Universidade Estadual de Santa Cruz

Departamento de Ciências Exatas e Tecnológicas

Curso de Ciência da Computação

Disciplina: Compiladores – 2017.2

Professor: Paulo Costa

1º trabalho:

Analisador Léxico para Portugol

Autores:

José Augusto Santos Rodrigues

Patrick Silva Ferraz

Ilhéus, BA

14/11/2017

Sumário

1. Código fonte

```
1  /*
2   *---------------------------------------------------------------------
3   *
4   *   File    : Portugol.c
5   *   Created : 2017-10-28
6   *   Modified: 2017-10-28
7   *
8   *   Analisador Léxico da linguagem Portugol
9   *
10  *---------------------------------------------------------------------
11  */
12
13  /**
14   * -------------------------------------------------
15   * INCLUDES
16   * -------------------------------------------------
17   */
18
19  #include <stdio.h>
20  #include <stdlib.h>
21  #include <string.h>
22  #include "header/lexema.h"
23  #include "header/tokens.h"
24  #include "header/tabSimbolos.h"
25  #include "header/errosLexicos.h"
26  #include "header/tabTransicoes.h"
27  #include "header/geradorSaidas.h"
28
29  /**
30   * -------------------------------------------------
31   * TIPOS
32   * -------------------------------------------------
33   */
34
35  typedef struct
36  {
37     unsigned int qnt_col_linha_anterior;
38     tPos pos;
39  } tCursor;
40
41  typedef struct
42  {
43      int  existe_barra_n; // Armazena 1 se encontrar '\n'
44      int  qnt_char_apos_barra_n;
45      tPos pos_barra_n;
46  } tControleBloco;
47
48  /**
49   * -------------------------------------------------
50   * VARIÁVEIS GLOBAIS
51   * -------------------------------------------------
52   */
53
54  FILE                    *f_in;
55  tPos                    pos_inicial_simbolo;
56  tCursor                 cursor;
57  tLexema                 lexema;
58  tTabSimbolo             tab_simbolo;
59  tTabTkReconhecido       tab_token_reconhecido;
60  tTabErroLexIdentificado tab_erro_identificado;
61  char                    *ref_inicio_lexema;
62
63  /**
64   * -------------------------------------------------
65   * PROTÓTIPOS DE FUNÇÕES
66   * -------------------------------------------------
67   */
68
69  tToken analisadorLexico   (void);
70  void   leiaProximoSimbolo (char *prox_simbolo);
71  void   devolva            (int n);
72
```

```
/**
 * ------------------------------------------------
 * FUNÇÃO PRINCIPAL MAIN
 * ------------------------------------------------
 * Função main p/ simular o analisador sintático
 * solicitando 1 token por vez ao analisador léxico.
 * @param int argc, char **arqv
 * @return int
 */

int main (int argc, char **argv)
{
    // Verifica a quantidade de argumentos
    if (argc != 2)
        printf
        ("uso:\n"
            "\texecucao: ./Portugol nomearquivo.ptg\n"
        "onde:\n"
            "\tnomearquivo.ptg\teh um arquivo contendo o algoritmo na linguagem Portugol\n"
        );
    else
    {
        // Abre fluxo com arquivo de entrada
        if ((f_in = fopen(argv[1], "r")) == NULL)
        {
            printf("O arquivo escolhido nao existe\n");
            exit(1);
        }

        // Inicializa lexema e guarda o endereço inicial do lexema.nome alocado
        if (inicializeLexema(&lexema) == NULL)
        {
            printf("Espaco de memoria insuficiente p/ inicializar o lexema\n");
            exit(1);
        }

        if (inicializeTabErroLex(&tab_erro_identificado) == -1)
        {
            printf("Espaco de memoria insuficiente p/ inicializar tabela de erro\n");
            exit(1);
        }

        if (inicializeTabTkReconhecido(&tab_token_reconhecido) == -1)
        {
            printf("Espaco de memoria insuficiente p/ inicializar tabela de token\n");
            exit(1);
        }

        cursor.qnt_col_linha_anterior = 0;
        cursor.pos.lin = 1;
        cursor.pos.col = 0;

        // Chamada ao léxico solicitando tokens
        while (feof(f_in) == 0)
            analisadorLexico();


        if (gereArquivoErro(f_in, argv[1], &tab_erro_identificado) == -1)
        {
            printf("Erro ao gerar arquivo de erros\n");
            exit(1);
        }

        if (gereArquivoToken(argv[1], &tab_token_reconhecido, tab_simbolo, &lexema.maior_lex,
            (tab_erro_identificado.indice_ult_registro + 1)) == -1)
