01. [Sebesta, 2000] Defina sintaxe e semântica.

02. [Sebesta, 2000] A quem se destinam as descrições de linguagem?

03. [Sebesta, 2000] Descreva a operação de um gerador de linguagem geral.

04. [Sebesta, 2000] Descreva a operação de um reconhecedor de linguagem geral.

05. [Sebesta, 2000] Qual é a diferença entre uma sentença e uma forma semântica?

06. [Sebesta, 2000] Defina uma regra de gramática recursiva à esquerda.

07. [Sebesta, 2000] Quais três extensões são comuns à maioria das EBNFs?

08. [Sebesta, 2000] Descreva semântica estática e dinâmica.

09. [Sebesta, 2000] A qual propósito servem os predicados em uma gramática de atributos?

10. [Sebesta, 2000] Qual é a diferença entre um atributo sintetizado e um herdado?

11. [Sebesta, 2000] Qual é a ordem de avaliação de atributos determinada para as árvores de determinada gramática de atributos?

12. [Sebesta, 2000] Qual é o principal uso das gramáticas de atributos?

13. [Sebesta, 2000] Qual é o problema quando se usa um interpretador puro de software para a semântica operacional?

14. [Sebesta, 2000] Explique o que as pré-condições e as pós-condições de determinada instrução significam na semântica axiomática.

15. [Sebesta, 2000] Descreva a abordagem de usar a semântica axiomática para provar a exatidão de determinado programa.

16. [Sebesta, 2000] Descreva o conceito básico da semântica denotacional.

17. [Sebesta, 2000] De que maneira a semântica operacional e a semântica denotacional diferem?

18. [Sebesta, 2000] Os dois modelos matemáticos de descrição de linguagens são geração e reconhecimento. Descreva como cada um pode definir a sintaxe de um linguagem de programação.

19. [Sebesta, 2000] Escreva descrições BNF e de grafo de sintaxe para cada item:

a) Uma instrução de cabeçalho (*header*) **procedure** Pascal.

b) Uma instrução de chamada **procedure** Pascal.

c) Uma instrução **switch** C.

d) Uma definição **union** C.

e) Literais **float** C.

24. [Sebesta, 2000] Usando a gramática a seguir, mostre uma árvore de análise e uma derivação à extrema esquerda das seguintes instruções:

a) `A := A * (B + (C * A))`

b) `B := C * (A * C + B)`

c) `A := A * (B + (C))`

```
G = ({atribuição, id, expr}, {:=, +, *, (, ), A, B, C}, P, atribuição)
P = { < atribuição > → < id > := < expr >
      < id >         → A | B | C
      < expr >       → < id > + < expr > | < id > * < expr >
                     | ( < expr > ) | < id >   }
```

25. [Sebesta, 2000] Usando a gramática a seguir, mostre uma derivação à extrema esquerda das seguintes instruções:

a) `A := (A + B) * C`

b) `A := B + C + A`

c) `A := A * (B + C)`

d) `A := B * (C * (A + B))`

```
G = ({atribuição, id, expr, termo, fator}, {:=, +, *, (, ), A, B, C}, P,
      atribuição)
P = { < atribuição > → < id > := < expr >
      < id >         → A | B | C
      < expr >       → < expr > + < expr > | < termo >
      < termo >      → < termo > * < fator > | < fator >
      < fator >      → ( < expr > ) | < id >   }
```

26. [Sebesta, 2000] Prove que a seguinte gramática é ambígua:

```
G = ({S, A, id}, {+, a, b, c}, P, S)
P = { < S >  → < A >
      < A >  → < A > + < A > | < id >
      < id > → a | b | c   }
```

27. [Sebesta, 2000] Modifique a gramática a seguir para adicionar um operador unitário que tenha uma precedência mais alta do que + ou *.

```
G = ({atribuição, id, expr, termo, fator}, {:=, +, *, (, ), A, B, C}, P,
      atribuição)
P = { < atribuição > → < id > := < expr >
      < id >         → A | B | C
      < expr >       → < expr > + < expr > | < termo >
      < termo >      → < termo > * < fator > | < fator >
      < fator >      → ( < expr > ) | < id >   }
```

