Randy Quindai
Exercícios de Compiladores

1) Pseudocódigo das funções de um analisador descendente preditivo recursivo, para a linguagem dada em sala.

PGM = LDECL '::' LSENT'.'
LDECL = DECL LDECLR
LDECLR = ';' DECL LDECLR | ε
DECL = Lid ':' TIPO
Lid = 'id' LIDR
LIDR = ',' 'id' LIDR | ε
TIPO = TBASE TIPOF
TIPOF = '[' ctel ']' | ε
TBASE = 'INT' | 'REAL'
LSENT = SENT LSENTR
LSENTR = ';'SENT LSENTR
SENT = ATR | COND | ITER
ATR = 'id' '=' Ea
COND = 'SE' Eb 'ENTAO' LSENT CONDF
CONDF = 'SENAO' LSENT 'FIM' | 'FIM'
ITER = 'PARA' ATR 'ATE' Ea PASSO 'FACA' LSENT 'FIM' | 'ENQUANTO' Eb 'FACA' LSENT 'FIM' | 'REPITA' LSENT 'ENQUANTO' Eb 'FIM'
PASSO = 'PASSO' EA | ε

resposta

a função `le_token` retorna um token lido a partir da sentença de entrada, e volta o ponteiro de leitura para o último ponto marcado

```
void PGM(){
  LDECL();
  if (nextToken == "::") {
    le_token();
    LSENT();
  } else if (token == ".") {
        le_token();
  } else ERRO(1);
}
```

```
void LDECL(){
  DECL();
  LDECLR();
}

void TIPO(){
  TBASE();
  TIPOF();
}
```

```
void DECL(){
  Lid();
  if(nextToken == ":"){
    le_token();
    TIPO();
  } else ERRO(2);
}
```

```
void LDECLR(){
  if(nextToken == ";"){
    le_token();
    DECL();
    LDECLR();
  } else ERRO(3);
  return;
}
```

```
void Lid(){
```

```
void LIDR(){
```

```
  if(nextToken == "id"){
    le_token();
    LIDR();
  } else ERRO(4);
}
```

```
  if(nextToken == ","){
    le_token();
    if(nextToken == "id"){
      LIDR();
    } else ERRO(5);
  } else return;
}
```

```
void LSENT(){
  SENT();
  LSENTR();
}

void SENT(){
  if(nextToken=="id"){
   ATR();
  }else if(nextToken=="PARA" ||
nextToken=="ENQUANTO" ||
nextToken=="REPITA") {
    ITER();
  }else if(nextToken=="SE"){
    COND();
  }
}
```

```
void TIPOF(){
  if(nextToken == "["){
    le_token();
    if(nextToken == "cteI"){
      le_token();
      if(nextToken == "]"){
        le_token();
      } else ERRO(5);
    } else ERRO(6);
  } else return;
}
```

```
void TBASE(){
  if(nextToken == "int"){
    le_token();
  }else if(nextToken =="real"){
    le_token();
  }else ERRO(7);
}
```

```
void LSENTR(){
  if(nextToken == ";"){
    le_token();
    SENT();
    LSENTR();
  } else ERRO(3);
}
```

```
void COND(){
  if(nextToken=="SE"){
   le_token();
   Eb();
   if(nextToken=="ENTAO"){
    le_token();
    LSENT();
    CONDF();
   } else ERRO(9);
  } else ERRO(10)
}
```

```
void ATR(){
  if(nextToken == "id"){
    le_token();
    if(nextToken == "="){
     le_token();
     Ea();
    } else ERRO(8);
  } else ERRO(5);
}
```

```
void ITER(){
  if(nextToken=="PARA"){
   le_token();
   ATR();
   if(nextToken=="ATE"){
```

```
void CONDF(){
  if(nextToken=="SENAO"){
   le_token();
   LSENT();
   if(nextToken=="FIM"){
```

```
    le_token();
    Ea();
    PASSO();
    if(nextToken==”FACA”){
     le_token();
     LSENT();
     if(nextToken==”FIM”){
      le_token();
     } else ERRO(11);
    } else ERRO(12);
   } else ERRO(13);
  }
else if(nextToken==”ENQUANTO”){
  Eb();
  if(nextToken==”FACA”){
    le_token();
    LSENT();
   if(nextToken==”FIM”){
    le_token();
   }else ERRO(11);
  }else ERRO(12);
}else if(nextToken==”REPITA”){
  le_token();
  LSENT();
  if(nextToken==”ENQUANTO”){
   le_token();
   Eb();
   if(nextToken==”FIM”){
    le_token();
   } else ERRO(11);
  } else ERRO(14);
}else ERRO(15);
}
```

```
    le_token();
   } else ERRO(11);
  }else if(nextToken==”FIM”){
    le_token();
  } else ERRO(11);
}
```