        {
            printf("Erro ao gerar arquivo de tokens reconhecidos\n");
            exit(1);
        }

        if (gereArquivoTabSimbolo(argv[1], tab_simbolo, &lexema.maior_lex) == -1)
        {
            printf("Erro ao gerar arquivo da tabela de simbolos\n");
            exit(1);
        }
```

```
148
149         finalizeLexema(&lexema);
150         finalizeTabSimbolo(tab_simbolo);
151         finalizeTabErroLex(&tab_erro_identificado);
152         finalizeTabTKReconhecido(&tab_token_reconhecido);
153         fclose(f_in);
154     }
155
156     return 0;
157
158 } // end-main (int argc, char **argv)
159
160 /**
161  * Função para ler o próximo símbolo do arquivo armazenando
162  * na variável prox_simbolo passada como referência.
163  * @param char *prox_simbolo
164  * @return void
165  */
166
167 void leiaProximoSimbolo (char *prox_simbolo)
168 {
169     if ((*prox_simbolo = fgetc(f_in)) == '\n')
170     {
171         cursor.qnt_col_linha_anterior = cursor.pos.col;
172         cursor.pos.col = 0;
173         cursor.pos.lin ++;
174     }
175     else
176         cursor.pos.col ++;
177
178 } // end-leiaProximoSimbolo (char *prox_simbolo)
179
180 /**
181  * Função para devolver n símbolos p/ entrada
182  * @param int n
183  * @return void
184  */
185
186 void devolva (int n)
187 {
188     fseek(f_in, n, SEEK_CUR);
189
190     if ((cursor.pos.col += n) < 0)
191     {
192         cursor.pos.lin --;
193         cursor.pos.col = cursor.qnt_col_linha_anterior - cursor.pos.col;
194     }
195
196 } // end-devolva (int n)
197
198 /**
199  * Função do analisador léxico. Retorna um token por vez ao
200  * analisador sintático, simulado na função main, identificado
201  * a partir do arquivo de entrada. Cada token é instalado na
202  * tabela de token, erros instalados na tabela de erros, e
203  * identificadores (tk_IDEN, tk_DECIMAL, tk_CADEIA, tk_INTEIRO)
204  * armazenado na tabela de símbolos. O analisador segue a ideia
205  * de uma máquina de estados, visível no switch e no header
206  * (tabTransicoes.h).
207  * @param void
208  * @return tToken
209  */
210
211 tToken analisadorLexico ()
212 {
213     char                simbolo;
214     int                 estado = 0;
215     tToken              token = -1;
216     tIdentificador      identificador;
217     tControleBloco      controle_bloco;
218     tTkReconhecido      token_reconhecido;
219     tErroLexIdentificado erro;
220
221     token_reconhecido.pos_tab_simbolo = -1;
222
```

```
223     controle_bloco.existe_barra_n = 0;
224     controle_bloco.qnt_char_apos_barra_n = 0;
225     controle_bloco.pos_barra_n.lin = 0;
226     controle_bloco.pos_barra_n.col = 0;
227
228     while (1)
229     {
230         switch (estado)
231         {
232             case 0:
233                 pos_inicial_simbolo.lin = cursor.pos.lin;
234                 pos_inicial_simbolo.col = cursor.pos.col + 1;
235                 break;
236             //---------------------------------------------------------------------
237             case 1:
238             case 4:
239             case 7:
240             case 9:
241             case 11:
242                 concateneSimboloLexema(&lexema, &simbolo);
243                 break;
244             //---------------------------------------------------------------------
245             case 3:
246                 erro.simbolo = simbolo;
247                 erro.lin = cursor.pos.lin;
248                 erro.col = cursor.pos.col;
249                 erro.id  = tErro_delimitador_esperado;
250                 instaleErroLex(&tab_erro_identificado, &erro);
251             case 2:
252                 devolva(-1);
253                 reinicieLexema(&lexema);
254                 token = tk_INTEIRO;
255                 if (gereIdentificador(&identificador, lexema.nome, &token, &pos_inicial_simbolo)
256 == -1)
257                 {
258                     printf("Espaco de memoria insuficiente p/ gerar identificador\n");
259                     exit(1);
260                 }
261                 if ((token_reconhecido.pos_tab_simbolo = instaleIdenTabSimbolo(tab_simbolo,
262 &identificador)) == -1)
263                 {
264                     printf("Espaco de memoria insuficiente p/ instalar identificador na tabela
