Lista 2 preparatória para prova de compiladores V2

Questão 1) A fase de análise semântica normalmente é a ultima fase de análise de um compilador e tem a responsabilidade de realizar as validações semânticas (contextuais). De forma análoga a fase de geração de código é a última fase de síntese. Nesse contexto, implementar em pseudocódigo a lógica de *analisar* e a lógica de *gerar* de um nó do tipo NodeFor com a regra a seguir (para a linguagem SPP – Script de Pascal em Português):

```
<For> ::=  'para' '(' <Atrib> ';' <Exp_Rel> ';' <Atrib> ')' '{' <List_Coman> '}'
```

Obs1.: um exemplo de comando estilo "For" , no MLP, na definição acima poderia ser:

```
i : inteiro ;
para ( i = 0 ; i < 10 ; i = i + 1 ) {
     escreve ( i ) ;
}
```

Obs2.: o pseudocódigo de *analisar(Node no) e gerar(Node no)* dos não terminais <List_Coman>, <Atrib> e <Exp_Rel> podem ser considerados já implementados.

Questão 2) Crie a(s) regra(s) gramatical(is) para um comando FOR (estilo C) no estilo do SLI (Script de List em Italiano) e implemente os métodos de análise semântica (*analisar*) e de interpretação (*interpretar*) para o nó do tipo No_For.

Questão 3) A fase de análise semântica de um compilador (ou interpretador) tem a responsabilidade de executar as validações que as fases de análise léxica e sintática não são capazes de processar. No fragmento de programa abaixo, escrito na linguagem SPP (Script de Pascal em Português), quais as ações semânticas serão necessárias? Cite e comente todas as ações semânticas que serão necessárias, em cada linha de código (se tiver ação semântica), detalhadamente.

```
def teste () : inteiro {
      a1, a2, b2 : inteiro ;
      a1 = 0 ;
      a2 = 0 ;
      se ( a1 > a2) {
            c1 = c2 + 43 ;
      }
      retorna c1 ;
}
```

Questão 4) Compiladores de linguagens de programação traduzem programas-fonte, em uma linguagem de entrada, para programas-objeto, em uma linguagem de saída. Durante o processo de tradução, o compilador deve verificar se as sentenças do programa-fonte estão sintaticamente

corretas. Esse processo de análise sintática pode ser realizado construindo-se uma árvore de análise segundo duas principais abordagens: top-down, quando a árvore é investigada da raiz às folhas; ou bottom-up, das folhas à raiz. Acerca desse assunto, julgue os itens seguintes.

I) A análise top-down é adequada quando a linguagem de entrada é definida por uma gramática recursiva à esquerda.

II) Independentemente da abordagem adotada, top-down ou bottom-up, o analisador sintático utiliza informações resultantes da análise léxica.

III) Se os programas em uma linguagem podem ser analisados tanto em abordagem top-down como em bottom-up, a gramática dessa linguagem é ambígua.

IV) A análise bottom-up utiliza ações comumente conhecidas como deslocamentos e reduções sobre as sentenças do programa-fonte.

Estão certos apenas os itens

a) II, III e IV.
b) II e IV.
c) I, III e IV.
d) I e III.
e) I e II.

Questão 5) As fases de análise de um compilador tradicional são: análise léxica, análise sintática e análise semântica. Nesse contexto definir a função de cada fase de análise e exemplificar com código SPP (Script de Pascal em Português) pelo menos um tipo de erro que deve ser capturado por cada analisador.

Questão 6) Uma das fases mais importantes de um compilador é a fase de análise sintática onde se têm algumas técnicas como a análise sintática preditiva recursiva. Nesse contexto considere a as regras da gramática abaixo (o símbolo de *start* é <Bl_Comando>) e implemente os procedimentos (em linguagem algorítmica) de um analisador sintático recursivo preditivo (ASRP):

```
<Bl_Comando> -> 'inicio' <Lista_Co> 'fim'
<Lista_Co>   -> <Comando> ';' <Lista_Co> | ε
<Comando>    -> <Bl_Comando> | <Atribuicao>
<Atribuicao> -> id '=' <Operando>
<Operando>   -> id | num
```

Obs.: id e num são classes de terminais

Questão 7) A fase de análise semântica normalmente é a ultima fase de análise de um compilador e tem a responsabilidade de realizar as validações semânticas (contextuais). De forma análoga a fase de geração de código é a última fase de síntese. Nesse contexto, implementar em pseudocódigo a lógica de *analisar e de gerar* de um nó do tipo NodeWhile com a regra a seguir (para a linguagem SPP – Script de Pascal em Português):

```
<While> ::= 'enquanto' '(' <Exp_Rel> ')' '{' <List_Coman> '}'
```

Obs2.: o pseudocódigo de *analisar(Node no) e gerar(Node no)* dos não terminais <List_Coman> e <Exp_Rel> podem ser considerados já implementados.

Questão 8) Crie a(s) regra(s) gramatical(is) para um comando SWITCH-CASE (estilo C) no estilo do SLI (Script de List em Italiano) e implemente os métodos de análise semântica (*analisar*), de interpretação (*interpretar*) e de geração de código (*gerar*) para o nó do tipo No_For.

Questão 9) Definir a gramatica do SS-SQL (Super Simple SQL) onde um arquivo SS-SQL pode conter 0 (zero) ou mais comandos SS-SQL, cada um separado por um ';' (ponto e vírgula). Cada comando SS-SQL pode ser (em qualquer ordem):

- criar uma entidade (com chave primaria e chave estrangeira);
- deletar uma entidade;
- inserir registro;
- atualizar registro;
- deletar registro;
- selecionar registros (com clausula WHERE e ORDER BY pelo menos).