28. [Sebesta, 2000] Descreva, em português, a linguagem definida pela seguinte gramática:

```
G = ({S, A, B, C}, {a, b, c}, P, S)
P = { < S > → < A > < B > < C >
      < A > → a < A > | a
      < B > → b < B > | b
      < C > → c < C > | c   }
```

29. [Sebesta, 2000] Considerando a gramática a seguir, quais das seguintes sentenças estão na linguagem gerada por essa gramática?

```
G = ({S, A, B}, {a, b}, P, S)
P = { < S > → < A > a < B > b
      < A > → < A > b | b
      < B > → a < B > | a   }
```

a) `baab`

b) `bbbab`

c) `bbaaaaa`

d) `bbaab`

30. [Sebesta, 2000] Considerando a gramática a seguir, qual das seguintes sentenças estão na linguagem gerada por essa gramática?

```
G = ({S, A, B}, {a, b, c, d}, P, S)
P = { < S > → a < S > c < B > | < A > | b
      < A > → c < A > | c
      < B > → d | < A >   }
```

a) `abcd`

b) `acccbd`

c) `acccbcc`

d) `acd`

e) `accc`

31. [Sebesta, 2000] Escreva uma gramática para a linguagem consistindo em *strings* que têm `n` cópias da letra `a` seguida do mesmo número de cópias da letra `b`, em que `n > 0`. Por exemplo, as *strings* `ab`, `aaaabbbb` e `aaaaaaaabbbbbbbb` estão na linguagem, mas `a`, `abb`, `ba` e `aaabb` não estão.

32. [Sebesta, 2000] Desenhe árvores de análise para as sentenças `aabcc` e `aaabbbc`, como derivadas da gramática produzida no exercício 31.

33. [Sebesta, 2000] Usando as instruções da máquina virtual definida a seguir, apresente uma definição semântica operacional do seguinte:

```
ident := var
ident := ident + 1
ident := ident - 1
ident := var op_bin var
ident := op_un var
goto label
if var relop var goto label
```

`ident` – é um identificador.
`var` – é um identificador ou uma constante.
`op_bin` – pode ser um dos operadores aritméticos do conjunto {+, -, *, /}.
`op_un` – pode ser um dos operadores unários do conjunto {+, -}.
`relop` – pode ser um dos operadores relacionais do conjunto {=, <>, >, <, >=, <=}.

a) repeat do Pascal

b) for-downto do Pascal

c) DO do FORTRAN da forma: DO N K = start, end, step

d) if-then-else do Pascal

e) switch do C

34. [Sebesta, 2000] Compute a pré-condição mais fraca para cada uma das seguintes instruções e pós-condições de atribuição:

a) `a := 2 * (b - 1) - 1 {a > 0}`

b) `b := (c + 10) / 3 {b > 6}`

c) `a := a + 2 * b - 1 {a > 1}`

d) `x := 2 * y + x - 1 {x > 11}`

35. [Sebesta, 2000] Compute a pré-condição mais fraca para cada uma das seguintes sequencias das instruções de atribuição e suas pós-condições:

a) `a := 2 * b + 1; b := a - 3; {b < 0}`

b) `a := 3 * (2 * b + a); b := 2 * a - 1; {b > 5}`

36. [Sebesta, 2000] Escreva a função de correspondência semântica denotacional para as seguintes instruções:

a) for do Pascal

b) repeat do Pascal

c) expressões booleanas do Pascal

d) for do C

e) switch do C

37. [Sebesta, 2000] Qual é a diferença entre um atributo intrínseco e um atributo sintetizado?

38. [Sebesta, 2000] Escreva uma gramática de atributos cuja base BNF seja a do Exemplo 3.6, mas cujas regras de linguagem sejam as seguintes: tipos de dados não podem ser misturados em expressões, mas as instruções de atribuição não precisam ter os mesmos tipos em ambos os lados do operador de atribuição.