2) Construção da tabela de análise SLR para a gramática abaixo:

(0) S = Ea
(1) Ea = Ea 'opa' Ta
(2) Ea = Ta
(3) Ta = Ta 'opm' Pa
(4) Ta = Pa
(5) Pa = Fa '' Pa**
(6) Pa = Fa
(7) Fa = '(' Ea ')'
(8) Fa = 'id'
(9) Fa = 'cten'

resposta

Estado Expressão

0 = {S=.Ea, Ea=.Ea'opa'Ta, Ea=.Ta, Ta=.Ta'opm'Pa, Ta=.Pa, Pa=.Fa'**'Pa, Pa=.Fa, Fa=.'('Ea')', Fa=.'id', Fa=.'cten'}

1 = goto(0, Ea) = {S=Ea., Ea=Ea.'opa'Ta}

2 = goto(0, Ta) = {Ea=Ta., Ta=Ta.'opm'Pa}

3 = goto(0, Pa) = {Ta=Pa.}

4 = goto(0, Fa) = {Pa=Fa.'**'Pa, Pa=Fa.}

5 = goto(0, '(') = {Fa='('.Ea')', Ea=.Ea'opa'Ta, Ea=.Ta, Ta=.Ta'opm'Pa, Ta=.Pa, Pa=.Fa'**'Pa, Pa=.Fa, Fa=.'('Ea')', Fa=.'id', Fa=.'cten'}

6 = goto(0, 'id') = {Fa='id'.}

7 = goto(0, 'cten') = {Fa='cten'.}

8 = goto(1, 'opa') = {Ea=Ea'opa'.Ta, Ta=.Ta'opm'Pa, Ta=.Pa, Pa=.Fa'**'Pa, Pa=.Fa, Fa=.'('Ea')', Fa=.'id', Fa=.'cten'}

9 = goto(2, 'opm') = {Ta=Ta'opm'.Pa, Pa=.Fa'**'Pa, Pa=.Fa, Fa=.'('Ea')', Fa=.'id', Fa=.'cten'}

10 = goto(4, '**') = {Pa=Fa'**'.Pa, Pa =.Fa'**'Pa, Pa=.Fa, Fa=.'('Ea')', Fa=.'id', Fa=.'cten'}

11 = goto(5, Ea) = {Fa='('Ea.')', Ea=Ea.'opa'Ta}

2 = goto(5, Ta) = {Ea=Ta., Ta=Ta.'opm'Pa}

3 = goto(5, Pa) = {Ta=Pa.}

4 = goto(5, Fa) = {Pa=Fa.'**'Pa, Pa=Fa.}

5 = goto(5, '(') = {Fa='('.Ea')', Ea=.Ea'opa'Ta, Ea=.Ta, Ta=.Ta'opm'Pa, Ta=.Pa, Pa=.Fa'**'Pa, Pa=.Fa, Fa=.'('Ea')', Fa=.'id', Fa=.'cten'}

6 = goto(5, 'id') = {Fa='id'.}

7 = goto(5, 'cten') = {Fa='cten'}

12 = goto(8, Ta) = {Ea=Ea'opa'Ta., Ta=Ta.'opm'Pa}

3 = goto(8, Pa) = {Ta=Pa.}

4 = goto(8, Fa) = {Pa=Fa.'**'Pa, Pa=Fa.}

5 = goto(8, '(') = {Fa='('.Ea')', Ea=.Ea'opa'Ta, Ea=.Ta, Ta=.Ta'opm'Pa, Ta=.Pa, Pa=.Fa'**'Pa, Pa=.Fa, Fa=.'('Ea')', Fa=.'id', Fa=.'cten'}

6 = goto(8, 'id') = {Fa='id'.}

7 = goto(8. 'cten') = {Fa='cten'.}

13 = goto(9, Pa) = {Ta=Ta'opm'Pa.}

4 = goto(9, Fa) = {Pa=Fa.'**'Pa, Pa=Fa.}

5 = goto(9, '(') = {Fa='('.Ea')', Ea=.Ea'opa'Ta, Ea=.Ta, Ta=.Ta'opm'Pa, Ta=.Pa, Pa=.Fa'**'Pa, Pa=.Fa, Fa=.'('Ea')', Fa=.'id', Fa=.'cten'}

