Terceira Lista de Exercícios de Linguagens de Programação.
Primeiro Semestre de 2007
Departamento de Computação – UFSCar.
José de Oliveira Guimarães.

O número entre parênteses ao lado do número do exercício indica a sua importância. Mais alto o número mais importante o exercício.

1. (10) Cite alguns motivos pelos quais linguagens funcionais são de mais alto nível do que as linguagens imperativas (em geral).

2. (10) Explique porque a atribuição implica na introdução do fator "tempo" nos programas tornando o seu entendimento difícil.

3. (3) Por quê eliminando a atribuição devemos eliminar também os comandos `while`, `for` e semelhantes ?

4. (3) Qual a diferença entre inicialização e atribuição ?

5. (3) Por quê uma linguagem funcional pura não pode ter a construção "seqüência de comandos" ? Uma seqüência de comandos é colocada entre `begin` e `end` na linguagem S. Não considere a existência de comandos de entrada e saída na sua resposta.

6. (3) Explique porque em uma linguagem funcional pura não podem existir variáveis globais e passagem por referência.

7. (8) Por quê o comando de atribuição torna as linguagens imperativas de mais baixo nível ?

8. (3) A comparação

```
f() == f()
```

resultará sempre em `true` na linguagem S (C, Pascal, Fortran, etc) ? E em uma linguagem puramente funcional ?

9. (10) O que é transparência referencial ?

10. (10) Como a transparência referencial ajuda na legibilidade dos programas ?

11. (3) Aplique a regra da reescrita sobre `comb(1,0)` usando a função

```
proc comb( n, i : integer ) : integer
  is
    fat(n)/(fat(n-i)*fat(i));
```

em que `fat` é a função fatorial.

12. (3) A regra da reescrita é válida em Pascal, C ou alguma outra linguagem imperativa que você conheça ? Por quê ?

13. (3) Explique polimorfismo em LISP, citando um exemplo como auxílio.

14. (3) Dê um exemplo de um erro de execução em Lisp. A linguagem é segura ?

15. (8) Que estrutura de dados são utilizadas para representar programas em Lisp ? E dados ? Pode-se gerar uma função em tempo de execução e executá-la ?

16. (10) Cite duas formas funcionais de FP.

17. (8) Qual é o gargalo das máquinas Von Neumann ? Em que formas funcionais de FP podemos identificar claramente que ela não foi feita para este tipo de arquitetura ?

18. (3) Seria possível colocar tipos em Lisp ? Dê um exemplo de um programa correto (que não produz erros de tipo) em Lisp que seria incorreto em Lisp com tipos. Faça você mesmo a sintaxe da linguagem "Lisp tipada".

19. (5) Seria possível colocar o comando de atribuição em uma linguagem funcional e ainda assim garantir que as funções não produzam efeitos colaterais ? Defina tudo o que for necessário para garantir este objetivo.

20. (3) Compare Lisp com Smalltalk com relação ao polimorfismo.

21. (3) Que formas funcionais de FP podem substituir alguns comandos de repetição ? Em particular, como você escreveria o código

```
i = 1;
while i < n do
  begin
  write(v[i]);
  i = i + 1;
  end
```

em FP ? Admita que `n` seja o tamanho do vetor `v`.

22. (3) Faça um comando `case` de Pascal ou `switch` de C para uma linguagem funcional.

23. (5) Podemos fazer um `if` para uma linguagem funcional onde o `else` é opcional ?

24. (8) Compare polimorfismo de SML com classes parametrizadas.

25. (10) Cite um exemplo onde o compilador de SML não conseguiria deduzir os tipos de uma expressão.

26. (10) Cite um exemplo em SML de uma função totalmente polimórfica (que aceita parâmetros de infinitos tipos).

27. (10) Que tipos um compilador de SML colocaria para as funções abaixo ?

```
proc f( a, b, c )
 is
   if a > 1
   then
     c
   else
     if a == b
     then
       true
     else
       false;

proc g(a, b, c)
  is
    if a > b
    then
      c
    else
      if b > 5
      then
        a
      else
        c;
```

28. (5) Alterando o código de uma função em SML podemos alterar o seu tipo e tornar inválido um código que chama esta função ? Cite um exemplo.

29. (3) Compare iteradores com forma funcional alpha (aplique a todos) de FP.

30. (10) Que tipo o compilador de SML colocaria para a função abaixo ?

```
proc  f( h::t )
        is h::h::f(t)
    | f( [] )
        is [];
```

O que ela faz ?

31. (10) O que é avaliação preguiçosa ?

32. (5) Cite um exemplo onde uma função é passada como parâmetro a outra função.

33. (10) Por quê programadores são mais produtivos se utilizam linguagens de alto nível ?

34. (10) Cite algumas vantagens de linguagens funcionais sobre linguagens imperativas.

35. (10) Cite as desvantagens de linguagens funcionais.

36. (10) Que característica de linguagens funcionais torna difícil o mapeamento de problemas do mundo real em programas ?

37. (10) Dada a base de dados

```
pA(a).
pA(b).
pB(X):- pA(X), X = b.
```

Quais as respostas (todas) dadas pelo goal abaixo ?
`?- pB(Y).`

38. (10) Defina e dê um exemplo de instanciação.

39. (10) Defina e dê um exemplo de retrocesso (*backtracking*).

40. (10) Para as tentativas de emparelhamento abaixo, descubra se há emparelhamento (*matching*) e qual a instanciação final das variáveis.

