Instituto Tecnológico de Aeronáutica

Curso de Engenharia da Computação

Disciplina CES-41: Compiladores

Relatório do Laboratório 3

Vitor Pimenta dos Reis Arruda

Professor Fábio Carneiro Mokarzel

São José dos Campos

26/03/2019

Sumário

1 Introdução

Este trabalho intenciona a escrita e o teste de um analisador sintático para a linguagem de programação COMP-ITA 2019 [1]. Implementa-se uma solução que processa um programa escrito na tal linguagem, captura sua árvore sintática, opera sobre ela e imprime uma nova versão do código-fonte reformatado (o chamado “pretty printer”).

Para o desenvolvimento, usou-se a linguagem de programação **typescript** com o analisador léxico **Moo** [3] e o analisador sintático **Nearley** [4]. Nota-se que o roteiro deste laboratório pede explicitamente ([2]) o uso da ferramenta **Yacc**, que não foi utilizada, como já explicado.

As seções seguintes se estruturam de maneira que a primeira explica o funcionamento básico do **Nearley** e as seguintes relatam o projeto do analisador específico desejado (para a linguagem COMP-ITA 2019).

2 Sintaxe da biblioteca Nearley

A operação do *Nearley* é similar à do *Yacc*: existe um arquivo que declara a estrutura da sintaxe usando gramática livre de contexto, com opcionais ações a serem executadas sempre que uma produção for processada (trecho de código delimitado por **{%** e **%}**). A Codificação 1 ilustra trechos do código implementado para processar a linguagem COMP-ITA 2019:

```
# Gramática em nearley para a linguagem COMPITA2019
# Este arquivo gera automaticamente o "grammar.ts" para uso programático

@preprocessor typescript

@{%
const lexer = require('../../lab2/lexer').lexer
%}

@lexer lexer

Prog -> %PROGRAM %ID %OPBRACE GlobDecls Functions %CLBRACE {%
    data => ({
        nodeName: 'Prog',
        nodeChildren: data
    })
%}

GlobDecls -> null {%
    data => ({
        nodeName: 'GlobDecls',
        nodeChildren: data
    })
%}
 | %GLOBAL %COLON DeclList {%
    data => ({
        nodeName: 'GlobDecls',
        nodeChildren: data
    })
%}
```

Codificação 1: Techos de sintaxe Nearley para capturar a árvore sintática de COMP-ITA 2019

A ação dessa gramática *Nearley* é ilustrada na Codificação 2, gerada por meio da aplicação da gramática a um código-fonte COMP-ITA 2019:

```
{
   "nodeName": "Prog",
   "nodeChildren": [{
      "type": "PROGRAM",
      "value": "program",
      "text": "program",
      "offset": 0,
      "lineBreaks": 0,
      "line": 1,
      "col": 1
   }, {
      "type": "ID",
      "value": "Math",
      "text": "Math",
      "offset": 8,
      "lineBreaks": 0,
      "line": 1,
      "col": 9
   }, {
      "type": "OPBRACE",
      "value": "",
      "text": "{",
      "offset": 13,
      "lineBreaks": 0,
      "line": 1,
      "col": 14
   }, {
      "nodeName": "GlobDecls",
      "nodeChildren": [{
         "type": "GLOBAL",
         "value": "global",
         "text": "global",
         "offset": 16,
         "lineBreaks": 0,
         "line": 3,
         "col": 1
      },
      {"..."
      }]
   }]
}
```

Codificação 2: Trecho da árvore sintática gerada a partir de um código-fonte COMP-ITA 2019 (não relatado)

3 Arquitetura do *pretty-printer*

Talvez a estrutura da árvore sintática supracitada seja suficiente para gerar um *pretty-printer*, porém, tendo em vista a pretensão futura de empreender ainda análise semântica e geração de código intermediário, optou-se por uma segunda transformação: converter a árvore sintática original “verbosa” em uma **árvore de sintaxe abstrata (AST)** , que se caracteriza por representar a mesma informação de maneira mais “enxuta”.

A estrutura da AST é formalizada pela Codificação 3. Para facilitar o entendimento, considere – como exemplo – que a linha "*declarations: Declaration[]*" dentro de *Program* significa que um nó do tipo *Program* tem um atributo chamado *declarations* cujo tipo é uma lista de outros nós do tipo *Declaration*.