265 de simbolos\n");
266                     exit(1);
267                 }
268                 break;
269             //---------------------------------------------------------------------
270             case 5:
271                 erro.simbolo = simbolo;
272                 erro.lin = cursor.pos.lin;
273                 erro.col = cursor.pos.col;
274                 erro.id  = tErro_delimitador_esperado;
275                 instaleErroLex(&tab_erro_identificado, &erro);
276             case 6:
277                 devolva(-1);
278                 reinicieLexema(&lexema);
279                 token = tk_DECIMAL;
280                 if (gereIdentificador(&identificador, lexema.nome, &token, &pos_inicial_simbolo)
281 == -1)
282                 {
283                     printf("Espaco de memoria insuficiente p/ gerar identificador\n");
284                     exit(1);
285                 }
286                 if ((token_reconhecido.pos_tab_simbolo = instaleIdenTabSimbolo(tab_simbolo,
287 &identificador)) == -1)
288                 {
289                     printf("Espaco de memoria insuficiente p/ instalar identificador na tabela
290 de simbolos\n");
291                     exit(1);
292                 }
293                 break;
294             //---------------------------------------------------------------------
295             case 8:
296                 erro.simbolo = '.';
297                 erro.lin = pos_inicial_simbolo.lin;
```

```
298                 erro.col = pos_inicial_simbolo.col;
299                 erro.id  = tErro_ponto_isolado;
300                 instaleErroLex(&tab_erro_identificado, &erro);
301                 devolva(-2);
302                 reinicieLexema(&lexema);
303                 break;
304             //--------------------------------------------------------------------
305             case 10:
306                 devolva(-1);
307                 if (simbolo == EOF)
308                 {
309                     fseek(f_in, 1, SEEK_CUR);
310                 }
311                 reinicieLexema(&lexema);
312                 if ((token = palavraReservadaOuIden(lexema.nome)) == -1)
313                 {
314                     printf("Espaco de memoria insuficiente p/ identificar palavra reservada\n");
315                     exit(1);
316                 }
317                 if (token == tk_IDEN)
318                 {
319                     if (gereIdentificador(&identificador, lexema.nome, &token,
320 &pos_inicial_simbolo) == -1)
321                     {
322                         printf("Espaco de memoria insuficiente p/ gerar identificador\n");
323                         exit(1);
324                     }
325                     if ((token_reconhecido.pos_tab_simbolo = instaleIdenTabSimbolo(tab_simbolo,
326 &identificador)) == -1)
327                     {
328                         printf("Espaco de memoria insuficiente p/ instalar identificador na ta-
329 bela de simbolos\n");
330                         exit(1);
331                     }
332                 }
333                 break;
334             //--------------------------------------------------------------------
335             case 12:
336                 erro.simbolo = '"';
337                 erro.lin = pos_inicial_simbolo.lin;
338                 erro.col = pos_inicial_simbolo.col;
339                 erro.id  = tErro_cadeia_nao_fechada;
340                 instaleErroLex(&tab_erro_identificado, &erro);
341                 devolva(-1);
342             case 13:
343                 concateneSimboloLexema(&lexema, "\"");
344                 reinicieLexema(&lexema);
345                 token = tk_CADEIA;
346                 if (gereIdentificador(&identificador, lexema.nome, &token, &pos_inicial_simbolo)
347 == -1)
348                 {
349                     printf("Espaco de memoria insuficiente p/ gerar identificador\n");
350                     exit(1);
351                 }
352                 if ((token_reconhecido.pos_tab_simbolo = instaleIdenTabSimbolo(tab_simbolo,
353 &identificador)) == -1)
354                 {
355                     printf("Espaco de memoria insuficiente p/ instalar identificador na tabela
356 de simbolos\n");
357                     exit(1);
358                 }