6 = goto(9, 'id') = {Fa='id'.}

7 = goto(9, 'cten') = {Fa='cten'.}

14 = goto(10, Pa) = {Pa=Fa'**'Pa.}

4 = goto(10, Fa) = {Pa=Fa.'**'Pa, Pa=Fa.}

5 = goto(10, '(') = {Fa='('.Ea')', Ea=.Ea'opa'Ta, Ea=.Ta, Ta=.Ta'opm'Pa, Ta=.Pa, Pa=.Fa'**'Pa, Pa=.Fa, Fa=.'('Ea')', Fa=.'id', Fa=.'cten'}

6 = goto(10, 'id') = {Fa='id'.}

7 = goto(10. 'cten') = {Fa='cten'.}

15 = goto(11, ')') = {Fa='('Ea')'.}

8 = goto(11, 'opa') = {Ea=Ea'opa'.Ta, Ta=.Ta'opm'Pa, Ta=.Pa,Pa=.Fa'**'Pa, Pa=.Fa, Fa=.'('Ea')', Fa=.'id', Fa=.'cten'}

9 = goto(12, 'opm') = {Ta=Ta'opm'.Pa, Pa=.Fa'**'Pa, Pa=.Fa, Fa=.'('Ea')', Fa=.'id', Fa=.'cten'}

3) Adaptar a gramática abaixo para ser reconhecida por um analisador descendente e adicionar suas ações semânticas.

		Descendente	**Semântica**
(1) Res = Ea '=' **(2) Ea = Ea '+' Ta** **(3) Ea = Ea '-' Ta** **(4) Ea = Ta** **(5) Ta = Ta '*' Fa** **(6) Ta = Ta '/' Fa** **(7) Ta = Fa** **(8) Fa = '(' Ea ')'** **(9) Fa = 'cteN'**	**=>**	Res = Ea Resr Resr = ‘=’ Ea Resr|ε Ea = Ta Ear Ear = ‘+’ Ta Ear|’-’TaEar |ε Ta = Fa Tar Tar = ‘*’FaTar|’/’FaTar|ε Fa = ‘(‘ Ea ‘)’ Fa = ‘cteN’	Imprimir(Res.val) Ea.val=Ea1.val+Ta.val Ea.val=Ea2.val+Ta.val Ea.val=Ta.val Ta.val=Ta1.val*Fa.val Ta.val=Ta2.val/Fa.val Ta.val=Fa.val Fa.val=Ea.val Fa.val=cteN.lex

4) Codificar o analisador descendente preditivo recursivo da gramática abaixo:

(1) Calc = Ea '=' {printf("%f", Ea.val);}
(2) Ea = Ta {Ear.vh = Ta.val } Ear {Ea.val = Ear.vs}
(3) Ear = '+' Ta {Ear1.vh = Ear.vh + Ta.val} Ear {Ear.vs=Ear1.vs}
(4) Ear = '-' Ta {Ear1.vh = Ear.vh - Ta.val} Ear {Ear.vs=Ear1.vs}
(5) Ear = $ {Ear.vs = Ear.vh}
(6) Ta = Fa {Tar.vh = Fa.val} Tar {Ta.val = Tar.vs}
(7) Tar = '*' Fa {Tar1.vh = Tar.vh * Fa.val} Tar {Tar.vs=Tar1.vs}
(8) Tar = '/' Fa {Tar1.vh = Tar.vh / Fa.val} Tar {Tar.vs=Tar1.vs}
(9) Tar = $ {Tar.vs=Tar.vh}
(10) Fa = '(' Ea ')' {Fa.val = Ea.val}
(11) Fa = 'cteN' {Fa.val = atof(cteN.lex);}

resposta

```
void Calc(Queue<string> ts){
  token = ts.dequeue();
  if(token == “(“ || token == “cteN”){
   token = nextToken();
   if(token == “=”) {printf("%f", token.val);}
   if(token == “)” {token = nextToken();}
   else {Ea();}
  }
}

float Ear(Queue<string> ts){
  float valor = valorDoTermo(ts);
  while(token.topo ==”+” || token.topo==”-”){
```

```
    string op = ts.dequeue();
    int proxTerm = valorDoTermo(ts);
    if(op.equal(“+”)){
    valor = valor + proxTerm;
   }else {valor=valor-proxTerm;}
  }
 return valor;
}

float Tar(Queue<string> ts){
  int valor = valorDoTermo(ts);
  while(token.topo ==”*” || token.topo==”/”){
    string op = ts.dequeue();
    float proxTerm = valorDoTermo(ts);
    if(op.equal(“*”)){
    valor = valor * proxTerm;
   }else {valor=valor/proxTerm;}
  }
 return valor
}
```