Afirma-se (sem demonstração) que a AST é equivalente à árvore de sintaxe original ("verbosa").

```
1
2
3  // Definitions for the abstract
4  // syntax tree associated to COMPITA-2019
5
6  export type VariableType = 'int' | 'float' | 'char' | 'logic' | 'void'
7
8  export type ASTNode = Program | Declaration | Identifier | IFunction | Statement | Expression
9
10 // any list may be empty
11
12 export interface Program {
13   kind: 'program'
14   name: string
15   declarations: Declaration[] // 1 Declaration declares 1 (no more !) identifier
16   functions: IFunction[]
17 }
18
19 export interface Declaration {
20   kind: 'declaration'
21   type: VariableType
22   identifier: Identifier
23 }
24
25 export interface Identifier {
26   kind: 'identifier'
27   name: string
28   dimensions: number[], // empty for scala identifier
29   subscripted: boolean
30 }
31
32 export type IFunction = MainFunction | RegularFunction
33
34 export interface MainFunction {
35   kind: 'main'
36   declarations: Declaration[]
37   statements: Statement[]
38 }
39
40 export interface RegularFunction {
41   kind: 'function'
42   returnType: VariableType
43   name: string
44   arguments: { [key: string]: VariableType }
45   declarations: Declaration[]
46   statements: Statement[]
```

```
47 }
48
49 export type Statement = If | While | Do | For | Read | Write | Assignment | FunctionCall |
   Return
50
51 export interface If {
52   kind: 'if'
53   condition: Expression
54   ifBody: Statement[]
55   elseBody: Statement[] // will be empty if the "If" has no else part
56 }
57
58 export interface While {
59   kind: 'while'
60   condition: Expression
61   body: Statement[]
62 }
63
64 export interface Do {
65   kind: 'do'
66   condition: Expression
67   body: Statement[]
68 }
69
70 export interface For {
71   kind: 'for'
72   initializer: Assignment
73   condition: Expression
74   increment: Assignment
75   body: Statement[]
76 }
77
78 export interface Read {
79   kind: 'read'
80   receptors: IdentifierReference[]
81 }
82
83 export interface Write {
84   kind: 'write'
85   sources: WriteSource[]
86 }
87
88 export type WriteSource = IString | Expression
89
90 export interface Assignment {
91   kind: 'assignment'
92   leftSide: IdentifierReference
93   rightSide: Expression
94 }
95
96 export interface IdentifierReference {
97   kind: 'identifier reference'
```

```
  name: string
  subscripts: Expression[] // may be empty
}

/**
 * Does not differentiate CALL from other function calls. This
 * distinction is inferred from context of ocurrence of the call.
 */
export interface FunctionCall {
  kind: 'function call'
  name: string
  arguments: Expression[]
}

export interface Return {
  kind: 'return'
  body?: Expression
}

export type Expression = BooleanOperation | Arithmetic | Negation | Constant |
IdentifierReference | FunctionCall

export type BooleanOperation = LogicalOR | LogicalAND | LogicalNOT | Comparison

export interface LogicalOR {
  kind: 'or'
  leftSide: Expression
  rightSide: Expression
}

export interface LogicalAND {
  kind: 'and'
  leftSide: Expression
  rightSide: Expression
}

export interface LogicalNOT {
  kind: 'not'
  target: Expression
}

export interface Comparison {
  kind: 'comparison'
  operator: '<=' | '<' | '>=' | '>' | '=' | '!='
  leftSide: Expression
  rightSide: Expression
}

export interface Arithmetic {
  kind: 'arithmetic'
  operator: '+' | '-' | '*' | '/' | '%'
  leftSide: Expression
```

```
149   rightSide: Expression
150 }
151
152 export interface Negation {
153   kind: 'negation'
154   target: Expression
155 }
156
157 export type Constant = IBoolean | Char | Int | Float
158
159 export interface IBoolean {
160   kind: 'boolean'
161   value: boolean
162 }
163
164 export interface Char {
165   kind: 'character'
166   intValue: number
167   stringValue: string
168   codeValue: string // value as would appear in code
169 }
170
171 export interface Int {
172   kind: 'integer'
173   value: number
174 }
175
176 export interface Float {
177   kind: 'float'
178   value: number
179 }
180
181 // although this is a literal, it is not considered a "Constant"
182 export interface IString {
183   kind: 'string'
184   value: string
185   codeValue: string // value as would appear in code
186 }
187
```

Codificação 3: Formalização da estrutura da AST para COMP-ITA 2019

Assim sendo, além da gramática em *Nearley*, foram necessárias codificações adicionais para (i) converter a árvore “verbosa” na AST, e (ii) percorrer a AST objetivando reimprimir o código-fonte reformatado.

A primeira, naturalmente, foi trabalhosa e exigiu 720 linhas de código *typescript* de complexidade considerável (trecho ilustrado pela Codificação 4). Todavia ela possibilitou tornar a segunda mais fácil, exigindo somente 217 linhas de código de fácil entendimento (trecho ilustrado pela Codificação 5).

```
function ParseAuxExpr2(auxExpr2Node: SyntaxTree): Expression {
  const children = auxExpr2Node.nodeChildren

  if (isSyntaxTree(children[0])) {
    return ParseAuxExpr3(cast('SyntaxTree', 'AuxExpr3', children[0]))
  }

  return {
    kind: 'not',
    target: ParseAuxExpr3(cast('SyntaxTree', 'AuxExpr3', children[1]))
  } as LogicalNOT
}

function ParseAuxExpr3(auxExpr3Node: SyntaxTree): Expression {
  const children = auxExpr3Node.nodeChildren

  if (children.length === 1) {
    return ParseAuxExpr4(cast('SyntaxTree', 'AuxExpr4', children[0]))
  }

  const leftExpr = cast('SyntaxTree', 'AuxExpr4', children[0])
  const operator = cast('Token', 'RELOP', children[1]).text
  const rightExpr = cast('SyntaxTree', 'AuxExpr4', children[2])

  return {
    kind: 'comparison',
    operator,
    leftSide: ParseAuxExpr4(leftExpr),
    rightSide: ParseAuxExpr4(rightExpr)
  } as Comparison
}
```

Codificação 4: Trecho do código não-trivial de conversão "árvore verbosa" → "AST"

```
1    visitProgram(node: Program) {
2      this.print(`program ${node.name} {`)
3      this.print('')
4
5      if (node.declarations.length) {
6        this.print('global:')
7        this.print('')
8        this.printDeclarations(node.declarations, 0)
9        this.print('')
10       this.print('')
11     }
12
13     this.print('functions:')
14     this.print('')
15     node.functions.forEach(func => this.printFunction(func))
16     this.print('')
17
18     // this.print('')
19     this.print('}')
20   }
```

Codificação 5: Trecho do código de fácil entendimento que imprime a AST

4 Testes realizados

Usaram-se alguns programas, entre eles alguns com erros sintáticos, para avaliar o funcionamento do analisador sintático (e a correção de seu *pretty-printer*). A seguir, listam-se os nomes desses programas e a intenção de cada um. Os arquivos-fonte e os resultados produzidos podem ser lidos nos apêndices respectivos.

- analise-de-texto (apêndices A e B): programa-exemplo fornecido pelo professor. Não tem erros sintáticos;
- *analise-de-texto-sem-linhas* (apêndices C e D): similar ao anterior, mas reformatado para ter somente 1 linha (todos os *\n* foram retirados). Não tem erros sintáticos;
- *no-glob-decls* (apêndices E e F): um programa sem declarações globais. Não tem erros sintáticos;
- *no-loc-decls* (apêndices G e H): um programa sem declarações locais. Não tem erros sintáticos;
- *no-stats* (apêndices I e J): um programa sem "*statements*". Não tem erros sintáticos;
- *erro1* e *erro2* (apêndices K, L, M e N)*:* dois programas com erro sintático. Observe que o analisador informa o primeiro erro encontrado e onde ele está (linha, coluna), mas não invoca o *pretty-printer* neste caso;
- *math* (apêndices O e P): um programa com algumas operações matemáticas, incluindo referência a um *float* (não usado nos programas anteriores). Não tem erros sintáticos;
- *redundant* (apêndices Q e R): um programa com chaves adicionais desnecessárias. Não tem erros sintáticos.

