Segunda Lista de Exercícios de Construção de Compiladores.
Primeiro Semestre de 2001. Departamento de Computação – UFSCar.
José de Oliveira Guimarães.

Muitas das questões desta lista admitem respostas subjetivas. As questões das provas serão mais exatas e com mais informações.

1. Defina sistema de tempo de execução. Cite algumas de suas funções.

2. Por que código interpretado é mais seguro para os outros programas e para o sistema de arquivos do que código compilado ?

3. Cite as razões pelas quais interpretadores são mais simples de fazer do que compiladores.

4. Cite um motivo para se usar um compilador ao invés de um interpretador.

5. Cite três aplicações de técnicas e algoritmos de compilação em programas que não sejam compiladores ou interpretadores.

6. Defina *front end* e *back end*. Cite as fases da compilação que fazem parte de cada uma delas.

7. Temos um compilador de C que gera código para o Pentium e queremos construir um outro compilador, baseado neste, que gere código para um outro processador. Que fases do compilador devemos modificar ?

8. O que é uma sentença de uma gramática G ?

9. Como é definida a linguagem gerada por uma gramática G ?

10. O que significa cada uma das linhas

$$S \stackrel{+}{\Longrightarrow} \alpha$$
$$w \in \Sigma \cup N$$
$$H(G) = \{ w \notin \Sigma^* \mid S \stackrel{+}{\Longrightarrow} w \}$$

?

11. Uma sentença de uma linguagem L(G) pode ser obtida por derivações que não são nem todas à esquerda nem todas à direita ?

12. Prove que a gramática

S ::= "if" E "then" S EL | id
EL ::= "else" S | ϵ
E ::= id | Numero

é ambígua. id e Numero são terminais.

13. Faça uma gramática de expressões com os operadores binários $+$, $-$, $*$ e os operadores unários $\hat{}$

e & de tal forma que :

- $+$ tenha maior precedência do que $-$ e $*$ e seja associativo à esquerda;
- $-$ tenha maior precedência do que $*$ e seja associativo à direita;
- $*$ seja associativo à direita;
- & tenha menor precedência que todos os outros operadores e seja associativo à direita;
- ^ tenha maior precedência do que todos os outros operadores e seja associativo à esquerda.

14. Que condições uma gramática deve obedecer para que ela possa ser utilizada para análise sintática descendente preditiva ? Por que estas condições são necessárias para a análise preditiva ?

15. Retire a recursão à esquerda das gramáticas mostradas a seguir. Dê exemplos de sentenças geradas por cada gramática.

1. IdL ::= IdL Id | ϵ
2. E ::= E E (+|−|∗|/)
   E ::= Numero
3. E ::= E “or” T | T
   T ::= T “and” F | F
   F ::= “true” | “false” | “(” E “)”

16. Fatore à esquerda as gramáticas abaixo

1. S ::= Aa | Ab | c | d
   A ::= BA | Bh
   B ::= f | gA
2. E ::= T “or” E | T
   T ::= F “and” T | F
   F ::= “true” | “false” | “(” E “)”

17. É possível eliminar a recursão à esquerda da gramática

S ::= Aa | b
A ::= Ac | Sd | E

com as regras dadas em aula ?

18. Prepare a gramática

E ::= E “or” T | T

T ::= T “and” F | F
F ::= “true” | “false” | “(” E “)”

para análise sintática preditiva. Após isto, faça a análise da sentença

```
true or false and true
```

passo a passo.

19. Para cada item a seguir, faça uma gramática capaz de gerar a sentença apresentada.

1. ```
   var a, b;
   a = 1, b = 2,
   c = 3.
   ```

2. ```
   var a = 1, b = 2;
   2*a + b/2 - 1;
   ```

3. ```
    1 a =, 2 b = ; 2 a * b 2 / + 1 - ;
   ```

20. Para cada uma das descrições a seguir, faça a expressão regular do JLex que a reconheça, o autômato finito correspondente e a implementação do autômato em Java.

1. Palavra que começa por letra maiúscula, possui qualquer número de letras minúsculas em seguida.
2. Comentário começando por { que pode se seguir por várias linhas e termina por }. Não pode haver nenhum dígito dentro do comentário.
3. Número Float como reconhecido por Java.
4. Palavra composta por letras seguida por espaço em branco seguido de ponto, como “fim .”.
5. String em Java. Não considere os caracteres de escape, mas considere que a string pode ter “;” dentro dela.

21. O que reconheçem as expressões regulares abaixo ?

1. `[A-Z0-9][a-z]*(^|~)[^0-5]`
2. `\-(r|w|x)(r|w|x)(r|w|x)" "+[0-9]" "+[A-Za-z]*.+`
3. `[A-Z]:(\\[A-Za-z0-9]+)+`
4. `^$`
5. `([0-9]+" "*("*" | "+" | "/" | "-")" "*[0-9]+)+`
   ```