que durante a avaliação das expressões aparece a seguinte mensagem: "*pattern matching is not exhaustive*". Isto ocorre porque não foi definida uma função para o argumento "*box*". Sendo assim, complete o exemplo acrescentando os comandos a seguir no arquivo criado. Salve-o e teste-o novamente.

```
...
fun area ....
    ...
    |area (box(w,h)) = w * h
...
val b = area(box(2.2,2.2));
```

4. **Exercício A**: crie, em um arquivo com extensão ".sml", um tipo de dado (*datatype*) chamado *ACTION* que possa ter os valores *Double*, *Cube* ou *Sub*. Depois, utilizando *pattern-matching*, crie uma função *FUNCTION* que recebe como parâmetro o tipo *ACTION* e um valor *x* e, conforme um dos parâmetros recebido, retorna o valor multiplicado por dois, o valor na potência três, ou o valor menos 1, respectivamente.

5. Digite o seguinte exemplo para achar o maior elemento de uma lista que utiliza comando de seleção *if*:

```
fun maior(L) = if tl(L) = []
               then hd(L)
               else if hd(L) > maior(tl(L))
                    then hd(L)
                    else maior(tl(L));
val lista = [3, 8, 4, 9, 0, 2, 1];
maior (lista);
```

6. Digite o seguinte exemplo para inverter os elementos de uma lista:

```
fun Inverte(l)=if l=nil then nil
          else Inverte(tl(l))@[hd(l)];
val lista = [3, 5, 8, 11, 14, 17, 20];
Inverte(lista);
```

7. **Exercício B**: crie, em um arquivo com extensão ".sml", uma função *SomaLista* que retorne o somatório de todos os elementos de uma lista, uma função *Double2* que recebe *x* como parâmetro e retorna *x* multiplicado por dois, e uma função *Sub2* que recebe *x* como parâmetro e retorna *x - 2*. Depois de testar estas funções, acrescente no final deste mesmo arquivo os exemplos de funções de alta ordem descritos abaixo e teste-o novamente.

```
load "list";
open list;

fun SomaLista .....
fun Double2 ....
fun Sub2 .....
:
(* funcao de alta ordem "map" - apply-to-all *)
val lista_resultados = map Double [9, 5, 10, 3, 4];

(* funcao de alta ordem "composition" *)
val F = Double2 o Sub2;
val resp1 = F(8);
val resp2 = F(5);

(* função de alta ordem "filter" *)
fun P(x) = if (x mod 2 = 0) then true else false;
filter P [8, 3, 10, 17, 2, 9];
```

8. Para exercitar o funcionamento de funções lambda (sem nome) e funções de alta ordem, digite o exemplo a seguir em um arquivo .sml e execute-o no ambiente interativo.

```
(* Funcao com nome X Funcao sem nome *)
fun cube(x) = x * x * x;
val a = cube(3);
val a = (fn x => x * x * x) (3);

(* Funcao de Alta Ordem usada para criar funcoes *)
fun reduce(binaryop, unity) =
let
      fun f(nil) = unity
        |f(x::xs) = binaryop(x,f(xs))
in
      f
end;
val soma = reduce(op +, 0)
and produto = reduce(op *, 1);
val L = [1,2,3,4];
val s = soma(L);
val p = produto(L);
```

9. **Exercício C**: crie um novo arquivo com extensão ".sml" e acrescente os comandos necessários para: amarrar uma lista de *strings* ao nome L1; amarrar uma lista de números reais que contenha números positivos e números negativos ao nome L2; usar a função *map* para trocar cada elemento negativo da lista de números reais L2 por 0, sem alterar os elementos positivos; usar a função *map* para mostrar o tamanho (=número de caracteres) de cada *string* da lista de *strings* L1. Observação: um número negativo em ML é identificado por ~.

10. O controle de fluxo em ML é feito através de dois comandos de seleção: *if <exp> then <then_exp> else <else_exp>* e *case <exp> of <match>*.

11. Digite o seguinte exemplo que utiliza comando de seleção *case*:

```
datatype MES = Jan|Fev|Mar|Abr|Mai|Jun|Jul|Ago|Set|Out|Nov|Dez;
fun dias_do_mes m = case m of Jan => 31
                            | Fev => 28
                            | Mar => 31
                            | Abr => 30
                            | Mai => 31
                            | Jun => 30
                            | Jul => 31
                            | Ago => 31
                            | Set => 30
```

```
                    | Out => 31
                    | Nov => 30
                    | Dez => 31;
dias_do_mes Ago;
dias_do_mes Fev;
```

12. **Exercício D**: crie, em um arquivo com extensão ".sml", um tipo de dado (*datatype*) chamado *DIR* que possa ter os valores *Norte*, *Sul*, *Leste* ou *Oeste*, e uma função *TESTE* que receba o tipo *DIR* como parâmetro e, utilizando *case*, retorne 0 se o valor recebido for *Norte*, 90 se for *Leste*, 180 se for *Sul* e 270 se for *Oeste*.

13. As estruturas de dados disponíveis em ML são: Lista (elementos do mesmo tipo), Tupla (elementos de tipos diferentes), Tipos Nomeados e Tipos Estruturados. Digite os exemplos a seguir no ambiente interativo para testar a utilização destes tipos e ver as "respostas" (ou assinatura) retornadas.

```
val Lista1 = [0, 1, 2];                          → Amarra uma lista ao nome Lista1
val Lista2 = [3, 4, 5];                          → Amarra uma lista ao nome Lista2
val L = [Lista1, nil, [6,7]];                    → Amarra uma lista de lista ao nome L
val Lista12 = Lista1 @ Lista2;                   → União de duas listas e amarração da lista resultante ao nome Lista12
hd (Lista12);                                    → Cabeça da lista (equivalente à função car da LP Lisp)
tl (Lista12);                                    → Cauda da lista (equivalente à função cdr da LP Lisp)
2::Lista2;                                       → Acréscimo de um elemento no início da lista
val Tupla1 = (10, 3.1416, "teste de tupla");     → Amarra uma tupla ao nome Tupla1
val Tupla2 = ("ML", 2, 8.9);                     → Amarra uma tupla ao nome Tupla2
#2Tupla1;                                        → Acesso ao segundo elemento da Tupla1
type Pessoa = string * int;                      → Amarra um nome a um tipo (usado para criar outros tipos)
type Data = int * int * int;                     → Amarra um nome a um tipo
val p1 = ("Joao", 24);                           → O nome dado ao tipo poderia ser usado para o tipo string * int
datatype Registro = Registro of Pessoa * Data;   → Definição de um tipo estruturado (Registro)
val r1 = Registro(("Joao Silva", 24), (12, 04, 1975));   → Amarra o tipo Registro ao nome r1
val r2 = Registro(("Maria Silva", 22), (21, 08, 1977));  → Amarra o tipo Registro ao nome r2
datatype Arq = Arq of Registro list;             → Definição de um tipo estruturado (Arq);
val Cadastro = Arq[r1, r2];                      → Amarra o tipo Arq ao nome Cadastro
```

14. **Exercício E**: faça, em um arquivo com extensão ".sml", uma função para criar um arquivo texto e colocar dentro dele o conteúdo de uma lista de números inteiros (um número em cada linha).