Linguagens de Programação (CK0115)
Lista de exercício 1 (Capítulo 1)

Fernanda Costa de Sousa - 485404

1. A calculator. Section 1.1

(a)

```
declare A B C D E F G
A = 2
B = A * A
C = B * B
D = C * C
E = D * D
F = E * E
G = F * F
H = G * F * C
{Browse H}
```

(b) Existe a possibilidade de haverem atalhos, nesse caso, depende de como a pessoa vai resolver o problema. O programador pode escrever todos os cálculos de forma direta mas se ele não conseguir abstrair bem o problema esse esforço talvez não compense.

2. Calculating combinations. Section 1.3

(a)

```
declare Comb Comb2
fun {Comb n k}
  if k==0 then 1
  else n*{Comb n-1 k-1} end
end
fun {Comb2 n k}
  if k==0 then 1
  else {Comb n k} div {Comb k k} end
end

{Browse {Comb2 5 3}}
```

(b)

```
declare Comb3
fun {Comb3 N K}
  if 2 * K =< N then {Comb N K}
  else {Comb N N-K} end
```

```
end

declare Fact Comb
fun {Fact N}
  if N == 0 then 1
  else N * {Fact N-1} end
end
fun {Comb N K}
  if K==0 then 1
  else {Fact N} div ({Fact N-K} * {Fact K}) end
end

{Browse {Comb3 4 2}}
```

3. Program correctness. Section 1.6

Quando N == 0.
{Pascal 0 } == [1] retorna o valor correto.
Para N == N-1: Suponha que {Pascal N-1} está correto.
Podemos assumir que AddList, ShiftList e LeftList também estão corretas.
Logo, {Pascal N} pode ser calculado como a soma de cada elemento de [0, ... {Pascal N-1}] e [... {Pascal N-1}, 0].
Com a definição {AddList {ShiftLeft {Pascal N-1}} {ShiftRight {Pascal N-1}}}
Por hipótese {Pascal N-1}, AddList, ShiftLeft, ShiftRight estão corretos
{Pascal N} também retorna a resposta correta.

4. Program complexity.

Depende do tamanho do problema. Não seria muito prático caso não houvesse limites para o tamanho. Em resumo, polinômios de ordem superior não são práticos, somente quando as entradas são pequenas.

5. Lazy evaluation. Section 1.8

Chamar {SumList {Ints 0}} não é uma boa ideia. Pois X|L1 acabará forçando a lazy. E passará X e L1 para cada chamada, em outras palavras SumList não chegaria ao outro caminho.

```
declare
fun lazy {Ints N}
  N|{Ints N+1}
end



declare SumList
fun {SumList L}
  case L of X|L1 then X+{SumList L1}
```

```
  else 0 end
end
```

6. Higher-order programming.

(a)

Em todos os lugares será 0, exceto para a linha 1. 0 aparece pela primeira vez ao calcular a linha 2.

(b)

```
declare Add Subtract Multiply Mull

fun {Add X Y}
  X+Y
end

fun {Subtract X Y}
  X-Y
end

fun {Multiply X Y}
  X*Y
end

fun {Mull X Y}
  (X+1)*(Y+1)
end

declare GenericPascalList

fun {GenericPascalList Op N}
  if N==1 then [1]
  else {GenericPascal Op N}|{GenericPascalList Op N-1} end
end

{Browse {GenericPascalList Add 7}}
{Browse {GenericPascalList Subtract 7}}
% menos
{Browse {GenericPascalList Multiply 7}}
% zeros após a segunda linha
{Browse {GenericPascalList Mull 7}}
% O aumento se concentra no centro da linha

{Browse {GenericPascal Mull 10}}
```

7. Explicit state.

Para variáveis o Browse mostra 23. E para células ele mostra 44. 44 é um escopo diferente. Como idenficador, vai funcionar aquele declarado ṕor último. Sobre as variáveis, 23 e 44 são armazenados em lugares diferentes.

```
local X in
       X = 23
       local X in
          X=44
       end
       {Browse X}
       % browse mostra o x que é colocado no mesmo escopo léxico (X=23)
    end

    local X in
       X={NewCell 23}
       X:=44
       {Browse @X}
       % mostra 44, porque := atribui a célula X para o valor 44
    end
```

8. Explicit state and functions

Podemos ver que a célula de memória está sendo declarada e inicializada dentro da função, os argumentos são salvos em áreas diferentes, no armazenamento. Isso faz com que o acumulador não acumule os inputs. Uma forma de corrigir seria:

```
declare
local Acc = {NewCell 0} in
  fun {Accumulate N}
     Acc:=@Acc+N
     @Acc
  end
end

{Browse {Accumulate 5}}
{Browse {Accumulate 100}}
{Browse {Accumulate 34}}
```

10. Explicit state and concurrency. Section 1.15

(a) Obtive como resultado: 0, 4, 6, 6, 6, 7. A única vez que obtive 1 foi quando executei apenas a thread J. (Mas isso não é válido para esse teste).

```
declare
C={NewCell 0}

thread I in
   I = @C
   C := I + 1
end

thread J in
   J = @C
   C := J + 1
end

{Browse @C}
```

(b)

```
declare
Z={NewCell 0}

thread K in
   K = @Z
   {Delay 1000}
   Z := K + 1
end

thread L in
   L = @Z
   Z := L + 1
end

{Delay 1000}

{Browse @Z}
```

(c) Mesmo adicionando delay, ela continua mostrando o resultado 2 e nunca o 1.