Construção de Compiladores
Daniel Lucrédio, Helena Caseli, Mário César San Felice e Murilo Naldi
Tópico 06 - Análise Sintática Conclusão - Lista de Exercícios Resolvida
(Última revisão: fev/2020)

1. Dada a gramática a seguir

```
S : ( L ) | a
L : L , S | S
```

a) Elimine a recursividade à esquerda.

R:

```
S : '(' L ')' | 'a'
L : S L2
L2 : ',' S L2 | <<vazio>>
```

b) Construa os procedimentos recursivos para a gramática obtida na letra a) bem como o programa principal, utilizando pseudocódigo.

R:

```
void s() {
   if(la(1) == '(') {
      match('(')
      l()
      match(')')
   } else {
      match('a')
   }
}

void l() {
   s()
   l2()
}

void l2() {
   if(la(1) == ',') {
      match(',')
      s()
      l2()
   } else {
      // vazio
   }
}

void principal() {
   s()
}
```

2. Construa os procedimentos recursivos e o programa principal, usando pseudocódigo, para a seguinte gramática:

```
comando : 'while' '(' expr ')' comando 'endwhile' comando2
comando2 : ';' listaComandos comando2 | ';' 'if' '(' expr ')' 'then'
           comando | <<vazio>>
expr : termo expr2
expr2 : '+' termo expr2 | '-' termo expr2 | <<vazio>>
termo : fator termo2
termo2 : '*' fator termo2 | '/' fator termo2 | <<vazio>>
fator : VAR fator2
fator2: '.' VAR fator2 | <<vazio>>
listacomandos : '{' comando '}'
```

R:

```
void comando() {
   match('while')
   match('(')
   expr()
   match(')')
   comando()
   match('endwhile')
   comando2()
}

void comando2() {
   match(';') // o ponto e vírgula vem antes do teste, pois
              // é comum a ambas as alternativas
              // outra solução seria fatorar a gramática
   if(la(1) == '{') {
      listaComandos()
      comando2()
   } else if(la(1) == 'if') {
      match('if')
      match('(')
      expr()
      match(')')
      match('then')
      comando()
   } else {
      // vazio
   }
}
void expr() {
   termo()
   expr2()
}
void expr2() {
   if(la(1) == '+') {
      match('+')
```

```
        termo()
        expr2()
    } else if(la(1) == '-') {
        match('-')
        termo()
        expr2()
    } else {
        // vazio
    }
}
void termo() {
    fator()
    termo2()
}
void termo2() {
    if(la(1) == '*') {
        match('*')
        fator()
        termo2()
    } else if(la(1) == '/') {
        match('/')
        fator()
        termo2()
    } else {
        // vazio
    }
}
void fator() {
    match(TipoToken.VAR)
    fator2()
}
void fator2() {
    if(la(1) == '.') {
        match('.')
        match(TipoToken.VAR)
        fator2()
    } else {
        // vazio
    }
}
listacomandos : '{' comando '}'
void listaComandos() {
    match('{')
    comando()
    match('}')
}
void principal() {
    comando()
}
```

3. Construa os procedimentos recursivos e o programa principal, usando pseudocódigo, para a

seguinte gramática:

```
comandoCondicao    :    'SE'    expressaoRelacional    'ENTAO'    comando
            comandoSenao 'FIMSE'
comandoSenao : 'SENAO' comando | <<vazio>>
```

R:

```
void comandoCondicao() {
   match('SE')
   expressaoRelacional()
   match('ENTAO')
   comando()
   comandoSenao()
   match('FIMSE')
}
void comandoSenao() {
   if(la(1) == 'SENAO') {
      match('SENAO')
      comando()
   } else {
      // vazio
   }
}
void principal() {
   comandoCondicao()
}
```

4. Construa os procedimentos recursivos e o programa principal, usando pseudocódigo, para a seguinte gramática:

```
lexp : atomo | lista
atomo : numero | identificador
lista : ( lexp+ )
```

R:

```
void lexp() {
   if(la(1)==TipoToken.numero | la(1)==TipoToken.identificador) {
      atomo()
   } else if(la(1) == '(') {
      lista()
   }
}
void atomo() {
   if(la(1) == TipoToken.numero) {
      match(TipoToken.numero)
   } else {
      match(TipoToken.identificador)
   }
}
void lista() {
```

```
   match('(')
   while(la(1) != ')') {
      lexp()
   }
   match(')')
}
void principal() {
   lexp()
}
```