359                 break;
360             //--------------------------------------------------------------------
361             case 15:
362                 token = tk_menor_igual;
363                 break;
364             //--------------------------------------------------------------------
365             case 16:
366                 token = tk_atrib;
367                 break;
368             //--------------------------------------------------------------------
369             case 17:
370                 token = tk_diferente;
371                 break;
372             //--------------------------------------------------------------------
```

```
373             case 18:
374                 devolva(-1);
375                 token = tk_menor;
376                 break;
377             //--------------------------------------------------------------------
378             case 20:
379                 token = tk_maior_igual;
380                 break;
381             //--------------------------------------------------------------------
382             case 21:
383                 devolva(-1);
384                 token = tk_maior;
385                 break;
386             //--------------------------------------------------------------------
387             case 23:
388                 token = tk_decr;
389                 break;
390             //--------------------------------------------------------------------
391             case 24:
392                 devolva(-1);
393                 token = tk_menos;
394                 break;
395             //--------------------------------------------------------------------
396             case 26:
397                 token = tk_incr;
398                 break;
399             //--------------------------------------------------------------------
400             case 27:
401                 devolva(-1);
402                 token = tk_mais;
403                 break;
404             //--------------------------------------------------------------------
405             case 28:
406                 token = tk_igual;
407                 break;
408             //--------------------------------------------------------------------
409             case 29:
410                 token = tk_vezes;
411                 break;
412             //--------------------------------------------------------------------
413             case 30:
414                 token = tk_pt_virg;
415                 break;
416             //--------------------------------------------------------------------
417             case 31:
418                 token = tk_virg;
419                 break;
420             //--------------------------------------------------------------------
421             case 32:
422                 token = tk_dois_pts;
423                 break;
424             //--------------------------------------------------------------------
425             case 33:
426                 token = tk_fecha_par;
427                 break;
428             //--------------------------------------------------------------------
429             case 35:
430                 devolva(-1);
431                 token = tk_abre_par;
432                 break;
433             //--------------------------------------------------------------------
434             case 36:
435             case 37:
436
437                 // Tratamento p/ erro de comentário de bloco não fechado
438                 if (simbolo == '\n' && controle_bloco.existe_barra_n == 0)
439                 {
440                     controle_bloco.pos_barra_n = cursor.pos;
441                     controle_bloco.existe_barra_n = 1;
442                 }
443                 else if (controle_bloco.existe_barra_n)
444                     controle_bloco.qnt_char_apos_barra_n ++;
445
446                 break;
447             //--------------------------------------------------------------------
```

```
448             case 38:
449                 erro.simbolo = '(';
450                 erro.lin = pos_inicial_simbolo.lin;
451                 erro.col = pos_inicial_simbolo.col;
452                 erro.id  = tErro_comentario_bloco_nao_fechado;
453                 instaleErroLex(&tab_erro_identificado, &erro);
454 
