Paradigmas de Linguagens de Programação
Exame Escrito
Centro de Informática – UFPE, 19 de junho de 2013

Questão 1 [2,0] Defina uma função ***update*** que recebe três argumentos (um número natural, ***i***, usado como um índice, um elemento, ***x***, e uma lista de elementos, ***L***, podendo assumir que o valor de ***i*** é sempre menor que o tamanho da lista ***L***). A função deve retornar uma outra lista com os mesmos elementos da lista original, exceto que, na posição indexada por ***i***, o elemento original deve ser substituído por ***x***. Considere que o primeiro elemento da lista tem índice 0, o segundo 1 e assim por diante, até o último elemento que tem como índice o tamanho da lista menos 1. Por exemplo, ***update(1,10,[2,3,5,8]) = [2,10,5,8]***. Defina uma outra função, ***get***, que recebe um índice e uma lista e retorna o elemento na respectiva posição. Novamente, pode assumir que o índice fornecido é menor que o tamanho da lista. Por exemplo***, get(1,[2,3,5,8]) = 3*.**

Questão 2 [2,0] a) Defina uma propriedade das funções definidas na questão anterior que garante o seguinte. Considere a aplicação da função ***update*** a uma lista, onde o novo elemento ***x*** deve substituir o que estava em uma posição ***i***. Se a função ***get*** for aplicada à lista retornada pelo ***update***, na mesma posição ***i***, então o resultado de *get* será sempre o próprio elemento ***x***. b) prove esta propriedade por indução.

Questão 3 [2,0] Considere o ambiente das linguagens imperativas LI1 e LI2, que incluem um mapeamento entre variáveis e valores (uma memória abstrata). É possível implementar passagem de parâmetros por referência com este mapeamento? Se sim, explique em linhas gerais uma solução. Caso contrário, sugira qual a mudança necessária no ambiente.

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

( ) O princípio da completude dos tipos (***type completeness principle***) afirma que variáveis persistentes e transientes devem ser tratadas uniformemente.
( ) Avaliação sob demanda e polimorfismo paramétrico são exclusivos do paradigma funcional.
( ) Uma linguagem de acesso a banco de dados, sem alguma estrutura de repetição ou recursão, não é, estritamente, uma linguagem de programação.
( ) Em um bloco com declarações colaterais, a ordem das declarações não importa

Questão 5 [3,0] Estenda a linguagem LI1 com os seguintes comandos:

a) Comando de atribuição múltipla (***x1 := x2 := ... xn := e***) que avalia a expressões ***e*** e atribui o respectivo valor às variáveis ***x1,...,xn***.
b) Comando Repeat.
   repeat c until b
   que repete a execução do comando ***c*** até que ***b*** se torne ***true***. Note que o comando ***c*** será executado pelo menos uma vez, já que a condição é testada ao final do bloco.

A implementação de ambos os comandos deve considerar os métodos de avaliação e checa tipo, bem como todas as classes auxiliares necessárias à completa implementação. Particularmente:

1) Defina a BNF para a linguagem redefinida, destacando apenas o que mudar.
2) Explique o que precisaria mudar no parser e se é necessária alguma mudança nos ambientes de compilação e execução.
3) Implemente as novas classes que se mostrem necessárias e modifique as existentes quando necessário.

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 ")"