Como garantia adicional, cada programa foi analisado 2 vezes: primeiro, o programa original foi analisado e reformatado; em seguida, o código-fonte reformatado foi analisado e também reformatado. Conferiu-se que as duas versões reformatadas eram sempre idênticas.

Note que o *pretty-printer* faz 4 escolhas inusitadas (propositalmente):

1. ele avalia *floats* e, ao reimprimí-los, exibe o resultado já aproximado (como seria interpretado na execução do programa);
2. expande declarações, ou seja, declarações múltiplas declaradas na mesma linha aparecem em múltiplas linhas no código reformatado;
3. Remove chaves adicionais desnecessárias (por exemplo, abrir um *if* com 6 chaves em vez de 2). Além disso, insere pelo menos um par de chaves a certos comandos (por exemplo, *if* e *while*), mesmo que o código-fonte não tivesse o tal par originalmente (uma espécie de *linting*);
4. Os conteúdos das seções *global: , functions:* e *statements:* do programa não são tabuladas.

5 Experimentação do código

Para testar o código usado, basta abrir o arquivo **index.html** fornecido junto com este laboratório em um navegador e seguir as instruções da página.

6 Referências

[1] Mokarzel, F. C. **Linguagem COMP-ITA 2019**. Especificação formal de uma linguagem de programação utilizada na disciplina CES-41 do Instituto Tecnológico de Aeronáutica, 2019.

[2] Mokarzel, F. C. **3º Laboratório de CES-41/2019**. Roteiro do laboratório 3 da disciplina CES-41 do Instituto Tecnológico de Aeronáutica, 2019.

[3] Radvan, T. **Moo**. Analisador léxico escrito em *javascript*. Repositório em: <https://github.com/no-context/moo>. Acesso em 04/03/2019.

[4] Hardmath123. **Nearley**. Analisador sintático em *javascript*. Repositório em: <https://github.com/vihanb/nearley>. Acesso em 09/04/2019.

APÊNDICE A: CÓDIGO-FONTE ANALISE-DE-TEXTO

```
1  /*  Programa para contar as ocorrencias das palavras de um texto */
2
3  program AnaliseDeTexto {
4
5  /*  Variaveis globais  */
6
7  global:
8     char nomes[50,10], palavra[10];
9     int ntab, nocorr[50];
10       char c; logic fim;
11
12 functions:
13
14 /*  Funcao para procurar uma palavra na tabela de palavras  */
15
16 int Procura () {
17
18 local:
19    int i, inf, sup, med, posic, compara;
20       logic achou, fimteste;
21 statements:
22    achou <- false; inf <- 1; sup <- ntab;
23    while (!achou && sup >= inf) {
24       med <- (inf + sup) / 2;
25       compara <- 0; fimteste <- false;
26       for (i <- 0; !fimteste && compara = 0; i <- i+1) {
27             if (palavra[i] < nomes[med,i])
28             compara <- ~1;
29             else if (palavra[i] > nomes[med,i])
30             compara <- 1;
31             if (palavra[i] = '\0' || nomes[med,i] = '\0')
32             fimteste <- true;
33       }
34       if (compara = 0)
35          achou <- true;
36       else if (compara < 0)
37          sup <- med - 1;
38       else inf <- med + 1;
39    }
40    if (achou) posic <- med;
41    else posic <- ~inf;
42    return posic;
43
44 } /* Fim da funcao Procura */
45
46 /*  Funcao para inserir uma palavra na tabela de palavras  */
47
48 void Inserir (int posic) {
```

```
49
50  local:
51      int i, j; logic fim;
52  statements:
53      ntab <- ntab + 1;
54      for (i <- ntab; i >= posic+1; i <- i-1) {
55          fim <- false;
56          for (j <- 0; !fim; j <- j+1) {
57              nomes[i,j] <- nomes[i-1,j];
58              if (nomes[i,j] = '\0') fim <- true;
59          }
60         nocorr[i] <- nocorr[i-1];
61      }
62          fim <- false;
63          for (j <- 0; !fim; j <- j+1) {
64              nomes[posic,j] <- palavra[j];
65              if (palavra[j] = '\0') fim <- true;
66          }
67      nocorr[posic] <- 1;
68
69  } /* Fim da funcao Inserir */
70
71  /*  Funcao para escrever a tabela de palavras  */
72
73  void ExibirTabela () {
74
75  local:
76      int i; logic fim;
77  statements:
78      write ("          ", "Palavra              ",
79                              "   Num. de ocorr.");
80      for (i <- 1; i <= 50; i <- i+1) write ("-");
81      for (i <- 1; i <= ntab; i <- i+1) {
82          write ("\n          "); fim <- false;
83          for (j <- 0; !fim; j <- j+1) {
84              if (nomes[i,j] = '\0') fim <- true;
85              else write (nomes[i,j]);
86          }
87          write (" | ", nocorr[i]);
88      }
89
90  } /* Fim da funcao ExibirTabela */
91
92
93  /*  Modulo principal  */
94
95  main {
96
97  local:
98      int i, posic;
99          char c; logic fim;
100 statements:
```

```
101     ntab <- 0;
102     write ("Nova palavra? (s/n): ");
103     read (c);
104     while (c = 's' || c = 'S') {
105             write ("\nDigite a palavra: ");
106             fim <- false;
107         for (i <- 0; !fim; i <- i+1) {
108                 read (palavra[i]);
109                 if (palavra[i] = '\n') {
110                     fim <- true;
111                     palavra[i] <- '\0';
112                 }
113             }
114         posic <- Procura ();
115         if (posic > 0)
116             nocorr[posic] <- nocorr[posic] + 1;
117         else
118             call Inserir (~posic, i);
119             write ("\n\nNova palavra? (s/n): ");
120             read (c);
121     }
122     call ExibirTabela ();
123
124 } /* Fim da funcao main */
125
126 } /* Fim do programa AnaliseDeTexto */
```

