Respostas da Primeira Prova de Compiladores I - 2012.1

1) Expressões regulares não conseguem expressar aninhamento arbitrário, então não é possível especificar o analisador léxico usando apenas expressões regulares.

2) Solução recursiva:

```
void leComentario() {
  nextChar();  // pula o '{' inicial
  while(true) {
    if(lookAhead == -1)
      throw new RuntimeException("comentário não terminado");
    switch((char)lookAhead) {
      case '{':
        leComentario();
        continue;
      case '}':
        nextChar();
        return;
      default:
        nextChar();
        continue;
    }
  }
}

// Reescrita do código original
case '{':
  leComentario();
  continue;
```

Solução iterativa:

```
case '{':
  int nivel = 1;
  nextChar();
  while(true) {
    if(lookAhead == -1)
      throw new RuntimeException("comentário não terminado");
    switch((char)lookAhead) {
      case '{':
        nextChar();
        nivel++;
        continue;
      case '}':
        nextChar();
        nivel--;
        if(nivel == 0) then break;  // sai do while(true)
      default:
        nextChar();
        continue;
    }
  }
  continue;
```

3)

(a) S -> aS -> aAb -> aXYZb -> aYZb -> aZb -> aeSb -> aeAbb -> aebb

(b) A inclusão da produção X -> bS causa um conflito LL(1) no não-terminal A, pois b já está no FIRST+(A -> epsilon), e a nova produção incluiria b também no FIRST+(A -> XYZ).

4)
(b) Recursão à esquerda ou ambiguidade.

(c) Desenhar duas árvores sintáticas para

```
begin
  write num + num + num
end
```

(d)

E -> E + T | E - T | T
T -> num | id | (E)

ou vale a mesma resposta dada em (e), com um comentário de que gramáticas LL(1) não são ambíguas!

(e)

LISTACMD -> CMD LISTACMD'
LISTACMD' -> CMD LISTACMD' | epsilon

FIRST+(LISTACMD' -> CMD LISTACMD') = { do, read, write, id }
FIRST+(LISTACMD' -> epsilon) = { end }, que é o FOLLOW(LISTACMD') que é o FOLLOW(LISTACMD)

FIRST+(CMD -> do …) = { do }
FIRST+(CMD -> read id) = { id }
FIRST+(CMD -> write EXP) = { write }
FIRST+(CMD -> id := EXP) = { id }

EXP -> T EXP'
EXP' -> + T EXP' | - T EXP' | epsilon
T -> id | num | (EXP)

FIRST+(EXP' -> + T EXP') = { + }
FIRST+(EXP' -> - T EXP') = { - }
FIRST+(EXP' -> epsilon) = {), end, do, read, write, id }, via FOLLOW(EXP) e FOLLOW(CMD)

FIRST+(T -> num) = { num }
FIRST+(T -> id) = { id }
FIRST+(T -> (EXP)) = { (}