455                 // Recuperação de erro p/ comentário de bloco não fechado
456                 fseek(f_in, -(++controle_bloco.qnt_char_apos_barra_n), SEEK_CUR);
457                 cursor.pos.lin = --controle_bloco.pos_barra_n.lin;
458                 break;
459             //-------------------------------------------------------------------
460             case 40:
461                 devolva(-1);
462                 token = tk_dividido;
463                 break;
464             //-------------------------------------------------------------------
465             case 42:
466                 token = tk_EOF;
467                 break;
468             //-------------------------------------------------------------------
469             case 43:
470                 erro.simbolo = simbolo;
471                 erro.lin = pos_inicial_simbolo.lin;
472                 erro.col = pos_inicial_simbolo.col;
473                 erro.id  = tErro_caractere_invalido;
474                 instaleErroLex(&tab_erro_identificado, &erro);
475                 devolva(-1);
476                 break;
477             //-------------------------------------------------------------------
478             // Default p/ estados: 14 - 19 - 22 - 25 - 34 - 39 - 41
479             default:
480                 break;
481         }
482 
483         if (token != -1)
484         {
485             token_reconhecido.lin = pos_inicial_simbolo.lin;
486             token_reconhecido.col = pos_inicial_simbolo.col;
487             token_reconhecido.id  = token;
488 
489             if (instaleToken(&tab_token_reconhecido, &token_reconhecido) == -1)
490             {
491                 printf("Espaco de memoria insuficiente p/ armazenar token\n");
492                 exit(1);
493             }
494             return token;
495         }
496 
497         leiaProximoSimbolo(&simbolo);
498         estado = transicao[estado][char2Simbolo(&simbolo)];
499 
500     }
501 
502 } // end-analisadorLexico (void)
503 
504 
505 /*
506  *---------------------------------------------------------------------
507  *
508  *   File    : lexema.h
509  *   Created : 2017-10-28
510  *   Modified: 2017-10-28
511  *
512  *   Header para manipulação dos lexemas p/ o analisador léxico de
513  *   Portugol
514  *
515  *---------------------------------------------------------------------
516  */
517 
518 #ifndef _LEXEMA_H
519 #define _LEXEMA_H
520 
521 /**
522  * ------------------------------------------------
```

```
 * INCLUDES
 * ------------------------------------------------
 */

#include <stdlib.h>
#include <string.h>

/**
 * ------------------------------------------------
 * MACROS
 * ------------------------------------------------
 */

#define TAM_INICIAL_LEXEMA 25

/**
 * ------------------------------------------------
 * TIPOS
 * ------------------------------------------------
 */

typedef struct
{
    char   *nome;
    int    indice_ult_registro;
    int    maior_lex;
    size_t tam_lexema;
} tLexema;

/**
 * ------------------------------------------------
 * PROTÓTIPOS DE FUNÇÕES
 * ------------------------------------------------
 */

char * inicializeLexema     (tLexema *lexema);
int  concateneSimboloLexema (tLexema *lexema, char *simbolo);
void reinicieLexema         (tLexema *lexema);
void finalizeLexema             (tLexema *lexema);

/**
 * ------------------------------------------------
 * DECLARAÇÕES DE FUNÇÕES
 * ------------------------------------------------
 */

/**
 * Função para inicializar lexema com TAM_INICIAL_LEXEMA
 * definido e retorna o ponteiro p/ lexema.nome (endereço
 * do primeiro indice do nome), ou seja, NULL caso não
 * exista memória suficiente.
 * @param tLexema *lexema
 * @return char *
 */

char * inicializeLexema (tLexema *lexema)
{
    lexema->maior_lex  = 0;
    lexema->indice_ult_registro = -1;
    lexema->tam_lexema = TAM_INICIAL_LEXEMA;
    lexema->nome = (char *) malloc (lexema->tam_lexema * sizeof(char) + 1);

    return lexema->nome;

} // end-inicializeLexema (tLexema *lexema)

/**
 * Função para cocatenar um caractere ao nome de um tLexema
 * realocando espaço caso não exista. Retorna o novo espaço
 * disponível para armazenar o nome do lexema ou -1 caso
 * ocorra erro ao cocatenar.
 * @param tLexema *lexema, char *simbolo
 * @return int
 */