APÊNDICE B: RESULTADO PARA ANALISE-DE-TEXTO

```
1  program AnaliseDeTexto {
2
3  global:
4
5  char nomes[50, 10];
6  char palavra[10];
7  int ntab;
8  int nocorr[50];
9  char c;
10 logic fim;
11
12
13 functions:
14
15 int Procura() {
16
17   local:
18
19   int i;
20   int inf;
21   int sup;
22   int med;
23   int posic;
24   int compara;
25   logic achou;
26   logic fimteste;
27
28
29   statements:
30
31   achou <- false;
32   inf <- 1;
33   sup <- ntab;
34   while(!achou && sup >= inf) {
35     med <- (inf + sup) / 2;
36     compara <- 0;
37     fimteste <- false;
38     for (i <- 0; !fimteste && compara = 0; i <- i + 1) {
39       if(palavra[i] < nomes[med, i]) {
40         compara <- ~1;
41       } else {
42         if(palavra[i] > nomes[med, i]) {
43           compara <- 1;
44         }
45       }
46       if(palavra[i] = '\0' || nomes[med, i] = '\0') {
47         fimteste <- true;
48       }
```

```
49      }
50      if(compara = 0) {
51        achou <- true;
52      } else {
53        if(compara < 0) {
54          sup <- med - 1;
55        } else {
56          inf <- med + 1;
57        }
58      }
59    }
60    if(achou) {
61      posic <- med;
62    } else {
63      posic <- ~inf;
64    }
65    return posic;
66
67  }
68
69  void Inserir(int posic) {
70
71    local:
72
73    int i;
74    int j;
75    logic fim;
76
77
78    statements:
79
80    ntab <- ntab + 1;
81    for (i <- ntab; i >= posic + 1; i <- i - 1) {
82      fim <- false;
83      for (j <- 0; !fim; j <- j + 1) {
84        nomes[i, j] <- nomes[i - 1, j];
85        if(nomes[i, j] = '\0') {
86          fim <- true;
87        }
88      }
89      nocorr[i] <- nocorr[i - 1];
90    }
91    fim <- false;
92    for (j <- 0; !fim; j <- j + 1) {
93      nomes[posic, j] <- palavra[j];
94      if(palavra[j] = '\0') {
95        fim <- true;
96      }
97    }
98    nocorr[posic] <- 1;
99
100 }
```

```
101
102 void ExibirTabela() {
103
104   local:
105
106   int i;
107   logic fim;
108
109
110   statements:
111
112   write("          ", "Palavra             ", "   Num. de ocorr.");
113   for (i <- 1; i <= 50; i <- i + 1) {
114     write("-");
115   }
116   for (i <- 1; i <= ntab; i <- i + 1) {
117     write("\n          ");
118     fim <- false;
119     for (j <- 0; !fim; j <- j + 1) {
120       if(nomes[i, j] = '\0') {
121         fim <- true;
122       } else {
123         write(nomes[i, j]);
124       }
125     }
126     write(" | ", nocorr[i]);
127   }
128
129 }
130
131 main {
132
133   local:
134
135   int i;
136   int posic;
137   char c;
138   logic fim;
139
140
141   statements:
142
143   ntab <- 0;
144   write("Nova palavra? (s/n): ");
145   read(c);
146   while(c = 's' || c = 'S') {
147     write("\nDigite a palavra: ");
148     fim <- false;
149     for (i <- 0; !fim; i <- i + 1) {
150       read(palavra[i]);
151       if(palavra[i] = '\n') {
152         fim <- true;
```

```
153         palavra[i] <- '\0';
154       }
155     }
156     posic <- Procura();
157     if(posic > 0) {
158       nocorr[posic] <- nocorr[posic] + 1;
159     } else {
160       call Inserir(~posic, i);
161     }
162     write("\n\nNova palavra? (s/n): ");
163     read(c);
164   }
165   call ExibirTabela();
166 
167 }
168 
169 
170 }
171 
```

APÊNDICE C: CÓDIGO-FONTE ANALISE-DE-TEXTO-SEM-LINHAS

```
1 /*  Programa para contar as ocorrencias das palavras de um texto */program
  AnaliseDeTextoSemLinhas {/*  Variaveis globais  */global:    char nomes[50,10], palavra[10];
  int ntab, nocorr[50];       char c; logic fim;functions:/*  Funcao para procurar uma palavra na
  tabela de palavras  */int Procura () {local:    int i, inf, sup, med, posic, compara;       logic achou,
  fimteste;statements:   achou <- false; inf <- 1; sup <- ntab;  while (!achou && sup >= inf) {
  med <- (inf + sup) / 2;     compara <- 0; fimteste <- false;        for (i <- 0; !fimteste && compara
  = 0; i <- i+1) {            if (palavra[i] < nomes[med,i])              compara <- ~1;            else if
  (palavra[i] > nomes[med,i])             compara <- 1;             if (palavra[i] = '\0' || nomes[med,i]
  = '\0')                 fimteste <- true;       }       if (compara = 0)          achou <- true;      else if
  (compara < 0)          sup <- med - 1;     else inf <- med + 1;    }   if (achou) posic <- med;    else
  posic <- ~inf; return posic;} /* Fim da funcao Procura *//*  Funcao para inserir uma palavra na
  tabela de palavras  */void Inserir (int posic) {local: int i, j; logic fim;statements: ntab <- ntab +
  1;   for (i <- ntab; i >= posic+1; i <- i-1) {        fim <- false;       for (j <- 0; !fim; j <- j+1)
  {           nomes[i,j] <- nomes[i-1,j];           if (nomes[i,j] = '\0') fim <- true;       }    nocorr[i] <-
  nocorr[i-1];    }       fim <- false;       for (j <- 0; !fim; j <- j+1) {          nomes[posic,j] <-
  palavra[j];          if (palavra[j] = '\0') fim <- true;     }   nocorr[posic] <- 1;} /* Fim da funcao
  Inserir *//*  Funcao para escrever a tabela de palavras  */void ExibirTabela () {local:   int i; logic
  fim;statements:    write ("          ", "Palavra          ",                    "   Num. de ocorr.");
  for (i <- 1; i <= 50; i <- i+1) write ("-");    for (i <- 1; i <= ntab; i <- i+1) {     write ("\n          ");
  fim <- false;       for (j <- 0; !fim; j <- j+1) {          if (nomes[i,j] = '\0') fim <- true;          else
  write (nomes[i,j]);          }       write (" | ", nocorr[i]);   }} /* Fim da funcao ExibirTabela *//*
  Modulo principal  */main {local: int i, posic;       char c; logic fim;statements:   ntab <- 0;  write
  ("Nova palavra? (s/n): ");    read (c);   while (c = 's' || c = 'S') {          write ("\nDigite a palavra:
  ");        fim <- false;       for (i <- 0; !fim; i <- i+1) {              read (palavra[i]);              if
  (palavra[i] = '\n') {                   fim <- true;                    palavra[i] <- '\0';           }         }
  posic <- Procura ();        if (posic > 0)          nocorr[posic] <- nocorr[posic] + 1;     else          call
  Inserir (~posic, i);          write ("\n\nNova palavra? (s/n): ");          read (c);   }   call
  ExibirTabela ();} /* Fim da funcao main */} /* Fim do programa AnaliseDeTexto */
```

