2ª Lista de Exercícios de Paradigmas de Programação

Prof. Glauber Cintra – Entrega: 27/mai/2013

EQUIPE: Alessandra Lino, Alisson Rangel, Gracyane Oliveira, RafaellaKeury

1. **O que são estruturas de controle? Cite alguns exemplos de estruturas de controle.**

   Uma estrutura de controle é uma instrução de controle e sua coleção de comandos cuja execução ela controla. Essas instruções de controle podem desviar de certos trechos de programas ou repetir a execução de partes do programa.

   Exemplos:

   //Estrutura de repetição

```
for(i = 1 ; i < = 10; i++){
        printf(“%d”, i);
}
```

   //Estrutura de desvio

```
If(i%2==0) printf(“Numero par”)
elseprintf(“Numero impar”);
```

2. **Justifique por que o uso de desvios incondicionais deve ser evitado. Em que casos eles são aceitáveis?**

   O uso de desvios incondicionais devem ser evitados porque dificultam o entendimento do fluxo de execução do programa, portanto os programas ficam difíceis de serem lidos, logo, pouco confiáveis e com alto custo de manutenção.
   É aceitável o uso do break quando ele está dentro de um switch;
   Também se aceita o uso do return que finaliza a execução de um subprograma.

3. **Explique o funcionamento do trecho de código abaixo. Para que serve tal código?**

   for (i = 0, s = 0; i <= n; i++) { s += i; }

   Antes da primeira iteração a expressão 1 é avaliada (**i = 0, s = 0**). No início de cada iteração, a expressão 2 é avaliada, se for verdadeira, o bloco é executado a expressão 3 é computada e uma nova iteração é iniciada. Se for falsa, o laço é finalizado.
   O código gera a soma dos n primeiros termos de uma progressão aritmética.

4. **Explique o que é a assinatura de um subprograma e como ela é constituída.**

   A assinatura é o protocolo de utilização do subprograma.É constituída de nome, especificação dos parâmetros de entrada, do tipo de valor devolvido, de erros e exceções que podem ocorrer, modificador de acessibilidade e outros.

5. **O que é sobrecarga de subprogramas? Cite linguagens que permitem sobrecarga de subprogramas.**

   A sobrecarga de subprogramas é quando um subprograma tem diversas versões com assinaturas diferentes. Como exemplo de linguagens que possui tal característica podemos citar: *Java, C++ e Ada*.

6. **Explique os mecanismos de passagem de parâmetros por valor e por referência.**

   Em passagem de *parâmetros por valor* os parâmetros declarados na assinatura do subprograma, chamados de parâmetros formais(PF), recebem uma cópia dos parâmetros que aparecem na chamada a um subprograma, chamados de parâmetros reais(PR).

   Na passagem de *parâmetros por referência* o parâmetro formal (PF) e o parâmetro real(PR) correspondente compartilham a mesma região de memória.

7. **Defina o que é uma Cláusula de Horn.**

   Uma cláusula de Horn é uma fórmula da lógica de predicados que expressa uma implicação na qual a premissa é uma conjunção de predicados e o consequente é o único predicado. Além disso, todas as variáveis estão quantificadas com quantificador universal.

8. **Indique qual será o resultado produzido pelo programa:**

```
pai(rui, gil).
pai(gil, ivo).
pai(gil, edu).
neto(X, Y) :- pai(Y, Z), pai(Z, X).
```

para as seguintes metas:

- pai(rui, edu).→**false**
- pai(R, edu).→gil
- pai(gil, R).→ivo
- neto(rui, gil).→**false**
- neto(edu, rui).→**true**
- neto(R, gil).→**false**
- neto(edu, R).→rui

9. **Dado o trecho de código abaixo, escrito em Prolog, qual será o valor atribuído a L após avaliar a seguinte meta: enigma([6, 2, 4], [ ], L). Para que serve o predicado enigma?**

```
1 -enigma([ ], Lista, Lista).
2 -enigma([Cb|Cd], L1, Nlista) :- aux(Cb, L1, L2), enigma(Cd, L2, Nlista).
3 -aux(X, [ ], [X]).
4 -aux(X, [Cb|Cd], [X, Cb|Cd]) :- X < Cb.
5 -aux(X, [Cb|Cd], [Cb|Lista]) :- aux(X, Cd, Lista).
```

Regra 2: **L2 = [6]** **L2 = [6]**

enigma ([6,2,4],[ ],L) : - aux (6, [ ] , L2), **enigma([2,4], L2, L).**

aux: casa com o fato 3: L2 = [6]

L2 = [6] **L2 = [6]** **L3 = [2,6]**

enigma([2,4], [L2], L): -aux (2, [L2] , L3), **enigma([4], L3, L).**

aux: casa com a regra 4: L3 = [2,6]

L3 = [2,6] **L3 = [2,6]**

enigma([4], [L3], L): -aux (4, [L3] , L4), enigma([ ], L4, L).

aux: casa com a regra5: L4 = [2,4,6]

enigma: casa com o fato 1: L = L4 = [2,4,6]

RESPOSTA:

Valor atribuído a L = [2, 4, 6];

O predicado enigma serve para colocar uma lista em ordem crescente.

