COMPILADORES

Prova 2 – 21/03/2019 – Prof. Marcus Ramos

Questão 1 (2 pontos): Para que serve e como é usada a Tabela de Símbolos durante a subfase identificação da fase de análise de contexto?

A Tabela de símbolos é central na subfase de identificação. Ela é usada, inicialmente (num primeiro passo), para coletar e armazenar todos os nomes declarados no programa e os seus atributos, identificando também o escopo se for o caso. Num segundo momento (segundo passo), ela é usada para verificar se todos os nomes que estão sendo usados no programa foram previamente declarados. Isto é feito por meio de uma consulta usando o nome em questão como chave. Como um tipo abstrato de dados (ADT), a Tabela de Símbolos pode preservar a sua estrutura física, garantindo acesso por meio de operações como inserção, pesquisa, abertura e fechamento de escopo.

Questão 2 (2 pontos): Descreva como a subfase verificação de tipos da fase de análise de contexto utiliza e modifica a AST (árvore de sintaxe abstrata).

A subfase de verificação de tipos percorre a AST em profundidade primeiro, visitando inicialmente os filhos e depois o pai (supondo que os filhos representam os operandos e os pais as operações), buscando determinar o tipo de cada operando (através da identificação prévia) e os tipos dos resultados (por meio das assinaturas de operações disponíveis), até que todas as expressões e subexpressões sejam tipadas. Como resultado, todos os nós de todas as expressões e subexpressões são rotulados com o tipo correspondente, ou alguma indicação de erro se não for possível fazer a tipagem.

Questão 3 (2 pontos): Considere que um certo gerador de código lineariza matrizes por colunas e não por linhas (como visto em sala de aula). Considere ainda a seguinte declaração Pascal:

```
var m: array [5..10] of array [7..20] of integer;
```

supondo que size [integer]=2. Obtenha a fórmula que retorna o endereço de um elemento m[i][j].

Se a linearização fosse por linhas (como visto em sala de aula), a fórmula seria:

$$address\ [m[i][j]] = base + (i-5) * 26 + (j-7) * 2$$

Com a linearização feita mas colunas, a fórmula se torna:

$$\mathbf{address\ [m[i][j]] = base + (j-7) * 12 + (i-5) * 2}$$

Questão 4 (2 pontos): Considere o programa Pascal apresentado abaixo. No ponto marcado com `(*AQUI*)`, mostre:

- O endereço de cada variável e parâmetro visível;

- A situação completa da pilha de execução, com SB, LB, LE, LD, ER, ST, variáveis globais e locais, parâmetros etc (considere o fluxo p→q→r→q→r→r);

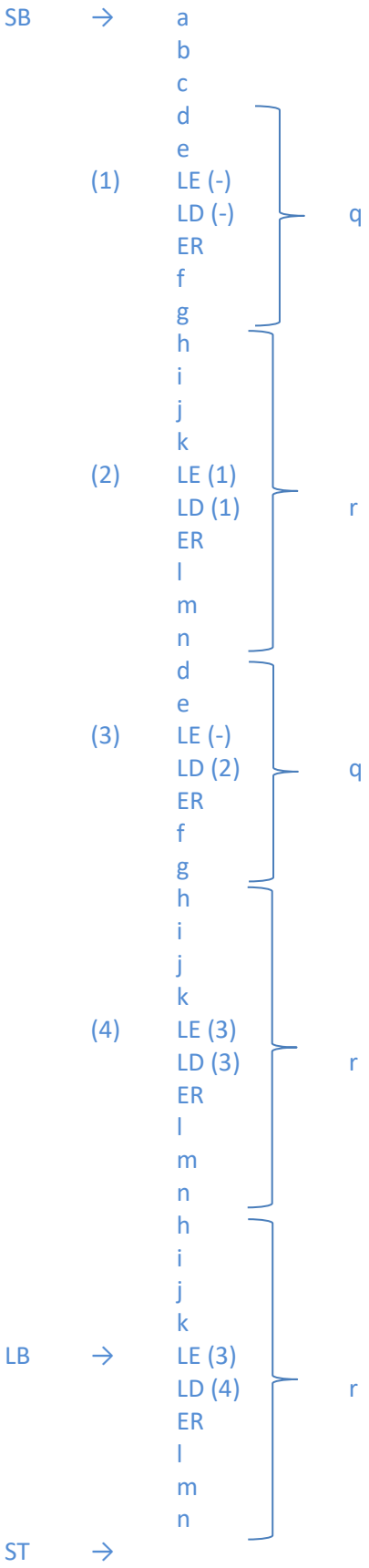
```
programa p;
var a,b,c: integer;
procedure q (d,e: integer);
      var f,g: integer;
      procedure r (h,i,j,k: integer);
            var l,m,n: integer;
            begin
            (*AQUI*)
            end;
      begin
      ...
      end;
begin
...
end.
```

Considere que cada endereço e valor inteiro ocupa uma única posição de memória.

Endereços no ponto indicado:

Nome	Natureza	Deslocamento	Registrador
a	Variável	0	SB
b	Variável	1	SB
c	Variável	2	SB
d	Parâmetro	-2	L1
e	Parâmetro	-1	L1
f	Variável	3	L1
g	Variável	4	L1
h	Parâmetro	-4	LB
i	Parâmetro	-3	LB
j	Parâmetro	-2	LB
k	Parâmetro	-1	LB
l	Variável	3	LB
m	Variável	4	LB
n	Variável	5	LB

Pilha de execução considerando o fluxo informado:



Questão 5 (2 pontos): Na linguagem Ada, os operadores lógicos AND THEN e OR ELSE recebem a denominação “curto-circuito” pois eles avaliam o segundo operando apenas se o valor do primeiro não permitir determinar o resultado da conjunção ou da disjunção, respectivamente. Em outras palavras, p AND THEN q só avalia q se p for verdadeiro, caso contrário retorna falso imediatamente. De maneira similar, p OR ELSE q só avalia q se p for falso, caso contrário retorna verdadeiro imediatamente. Em comparação, os operadores AND e OR sempre avaliam os dois operandos, independentemente do valor do primeiro deles. Obtenha um template de código para cada um dos operadores AND THEN e OR ELSE da linguagem Ada.

```
evaluate [p AND THEN q] =
      evaluate [p]
      JUMPIF (0) g
      evaluate [q]
      JUMP h
g:    LOADL 0
h:

evaluate [p OR ELSE q] =
      evaluate [p]
      JUMPIF (1) g
      evaluate [q]
      JUMP h
g:    LOADL 1
h:
```

supondo, em ambos os casos, que 0 representa falso e 1 representa verdadeiro.