- Prova de Compiladores - 1/2002 - 10/07/2002 -
==========

Para as questões abaixo, utilize as seguintes alternativas como resposta:

a) análise semântica
b) gerador de código intermediário
c) gerador de código
d) análise sintática
e) análise léxica
f) otimizador de código

1) Quais as fases que fazem parte do front-end de um compilador?

2) Quais as fases que fazem parte do back-end de um compilador?

3) Quais as fases que interagem com o gerenciador da tabela de símbolos (i.e. fazem uso da tabela de símbolos)?

4) Quais fases se comunicam pela solicitação e envio de *tokens*?

5) Em que fase se verifica se o número de argumentos passados para uma função está correto?

6) Qual a fase responsável por verificar se uma determinada seqüência de caracteres é ou não um identificador válido?

7) Qual a fase responsável por verificar se os tipos da variável e da expressão em uma atribuição (i.e. “x = 10 + y”) estão corretos?

8) Qual a fase que normalmente é especificada através de gramáticas livres de contexto?

9) Qual a fase responsável por verificar se apareceu um “else” sem antes aparecer um “if”?

10) Qual a fase normalmente especificada através de expressões regulares?

11) Qual a fase responsável por verificar se uma variável foi definida antes de ser utilizada?

12) Qual a fase responsável por verificar se uma determinada seqüência de caracteres é ou não um identificador válido?

13) A expressão [A-Za-z]+:
   a) Define uma seqüência de zero ou mais letras maiúsculas ou minúsculas
   b) Define ma seqüência de zero ou mais repetições da seqüência de caracteres “A-Za-z”
   c) É equivalente à expressão [A-Za-z][A-Za-z]*
   d) Define uma seqüência de um ou mais letras maiúsculas ou minúsculas
   e) É uma expressão regular

14) Árvores sintáticas servem para:
   a) comunicação entre o Analisador sintático e o Analisador léxico
   b) Podem ser usadas como estruturas intermediárias entre o analisador sintático e o analisador semântico
   c) Normalmente são criadas pelo analisador sintático.
   d) Normalmente são criadas pelo analisador léxico.
   e) comunicação entre o Analisador léxico e o gerador de código intermediário

15) Indique as afirmações verdadeiras
   a) Os compiladores funcionam através de um mecanismo de análise e síntese.
   b) Yacc é um exemplo de gerador de analisador sintático a partir de expressões regulares.
   c) Lex é um gerador de analisador léxico.
   d) coerção é a transformação implícita ou explícita de um valor de um tipo em outro tipo.
   e) Interpretadores e editores de texto dirigidos por sintaxe são exemplos de programas focados em análise de estruturas.

16) Indique as afirmações verdadeiras
   a) Os parsers top-down não tem problemas em relação a gramáticas recursivas à esquerda.
   b) Yacc gera parsers bottom-up, que são mais eficientes.
   c) os parsers bottom-up são normalmente gerados por ferramentas.
   d) Recursive descent parsers são um exemplo de parser top-down.
   e) É relativamente fácil escrever um parser top-down manualmente, usando funções recursivas.

17) Definições dirigidas por sintaxe:
   a) em alguns casos podem ser usadas para realizar a verificações semânticas (por exemplo de tipos).
   b) Podem ser usadas para construir a árvore sintática da linguagem no compilador.
   c) Suporta atributos sintetizados e herdados.
   d) São suportadas pelo Yacc.
   e) permitem associar ações/regras semânticas à especificação sintática de uma linguagem.

18) Podemos afirmar sobre a verificação de tipos:
   a) No overloading o mesmo código é usado para vários tipos de dados diferentes.
   b) No polimorfismo o mesmo código é usado para vários tipos de dados diferentes.
   c) Na verificação dinâmica de tipos é preciso manter informações de tipo durante a execução do programa
   d) A verificação de tipos pode ser especificada formalmente.
   e) Ela pode garantir estaticamente que todos os acessos a arrays são feitos dentro dos limites do tamanho do array.

19) Podemos afirmar sobre a gramática abaixo que:

```
Program : Statement ";" Program
          | /* producao vazia */
          ;

Statement : "if" Expression "then" Statement "else" Statement
          | identifier "=" Expression
          ;

Expression : identifier
           | number
           | identifier "+" Expression
           | number "+" Expression
           ;
```

a) Ela Não é ambígua
b) Ela é ambígua devido à produção Program
c) Ela é ambígua devido à produção Statement
d) Ela é ambígua devido à produção Expression
e) Ela possui produções com recursão à esquerda.

20) Ainda sobre a gramática acima, podemos afirmar que ela reconhece os seguintes programas:

a) `if cond then x = 10 else x = 20 + 3 ;`
b) `if x + 3 then 4 + 1 else 7 ;`
c) `if cond1 then if cond2 then x = 10 else x = 12 ;`
d) `x = y + z + 4 ;`
e) `x = 20; y = 10`