APÊNDICE D: RESULTADO PARA ANALISE-DE-TEXTO-SEM-LINHAS

```
1  program AnaliseDeTextoSemLinhas {
2
3  global:
4
5  char nomes[50, 10];
6  char palavra[10];
7  int ntab;
8  int nocorr[50];
9  char c;
10 logic fim;
11
12
13 functions:
14
15 int Procura() {
16
17   local:
18
19   int i;
20   int inf;
21   int sup;
22   int med;
23   int posic;
24   int compara;
25   logic achou;
26   logic fimteste;
27
28
29   statements:
30
31   achou <- false;
32   inf <- 1;
33   sup <- ntab;
34   while(!achou && sup >= inf) {
35     med <- (inf + sup) / 2;
36     compara <- 0;
37     fimteste <- false;
38     for (i <- 0; !fimteste && compara = 0; i <- i + 1) {
39       if(palavra[i] < nomes[med, i]) {
40         compara <- ~1;
41       } else {
42         if(palavra[i] > nomes[med, i]) {
43           compara <- 1;
44         }
45       }
46       if(palavra[i] = '\0' || nomes[med, i] = '\0') {
```

```
47          fimteste <- true;
48        }
49      }
50      if(compara = 0) {
51        achou <- true;
52      } else {
53        if(compara < 0) {
54          sup <- med - 1;
55        } else {
56          inf <- med + 1;
57        }
58      }
59    }
60    if(achou) {
61      posic <- med;
62    } else {
63      posic <- ~inf;
64    }
65    return posic;
66
67  }
68
69  void Inserir(int posic) {
70
71    local:
72
73    int i;
74    int j;
75    logic fim;
76
77
78    statements:
79
80    ntab <- ntab + 1;
81    for (i <- ntab; i >= posic + 1; i <- i - 1) {
82      fim <- false;
83      for (j <- 0; !fim; j <- j + 1) {
84        nomes[i, j] <- nomes[i - 1, j];
85        if(nomes[i, j] = '\0') {
86          fim <- true;
87        }
88      }
89      nocorr[i] <- nocorr[i - 1];
90    }
91    fim <- false;
92    for (j <- 0; !fim; j <- j + 1) {
93      nomes[posic, j] <- palavra[j];
94      if(palavra[j] = '\0') {
95        fim <- true;
96      }
97    }
98    nocorr[posic] <- 1;
```

```
 99
100 }
101
102 void ExibirTabela() {
103
104   local:
105
106   int i;
107   logic fim;
108
109
110   statements:
111
112   write("          ", "Palavra             ", "   Num. de ocorr.");
113   for (i <- 1; i <= 50; i <- i + 1) {
114     write("-");
115   }
116   for (i <- 1; i <= ntab; i <- i + 1) {
117     write("\n          ");
118     fim <- false;
119     for (j <- 0; !fim; j <- j + 1) {
120       if(nomes[i, j] = '\0') {
121         fim <- true;
122       } else {
123         write(nomes[i, j]);
124       }
125     }
126     write(" | ", nocorr[i]);
127   }
128
129 }
130
131 main {
132
133   local:
134
135   int i;
136   int posic;
137   char c;
138   logic fim;
139
140
141   statements:
142
143   ntab <- 0;
144   write("Nova palavra? (s/n): ");
145   read(c);
146   while(c = 's' || c = 'S') {
147     write("\nDigite a palavra: ");
148     fim <- false;
149     for (i <- 0; !fim; i <- i + 1) {
150       read(palavra[i]);
```

```
151       if(palavra[i] = '\n') {
152         fim <- true;
153         palavra[i] <- '\0';
154       }
155     }
156     posic <- Procura();
157     if(posic > 0) {
158       nocorr[posic] <- nocorr[posic] + 1;
159     } else {
160       call Inserir(~posic, i);
161     }
162     write("\n\nNova palavra? (s/n): ");
163     read(c);
164   }
165   call ExibirTabela();
166 
167 }
168 
169 
170 }
171 
```

