Paradigmas de Linguagens de Programação
Exame Escrito
Centro de Informática – UFPE, 12 de junho de 2018

Questão 1 [2,0]
Considere uma lista de pares da forma ***[(e1,true),(e2,false),(e3,true),(e4,true),...],*** representando elementos ativos (quando o booleano é ***true***) ou inativos. Defina uma função ***act*** que recebe uma lista com este formato e retorna uma lista com os elementos ativos. Ex.: ***act [(e1,true),(e2,false),(e3,true),(e4,true)] = [e1,e3,e4]***. Analogamente, defina a função ***inact*** que retorna uma lista com os elementos inativos.
Ex.: ***inact [(e1,true),(e2,false),(e3,true),(e4,true)] = [e2]*** .

Questão 2 [2,0] Assumindo que ***xs*** é uma lista de pares como na questão antetior, prove, por indução: ***# xs = # (act xs) + # (inact xs)***. [Dicas: você vai precisar definir a função #, que retorna o tamanho de uma lista. No passo indutivo, você irá precisar considerar um par ***(e,b)*** e deve considerar tanto o caso onde ***b*** é ***true***, como o caso onde ***b*** é ***false***.]

Questão 3 [2,0] Considere o seguinte programa em uma linguagem semelhante à LI2:

```
{ var a  =  1; b = 2;
  { proc P (par int x, par int y)  {x := x + a; y := b + a};
    call P(a,b);
    write(a,b)
  }
}
```

Informe o resultado produzido para cada um dos casos quando ***par*** é interpretado com os seguintes tipos de passagem de parâmetro: a) valor-resultado e b) referência.

Questão 4 [1,0] Assinale com V (para verdadeiro) ou F (para falso):

( ) O conceito de classes pode ser considerado uma extensão do conceito de tipos abstratos de dados (TAD) com atributos que representam estado, já que TAD não possuem estado.

( ) Segundo um dos princípios de regularidade sugeridos no livro texto: todos os tipos de uma linguagem devem ser disponíveis tanto para variáveis *transitórias* quanto *persistentes*.

Questão 5 [3,0] Estenda a linguagem LI1 com um comando condicional **case**, como em Pascal, com a seguinte forma geral, onde os termos sublinhados são palavras reservadas:

```
case exp of
   val1: c1
   ...
   valn: cn
  else cn+1
end
```

A expressão ***exp*** tem que ser inteira e os valores ***val1 ... valn*** são inteiros ou intervalos da forma ***inf..sup***, onde ***inf*** e ***sup*** são valores inteiros. Se o valor de ***exp*** for (ou estiver no intervalo definido por) ***val1***, então ***c1*** deve ser executado, senão o próximo valor é testado, e assim por diante. Se o valor de ***exp*** for diferente (e estiver fora dos intervalos) de ***val1...valn***, então o comando associado à cláusula **else** é executado. A implementação deve considerar os métodos de avaliação e checa tipo, bem como as classes novas ou modificadas, impactadas por esta mudança. Particularmente:

1) Defina a BNF para a linguagem redefinida, destacando apenas o que mudar.
2) Explique se é necessária alguma mudança nos ambientes de compilação e execução.
3) Implemente novas classes, se for o caso, e indique todas as classes que seriam afetadas (ilustre a modificação em pelo menos uma classe impactada).

Boa Sorte

Apêndice 1. BNF de LI1.

Programa ::= Comando

Comando ::= Atribuicao
| ComandoDeclaracao
| While
| IfThenElse
| IO
| Comando ";" Comando
| Skip

Skip ::=

Atribuicao ::= Id ":=" Expressao

Expressao ::= Valor | ExpUnaria | ExpBinaria | Id

Valor ::=ValorConcreto

ValorConcreto ::= ValorInteiro | ValorBooleano | ValorString

ExpUnaria ::= "-" Expressao | "not" Expressao | "length" Expressao

ExpBinaria ::= Expressao "+" Expressao
| Expressao "-" Expressao
| Expressao "and" Expressao
| Expressao "or" Expressao
| Expressao "==" Expressao
| Expressao "++" Expressao

ComandoDeclaracao :: = "{" Declaracao ";" Comando "}"

Declaracao ::= DeclaracaoVariavel | DeclaracaoComposta

DeclaracaoVariavel ::= "var" Id "=" Expressao

DeclaracaoComposta ::= Declaracao "," Declaracao

While ::= "while" Expressao "do" Comando

IfThenElse ::= "if" Expressao "then" Comando "else" Comando

IO ::= "write" "(" Expressao ")" | "read" "(" Id ")"