Exemplo3.6: Uma Gramática de Atributos para Instruções de Atribuição

1. Regra de sintaxe: `<atribuição> → <var> := <expr>`
Regra semântica: `<expr>.tipo_esperado ← <var>.tipo_efetivo`

2. Regra de sintaxe: `<expr> → <var>[2] + <var>[3]`
Regra semântica:
```
<expr>.tipo_efetivo ←
  if(<var>[2].tipo_efetivo = int) e
    (<var>[3].tipo_efetivo = int)
      then int
      else real
  end if
```
Predicado: `<expr>.tipo_efetivo = <expr>.tipo_esperado`

3. Regra de sintaxe: `<expr> → <var>`
Regra semântica: `<expr>.tipo_efetivo ← <var>.tipo_efetivo`
Predicado: `<expr>.tipo_efetivo = <expr>.tipo_esperado`

4. Regra de sintaxe: `<var> → A | B | C`
Regra semântica: `<var>.tipo_efetivo ← look-up(<var>.string)`

A função *look-up* pesquisa determinado nome de variável na tabela de símbolos e retorna o tipo desta.

39. [Sebesta, 2000] Escreva uma gramática de atributos cuja base BNF seja a do Exemplo 3.2 e cujas regras de tipo sejam as mesmas do exemplo de instrução de atribuição do Exemplo 3.6.

Exemplo 3.2: Uma Gramática para Instruções de Atribuição Simples

```
G = ({atribuição, id, expr}, {:=, +, *, (, ), A, B, C}, P, atribuição)
P = { < atribuição > → < id > := < expr >
      < id >         → A | B | C
      < expr >       → < id > + < expr > | < id > * < expr >
                     | ( < expr > ) | < id >   }
```

Exemplo3.6: Uma Gramática de Atributos para Instruções de Atribuição

1. Regra de sintaxe: `<atribuição> → <var> := <expr>`

Regra semântica: `<expr>.tipo_esperado ← <var>.tipo_efetivo`

2. Regra de sintaxe: `<expr> → <var>[2] + <var>[3]`

Regra semântica:

```
<expr>.tipo_efetivo ←
  if(<var>[2].tipo_efetivo = int) e
    (<var>[3].tipo_efetivo = int)
      then int
      else real
  end if
```

Predicado: `<expr>.tipo_efetivo = <expr>.tipo_esperado`

3. Regra de sintaxe: `<expr> → <var>`

Regra semântica: `<expr>.tipo_efetivo ← <var>.tipo_efetivo`

Predicado: `<expr>.tipo_efetivo = <expr>.tipo_esperado`

4. Regra de sintaxe: `<var> → A | B | C`

Regra semântica: `<var>.tipo_efetivo ← look-up(<var>.string)`

A função *look-up* pesquisa determinado nome de variável na tabela de símbolos e retorna o tipo desta.

40. [Sebesta, 2000] Prove que o seguinte programa está correto:

```
{x = Vx e y = Vy}
temp = x;
x = y;
y = temp;
{x = Vy e y = Vx}
```

41. [Sebesta, 2000] Prove que o seguinte programa está correto:

```
{n > 0}
cont = n;
soma = 0;
while cont <> 0 do
  soma = soma + cont;
  cont = cont - 1;
end
{soma = 1 + 2 + ... + n}
```

42. Usando a gramática a seguir, mostre uma derivação à extrema esquerda, uma derivação à extrema direita e uma árvore de análise da instrução `A = B * C + (B + C).`

```
G = ({atr, exp, ter, fat, id}, {A, B, C, +, *, =, (, )}, P, atr)
P = { <atr> ::= <id> = <exp>
      <exp> ::= <exp> + <ter> | <ter>
      <ter> ::= <ter> * <fat> | <fat>
```

```
<fat> ::= ( <exp> ) | <id>
<id>  ::= A | B | C }
```