APÊNDICE E: CÓDIGO-FONTE NO-GLOB-DECLS

```
1  /*  Programa para contar as ocorrencias das palavras de um texto */
2
3  program NoGlobDecls {
4
5  /*  Variaveis globais  */
6
7  /*
8  global:
9     char nomes[50,10], palavra[10];
10    int ntab, nocorr[50];
11       char c; logic fim;
12 */
13
14 functions:
15
16 /*  Funcao para procurar uma palavra na tabela de palavras  */
17
18 int Procura () {
19
20 local:
21    int i, inf, sup, med, posic, compara;
22       logic achou, fimteste;
23 statements:
24    achou <- false; inf <- 1; sup <- ntab;
25    while (!achou && sup >= inf) {
26       med <- (inf + sup) / 2;
27       compara <- 0; fimteste <- false;
28       for (i <- 0; !fimteste && compara = 0; i <- i+1) {
29             if (palavra[i] < nomes[med,i])
30             compara <- ~1;
31             else if (palavra[i] > nomes[med,i])
32             compara <- 1;
33             if (palavra[i] = '\0' || nomes[med,i] = '\0')
34             fimteste <- true;
35       }
36       if (compara = 0)
37          achou <- true;
38       else if (compara < 0)
39          sup <- med - 1;
40       else inf <- med + 1;
41    }
42    if (achou) posic <- med;
43    else posic <- ~inf;
44    return posic;
45
46 } /* Fim da funcao Procura */
47
48 /*  Funcao para inserir uma palavra na tabela de palavras  */
```

```
49
50  void Inserir (int posic) {
51
52  local:
53     int i, j; logic fim;
54  statements:
55     ntab <- ntab + 1;
56     for (i <- ntab; i >= posic+1; i <- i-1) {
57        fim <- false;
58        for (j <- 0; !fim; j <- j+1) {
59           nomes[i,j] <- nomes[i-1,j];
60           if (nomes[i,j] = '\0') fim <- true;
61        }
62       nocorr[i] <- nocorr[i-1];
63     }
64        fim <- false;
65        for (j <- 0; !fim; j <- j+1) {
66           nomes[posic,j] <- palavra[j];
67           if (palavra[j] = '\0') fim <- true;
68        }
69     nocorr[posic] <- 1;
70
71  } /* Fim da funcao Inserir */
72
73  /*  Funcao para escrever a tabela de palavras  */
74
75  void ExibirTabela () {
76
77  local:
78     int i; logic fim;
79  statements:
80     write ("          ", "Palavra              ",
81                           "   Num. de ocorr.");
82     for (i <- 1; i <= 50; i <- i+1) write ("-");
83     for (i <- 1; i <= ntab; i <- i+1) {
84        write ("\n          "); fim <- false;
85        for (j <- 0; !fim; j <- j+1) {
86           if (nomes[i,j] = '\0') fim <- true;
87           else write (nomes[i,j]);
88        }
89        write (" | ", nocorr[i]);
90     }
91
92  } /* Fim da funcao ExibirTabela */
93
94
95  /*  Modulo principal  */
96
97  main {
98
99  local:
100    int i, posic;
```

```
101         char c; logic fim;
102 statements:
103     ntab <- 0;
104     write ("Nova palavra? (s/n): ");
105     read (c);
106     while (c = 's' || c = 'S') {
107             write ("\nDigite a palavra: ");
108             fim <- false;
109         for (i <- 0; !fim; i <- i+1) {
110                 read (palavra[i]);
111                 if (palavra[i] = '\n') {
112                     fim <- true;
113                     palavra[i] <- '\0';
114                 }
115             }
116         posic <- Procura ();
117         if (posic > 0)
118             nocorr[posic] <- nocorr[posic] + 1;
119         else
120             call Inserir (~posic, i);
121             write ("\n\nNova palavra? (s/n): ");
122             read (c);
123     }
124     call ExibirTabela ();
125
126 } /* Fim da funcao main */
127
128 } /* Fim do programa AnaliseDeTexto */
```

APÊNDICE F: RESULTADO PARA NO-GLOB-DECLS

```
1   program NoGlobDecls {
2
3   functions:
4
5   int Procura() {
6
7     local:
8
9     int i;
10    int inf;
11    int sup;
12    int med;
13    int posic;
14    int compara;
15    logic achou;
16    logic fimteste;
17
18
19    statements:
20
21    achou <- false;
22    inf <- 1;
23    sup <- ntab;
24    while(!achou && sup >= inf) {
25      med <- (inf + sup) / 2;
26      compara <- 0;
27      fimteste <- false;
28      for (i <- 0; !fimteste && compara = 0; i <- i + 1) {
29        if(palavra[i] < nomes[med, i]) {
30          compara <- ~1;
31        } else {
32          if(palavra[i] > nomes[med, i]) {
33            compara <- 1;
34          }
35        }
36        if(palavra[i] = '\0' || nomes[med, i] = '\0') {
37          fimteste <- true;
38        }
39      }
40      if(compara = 0) {
41        achou <- true;
42      } else {
43        if(compara < 0) {
44          sup <- med - 1;
45        } else {
46          inf <- med + 1;
47        }
48      }
```

```
49    }
50    if(achou) {
51      posic <- med;
52    } else {
53      posic <- ~inf;
54    }
55    return posic;
56
57  }
58
59  void Inserir(int posic) {
60
61    local:
62
63    int i;
64    int j;
65    logic fim;
66
67
68    statements:
69
70    ntab <- ntab + 1;
71    for (i <- ntab; i >= posic + 1; i <- i - 1) {
72      fim <- false;
73      for (j <- 0; !fim; j <- j + 1) {
74        nomes[i, j] <- nomes[i - 1, j];
75        if(nomes[i, j] = '\0') {
76          fim <- true;
77        }
78      }
79      nocorr[i] <- nocorr[i - 1];
80    }
81    fim <- false;
82    for (j <- 0; !fim; j <- j + 1) {
83      nomes[posic, j] <- palavra[j];
84      if(palavra[j] = '\0') {
85        fim <- true;
86      }
87    }
88    nocorr[posic] <- 1;
89
90  }
91
92  void ExibirTabela() {
93
94    local:
95
96    int i;
97    logic fim;
98
99
100   statements:
```

```
101
102   write("          ", "Palavra             ", "   Num. de ocorr.");
103   for (i <- 1; i <= 50; i <- i + 1) {
104     write("-");
105   }
106   for (i <- 1; i <= ntab; i <- i + 1) {
107     write("\n          ");
108     fim <- false;
109     for (j <- 0; !fim; j <- j + 1) {
110       if(nomes[i, j] = '\0') {
111         fim <- true;
112       } else {
113         write(nomes[i, j]);
114       }
115     }
116     write(" | ", nocorr[i]);
117   }
118
119 }
120
121 main {
122
123   local:
124
125   int i;
126   int posic;
127   char c;
128   logic fim;
129
130
131   statements:
132
133   ntab <- 0;
134   write("Nova palavra? (s/n): ");
135   read(c);
136   while(c = 's' || c = 'S') {
137     write("\nDigite a palavra: ");
138     fim <- false;
139     for (i <- 0; !fim; i <- i + 1) {
140       read(palavra[i]);
141       if(palavra[i] = '\n') {
142         fim <- true;
143         palavra[i] <- '\0';
144       }
145     }
146     posic <- Procura();
147     if(posic > 0) {
148       nocorr[posic] <- nocorr[posic] + 1;
149     } else {
150       call Inserir(~posic, i);
151     }
152     write("\n\nNova palavra? (s/n): ");
```

```
153     read(c);
154   }
155   call ExibirTabela();
156
157 }
158
159
160 }
161
```