10. Defina um predicado em Prolog que receba os coeficientes a, b e c de um polinômio do segundo grau da forma $ax^2 + bx + c$ e escreva as raízes do polinômio (seu predicado deverá ser capaz de tratar o caso em que as raízes são imaginárias).

```
raizComplexa(A,B,C):- calculo(A,B,C,R,P),
        write('Raizescomplexas'),nl,
        write('Raiz 1: '),
        write(R),write('+'), write(P),write('i'),nl,
        write('Raiz 2: '),
        write(R),Qis (-1)*P, write(Q),write('i').

calculo(A,B,C,R,P):-
        Sis (B^2 - 4*A*C),
        S< 0,T is S*(-1),
        Ris -B/(2*A),
        P issqrt(T)/2*A.

raizReal(A,B,C,R1,R2):- Mis (B^2 - 4*A*C),
        M>=0,
        R1is (-B+sqrt(M))/(2*A),
        R2is (-B-sqrt(M))/(2*A);
        write(R1),nl,write(R2).

raiz(A,B,C):-raizReal(A,B,C);raizComplexa(A,B,C).
```

11. Defina um predicado em Prolog de aridade3 onde os dois primeiros argumentos são listas e o terceiro argumento é uma lista contendo a interseção das duas primeiras. Por exemplo, se o nome do seu predicado for intersecao, a meta intersecao([b, o, l, a], [l, u, a], Resultado) deve atribuir [l, a] à variável Resultado.

```
membro(X, [X|_]).
membro(X, [_|Y]) :-membro(X, Y).
intersecao([X | Y], L, [X |Z]) :-membro(X, L),intersecao(Y, L, Z).
intersecao([_ |X], L, Y) :-intersecao(X, L, Y).
intersecao(_, _, []).
```

12. Para que serve a seguinte função?

```
(defun enigma (L)
   (if (null L) t
      (if (null (cdr L)) t
         (if (> (car L) (car (cdr L))) nil
       (enigma (cdr L))
       )
)
)
```

)

A função serve para verificar se uma *lista está em ordem crescente*.
Primeiro é feito duas verificações, se a lista ou a cauda forem nulas, então retorna **t (true),** indicando que a lista está em ordem crescente**,** se a lista não for nula é verificado se a cabeça da lista é maior que a cabeça da cauda, se for então é retornado **nil**(false), caso contrário o enigma é novamente chamado com a cauda como parâmetro e o processo se repeti a até a verificação completa da lista.

13. Descreva a semântica da função COND do Lisp.

A função **COND**implementa um desvio condicional. Pode ter um número arbitrário de argumentos, chamado de cláusula, e consiste de uma lista de exatamente duas expressões. O primeiro elemento da cláusula é a condição e o segundo elemento é a ação ou resultado.
O **COND** encontra a primeira cláusula que avalia para true, executando a ação respectiva e retornando o valor resultante. Nenhuma das restantes é avaliada.

Sintaxe:

```
(cond
        (<condição1><resultado1>)
        (<condição2><resultado2>)
        ....
        (<condição3><resultado3>))
(tresultadoN ) )
```

O valor retornado pelo **COND** é calculado da seguinte forma: se condição1 é verdade (não NIL) retorna resultado1; senão secondição2 for verdade, então retorna resultado2; senão se ..., então retornar resultadoN.

Na maioria dos sistemas LISP, é um erro se nenhuma das condições forem verdadeiras deixando o resultado da **COND** indefinido. Por esta razão, **t** é geralmente utilizado como a condição final. Porém senão for utilizada esta condição final e se todas as condições são avaliadas como nil(false) **COND** retornará nil.

14. Escreva uma função em Lisp que receba como argumento uma lista contendo apenas números e devolva a multiplicação dos números contidos na lista.

```
(defun mult(L)
        (if (null L) nil
                (if(null (cdr L)) (car L)
                        (* (car L)(mult (cdr L)))
                )
        )
)
```

15. Escreva uma função em Lisp que calcule os números de Fibonacci. Lembre-se que fib(n) = 1, se n = 0 ou n = 1; caso contrário, fib(n) = fib(n -1) + fib(n - 2).

```
(defun fib(n)
        ( if(< n 2) ( if (>= n 0) 1 “valor invalido”)
                (+( fib (-n 1)) ( fib (- n 2) ) ) )
        )
)
```