- `pA([1, 2])` ≡ `pB([1, 2])`
- `map([1, 2, 3])` ≡ `map([H::T])`
- `pE([])` ≡ `pE([H::T])`
- `map([joao, maria])` ≡ `map([X, Y])`
- `map([joao, maria])` ≡ `map([X, Y | T])`
- `pE([1])` ≡ `pE([H::T])`
- `composto(agua, F, oleo, G)` ≡ `composto(A, farinha, oleo, H)`

41. (10) Admita que o predicado `writeln` seja sempre verdadeiro e escreva o seu argumento no vídeo. O que imprime o goal
`?- pC(X).`
considerando a base de dados abaixo ?

```
pA(a):- writeln('A#a').
pA(b):- writeln('A#b').
pA(c):- writeln('A#c').
pB(b):- writeln('B#b').
pB(c):- writeln('B#c').
pC(X):- pA(X), pB(X).
pC(X):- pB(X), X = a.
```

42. (10) Faça um programa em Prolog e mostre quais são as suas regras, fatos, cláusulas e predicados. O que é a base de dados ?

43. (5) Dado um conjunto de fatos representados pela estrutura

```
movel( Nome, Fabricante, Peso,
       Altura, Comprimento,
       Tipo )
```

podemos criar operações como:

```
nome_movel( Nome,
   movel(Nome, _, _, _, _, _)
         )
```

O predicado acima retorna, em seu primeiro argumento, o nome do móvel. Um programa que usa apenas estas operações, sem manipular as estruturas `movel` diretamente, é independente da organização desta estrutura. Identifique esta técnica dentre os conceitos já vistos no curso.

44. (10) O que imprime o goal
`?- pC(X).`
se é usada a base de dados abaixo ?

```
pA(a):- writeln('A#a').
pA(b):- writeln('A#b').
pA(c):- writeln('A#c').
pB(b):- writeln('B#b').
pB(c):- writeln('B#c').
pB(d):- writeln('B#d').
pC(c):- writeln('C#c').
pC(d):- writeln('C#d').
pE(X):= pA(X), pB(X), !, pC(X).
```

45. (8) Explique como o *cut* pode aumentar a eficiência de um programa.

46. (10) Faça um pequeno predicado onde a introdução de um *cut* causa a modificação da semântica do predicado. Isto é, pelo menos uma consulta a ela resulta em resposta diferente da anterior (sem *cut*).

47. (10) Por que um programa em Prolog não é formado apenas pela base de dados ?

48. (5 Faça um programa em Prolog em que todo o algoritmo utilizado está implícito na linguagem.

49. (3) O que acontece se um programa em Prolog utiliza algoritmos muitíssimo diferentes do algoritmo de unificação ?

50. (10) Cite um exemplo de erro em execução em Prolog.

51. (10) Cite um exemplo de um erro de tipos em Prolog que não é notado em compilação ou execução. Isto é, o programa está errado, o erro não é identificado pelo sistema mas poderia ser se Prolog fosse estaticamente tipado.

52. (3) Faça um predicado polimórfico em Prolog.

53. (5) Qual a diferença entre *cuts* vermelhos e verdes ?

54. (8) Como funcionam os predicados `assert` e `retract` ?

55. (8) Dada a base de dados

```
fibo(1, 1).
fibo(2, 1).
fibonacci(N, S) :-
  fibo(N, S).
fibonacci(N, S) :-
  N1 is N - 1,
  N2 is N - 2,
  fibonacci(N1, S1),
  fibonacci(N2, S2),
  S is S1 + S2,
  assert( fibo(N, S) ).
```

O que escreve o goal

```
?- fibonacc(5, S), list(fibo)
```

em que `list` é um predicado que sempre sucede e imprime todas as cláusulas do predicado que é seu argumento (parâmetro) ?

56. (10) Cite os motivos que fazem com que Prolog não seja uma linguagem completamente lógica.

57. (8) Faça um predicado em que a ordem em que as cláusulas de um predicado estão textuamente ordenadas é importante.

58. (8) Faça um predicado em que a ordem dos *goals* no corpo de um predicado é importante.

59. (10) Monte o gráfico de fluxo de dados para os trechos de código abaixo.

- a)

```
a = b + 1;
if a > 2 then c = b; endif
b = 3;
c = a + b;
```

- b)

```
i = 1;
s = 0;
while i <= N do
  s = s + i;
```

- c)

```
a = b;
b = a;
c = d + 1;
d = a + b;
```

60. (10) Cite duas instruções do exercício anterior que podem e duas que não podem ser executadas em paralelo. Explique.

61. (8) Explique o que é regra de atribuição única. Cite um exemplo que demonstra que ela é necessária.

62. (10) Explique como podemos ter laços `for` e `while` em linguagens data-flow e ainda ter atribuições dentro destes laços. Podemos executar diversas instruções de diversos passos do laço em paralelo ? A regra da atribuição única não é quebrada ? Os valores de uma variável em diversos passos do laço não se confundem ?

63. (10) Qual é o objetivo das linguagens *data flow* ? Como e quando as instruções de um programa são executadas ?

64. (5) Linguagens *Data-Flow* possuem granularidade de paralelismo muito alta — muitas pequenas instruções executando em paralelo. O gerenciamento deste paralelismo é complexo para ser eficiente. Portanto, uma alternativa para aumentar a eficiência deste tipo de linguagem é diminuir a granularidade. Por exemplo, poderíamos permitir paralelismo apenas entre subrotinas, nunca dentro delas. Desenvolva esta idéia. Cite exemplos.