APÊNDICE G: CÓDIGO-FONTE NO-LOC-DECLS

```
1  /*  Programa para contar as ocorrencias das palavras de um texto */
2
3  program NoLocDecls {
4
5  /*  Variaveis globais  */
6
7  global:
8     char nomes[50,10], palavra[10];
9     int ntab, nocorr[50];
10       char c; logic fim;
11
12 functions:
13
14 /*  Funcao para procurar uma palavra na tabela de palavras  */
15
16 int Procura () {
17
18 statements:
19    achou <- false; inf <- 1; sup <- ntab;
20    while (!achou && sup >= inf) {
21       med <- (inf + sup) / 2;
22       compara <- 0; fimteste <- false;
23       for (i <- 0; !fimteste && compara = 0; i <- i+1) {
24             if (palavra[i] < nomes[med,i])
25             compara <- ~1;
26             else if (palavra[i] > nomes[med,i])
27             compara <- 1;
28             if (palavra[i] = '\0' || nomes[med,i] = '\0')
29             fimteste <- true;
30       }
31       if (compara = 0)
32          achou <- true;
33       else if (compara < 0)
34          sup <- med - 1;
35       else inf <- med + 1;
36    }
37    if (achou) posic <- med;
38    else posic <- ~inf;
39    return posic;
40
41 } /* Fim da funcao Procura */
42
43 /*  Funcao para inserir uma palavra na tabela de palavras  */
44
45 void Inserir (int posic) {
46
47 statements:
48    ntab <- ntab + 1;
```

```
49      for (i <- ntab; i >= posic+1; i <- i-1) {
50          fim <- false;
51          for (j <- 0; !fim; j <- j+1) {
52              nomes[i,j] <- nomes[i-1,j];
53              if (nomes[i,j] = '\0') fim <- true;
54          }
55         nocorr[i] <- nocorr[i-1];
56      }
57          fim <- false;
58          for (j <- 0; !fim; j <- j+1) {
59              nomes[posic,j] <- palavra[j];
60              if (palavra[j] = '\0') fim <- true;
61          }
62      nocorr[posic] <- 1;
63
64  } /* Fim da funcao Inserir */
65
66  /*  Funcao para escrever a tabela de palavras  */
67
68  void ExibirTabela () {
69
70  statements:
71      write ("          ", "Palavra             ",
72                              "   Num. de ocorr.");
73      for (i <- 1; i <= 50; i <- i+1) write ("-");
74      for (i <- 1; i <= ntab; i <- i+1) {
75          write ("\n          "); fim <- false;
76          for (j <- 0; !fim; j <- j+1) {
77              if (nomes[i,j] = '\0') fim <- true;
78              else write (nomes[i,j]);
79          }
80          write (" | ", nocorr[i]);
81      }
82
83  } /* Fim da funcao ExibirTabela */
84
85
86  /*  Modulo principal  */
87
88  main {
89
90  statements:
91      ntab <- 0;
92      write ("Nova palavra? (s/n): ");
93      read (c);
94      while (c = 's' || c = 'S') {
95              write ("\nDigite a palavra: ");
96              fim <- false;
97          for (i <- 0; !fim; i <- i+1) {
98                  read (palavra[i]);
99                  if (palavra[i] = '\n') {
100                     fim <- true;
```

```
101                 palavra[i] <- '\0';
102             }
103         }
104     posic <- Procura ();
105     if (posic > 0)
106         nocorr[posic] <- nocorr[posic] + 1;
107     else
108         call Inserir (~posic, i);
109         write ("\n\nNova palavra? (s/n): ");
110         read (c);
111   }
112   call ExibirTabela ();
113
114 } /* Fim da funcao main */
115
116 } /* Fim do programa AnaliseDeTexto */
```

APÊNDICE H: RESULTADO PARA NO-LOC-DECLS

```
1  program NoLocDecls {
2
3  global:
4
5  char nomes[50, 10];
6  char palavra[10];
7  int ntab;
8  int nocorr[50];
9  char c;
10 logic fim;
11
12
13 functions:
14
15 int Procura() {
16
17   statements:
18
19   achou <- false;
20   inf <- 1;
21   sup <- ntab;
22   while(!achou && sup >= inf) {
23     med <- (inf + sup) / 2;
24     compara <- 0;
25     fimteste <- false;
26     for (i <- 0; !fimteste && compara = 0; i <- i + 1) {
27       if(palavra[i] < nomes[med, i]) {
28         compara <- ~1;
29       } else {
30         if(palavra[i] > nomes[med, i]) {
31           compara <- 1;
32         }
33       }
34       if(palavra[i] = '\0' || nomes[med, i] = '\0') {
35         fimteste <- true;
36       }
37     }
38     if(compara = 0) {
39       achou <- true;
40     } else {
41       if(compara < 0) {
42         sup <- med - 1;
43       } else {
44         inf <- med + 1;
45       }
46     }
47   }
48   if(achou) {
```

```
49      posic <- med;
50    } else {
51      posic <- ~inf;
52    }
53    return posic;
54
55  }
56
57  void Inserir(int posic) {
58
59    statements:
60
61    ntab <- ntab + 1;
62    for (i <- ntab; i >= posic + 1; i <- i - 1) {
63      fim <- false;
64      for (j <- 0; !fim; j <- j + 1) {
65        nomes[i, j] <- nomes[i - 1, j];
66        if(nomes[i, j] = '\0') {
67          fim <- true;
68        }
69      }
70      nocorr[i] <- nocorr[i - 1];
71    }
72    fim <- false;
73    for (j <- 0; !fim; j <- j + 1) {
74      nomes[posic, j] <- palavra[j];
75      if(palavra[j] = '\0') {
76        fim <- true;
77      }
78    }
79    nocorr[posic] <- 1;
80
81  }
82
83  void ExibirTabela() {
84
85    statements:
86
87    write("          ", "Palavra             ", "   Num. de ocorr.");
88    for (i <- 1; i <= 50; i <- i + 1) {
89      write("-");
90    }
91    for (i <- 1; i <= ntab; i <- i + 1) {
92      write("\n          ");
93      fim <- false;
94      for (j <- 0; !fim; j <- j + 1) {
95        if(nomes[i, j] = '\0') {
96          fim <- true;
97        } else {
98          write(nomes[i, j]);
99        }
100     }
```

```
101     write(" | ", nocorr[i]);
102   }
103
104 }
105
106 main {
107
108   statements:
109
110   ntab <- 0;
111   write("Nova palavra? (s/n): ");
112   read(c);
113   while(c = 's' || c = 'S') {
114     write("\nDigite a palavra: ");
115     fim <- false;
116     for (i <- 0; !fim; i <- i + 1) {
117       read(palavra[i]);
118       if(palavra[i] = '\n') {
119         fim <- true;
120         palavra[i] <- '\0';
121       }
122     }
123     posic <- Procura();
124     if(posic > 0) {
125       nocorr[posic] <- nocorr[posic] + 1;
126     } else {
127       call Inserir(~posic, i);
128     }
129     write("\n\nNova palavra? (s/n): ");
130     read(c);
131   }
132   call ExibirTabela();
133
134 }
135
136
137 }
138
```

APÊNDICE I: CÓDIGO-FONTE NO-STATS

```
1  /*  Programa para contar as ocorrencias das palavras de um texto */
2
3  program NoStats {
4
5  /*  Variaveis globais  */
6
7  global:
8     char nomes[50,10], palavra[10];
9     int ntab, nocorr[50];
10        char c; logic fim;
11
12 functions:
13
14 /*  Funcao para procurar uma palavra na tabela de palavras  */
15
16 int Procura () {
17
18 local:
19    int i, inf, sup, med, posic, compara;
20       logic achou, fimteste;
21 statements:
22
23 } /* Fim da funcao Procura */
24
25 /*  Funcao para inserir uma palavra na tabela de palavras  */
26
27 void Inserir (int posic) {
28
29 local:
30    int i, j; logic fim;
31 statements:
32
33 } /* Fim da funcao Inserir */
34
35 /*  Funcao para escrever a tabela de palavras  */
36
37 void ExibirTabela () {
38
39 local:
40    int i; logic fim;
41 statements:
42
43 } /* Fim da funcao ExibirTabela */
44
45
46 /*  Modulo principal  */
47
48 main {
```

```
49
50 local:
51     int i, posic;
52         char c; logic fim;
53 statements:
54
55 } /* Fim da funcao main */
56
57 } /* Fim do programa AnaliseDeTexto */
```

APÊNDICE J: RESULTADO PARA NO-STATS

```
1  program NoStats {
2
3  global:
4
5  char nomes[50, 10];
6  char palavra[10];
7  int ntab;
8  int nocorr[50];
9  char c;
10 logic fim;
11
12
13 functions:
14
15 int Procura() {
16
17   local:
18
19   int i;
20   int inf;
21   int sup;
22   int med;
23   int posic;
24   int compara;
25   logic achou;
26   logic fimteste;
27
28
29   statements:
30
31
32 }
33
34 void Inserir(int posic) {
35
36   local:
37
38   int i;
39   int j;
40   logic fim;
41
42
43   statements:
44
45
46 }
47
48 void ExibirTabela() {
```

```
49
50    local:
51
52    int i;
53    logic fim;
54
55
56    statements:
57
58
59  }
60
61  main {
62
63    local:
64
65    int i;
66    int posic;
67    char c;
68    logic fim;
69
70
71    statements:
72
73
74  }
75
76
77  }
78
```

APÊNDICE K: CÓDIGO-FONTE PARA ERRO1

```
1  program Erro1 {
2
3  global:
4    int globalVar;
5
6  functions:
7
8  void Search(int vector) {
9    local:
10     char localVar;
11
12   statements:
13     localVar <- 1
14 }
15
16 }
```

APÊNDICE L: RESULTADO PARA ERRO1

```
1  Error: invalid syntax at line 14 col 1:
2
3    }
4    ^
5  Unexpected CLBRACE token: ""
```

APÊNDICE M: CÓDIGO-FONTE PARA ERRO2

```
1 program Erro2 {
2
3
4
5 }
```

APÊNDICE N: RESULTADO PARA ERRO2

```
1  Error: invalid syntax at line 5 col 1:
2
3    }
4    ^
5  Unexpected CLBRACE token: ""
```

APÊNDICE O: CÓDIGO-FONTE PARA MATH

```
1  program Math {
2
3  global:
4    int globalVar;
5
6  functions:
7
8  void Search(int vector) {
9    local:
10     char localVar;
11
12   statements:
13     localVar <- 'a';
14     globalVar <- 1 <= 2;
15     localVar <- 1 + (2 * localVar) / 3 % 5 + ~1;
16     globalVar <- 1.12345678987654321234567890;
17     globalVar <- 1.12345678987654321;
18 }
19
20 }
```

APÊNDICE P: RESULTADO PARA MATH

```
1  program Math {
2
3  global:
4
5  int globalVar;
6
7
8  functions:
9
10 void Search(int vector) {
11
12   local:
13
14   char localVar;
15
16
17   statements:
18
19   localVar <- 'a';
20   globalVar <- 1 <= 2;
21   localVar <- 1 + (2 * localVar) / 3 % 5 + ~1;
22   globalVar <- 1.1234567898765433;
23   globalVar <- 1.1234567898765433;
24
25 }
26
27
28 }
29
```

APÊNDICE Q: CÓDIGO FONTE REDUNDANT

```
1  program Redundant {
2     functions:
3
4     void SomeFunction() {
5        statements:
6        {{{{{
7           ih <- essasChavesSaoInuteis;
8           while(true) {{{{
9              maisUmaVez <- ChavesInuteis;
10             }}}}
11          if(true) {
12             call a();
13          }
14          else {{{
15             call b();
16             }}}
17          }}}}}
18    }
19 }
```

APÊNDICE R: RESULTADO PARA REDUNDANT

```
1  program Redundant {
2
3  functions:
4
5  void SomeFunction() {
6
7    statements:
8
9    ih <- essasChavesSaoInuteis;
10   while(true) {
11     maisUmaVez <- ChavesInuteis;
12   }
13   if(true) {
14     call a();
15   } else {
16     call b();
17   }
18
19 }
20
21
22 }
23
```