```

```
598 int concateneSimboloLexema (tLexema *lexema, char *simbolo)
599 {
600        if ((lexema->indice_ult_registro + 1) == lexema->tam_lexema)
601        {
602               lexema->tam_lexema <<= 1;
603
604               lexema->nome = (char *) realloc (lexema->nome, (lexema->tam_lexema * sizeof(char)
605 + 1));
606               if (lexema->nome == NULL)
607                      return -1;
608        }
609
610        lexema->nome[ ++ lexema->indice_ult_registro ] = *simbolo;
611        lexema->nome[ lexema->indice_ult_registro + 1] = '\0';
612
613        return lexema->tam_lexema;
614
615 } // end-concateneSimboloLexema (tLexema *lexema, char *simbolo)
616
617 /**
618  * Função reinicia o lexema, alterando o indice do ultimo
619  * registro para -1.
620  * @param tLexema *lexema
621  * @return void
622  */
623
624 void reinicieLexema (tLexema *lexema)
625 {
626        size_t tam_lex = strlen(lexema->nome);
627
628     lexema->indice_ult_registro = -1;
629
630     if (tam_lex > lexema->maior_lex)
631        lexema->maior_lex = tam_lex;
632
633 } // end-reinicieLexema (tLexema *lexema)
634
635 /**
636  * Função finaliza o lexema desalocando o espaço de memória
637  * reservado para o mesmo.
638  * @param tLexema *lexema
639  * @return void
640  */
641
642 void finalizeLexema (tLexema *lexema)
643 {
644        free(lexema->nome);
645
646 } // end-finalizeLexema (tLexema *lexema)
647
648 #endif
649
650
651 /*
652  *--------------------------------------------------------------------
653  *
654  *   File    : tokens.h
655  *   Created : 2017-10-28
656  *   Modified: 2017-10-28
657  *
658  *   Header para manipulação de Tokens do analisador léxico
659  *      da linguagem Portugol
660  *
661  *--------------------------------------------------------------------
662  */
663
664 #ifndef _TOKENS_H
665 #define _TOKENS_H
666
667 /**
668  * ------------------------------------------------
669  * INCLUDES
670  * ------------------------------------------------
671  */
672
```

```
673 #include <ctype.h>
674 #include <stdlib.h>
675 #include <string.h>
676
677 /**
678  * ------------------------------------------------
679  * MACROS
680  * ------------------------------------------------
681  */
682
683 #define QNT_TOKENS 41
684 #define TAM_INICIAL_TAB_TOKEN 100
685 //#define QNT_PALAVRAS_RESERVADAS 18
686
687 /**
688  * ------------------------------------------------
689  * TIPOS
690  * ------------------------------------------------
691  */
692
693 typedef enum
694 {
695        tk_EOF = 1,       // 1
696        tk_IDEN,          // 2
697        tk_INTEIRO,       // 3
698        tk_DECIMAL,       // 4
699        tk_CADEIA,        // 5
700        tk_inicio,        // 6
701        tk_fim,           // 7
702        tk_int,           // 8
703        tk_dec,           // 9
704        tk_leia,          // 10
705        tk_imprima,       // 11
706        tk_para,          // 12
707        tk_de,            // 13
708        tk_ate,           // 14
709        tk_passo,         // 15
710        tk_fim_para,   // 16
711        tk_se,            // 17
712        tk_entao,         // 18
713        tk_senao,         // 19
714        tk_fim_se,        // 20
715        tk_e,             // 21
716        tk_ou,            // 22
717        tk_nao,           // 23
718        tk_virg,          // 24
719        tk_pt_virg,       // 25
720        tk_dois_pts,   // 26
721        tk_abre_par,   // 27
722        tk_fecha_par,  // 28
723        tk_menor,         // 29
724        tk_menor_igual,// 30
725        tk_maior,         // 31
726        tk_maior_igual,// 32
727        tk_diferente,  // 33
728        tk_igual,         // 34
729        tk_incr,          // 35
730        tk_decr,          // 36
731        tk_atrib,         // 37
732        tk_mais,          // 38
733        tk_menos,         // 39
734        tk_vezes,         // 40
735        tk_dividido       // 41
736 } tToken;
737
738 typedef struct
739 {
740     int    lin;
741     int    col;
742     int    pos_tab_simbolo;
743     tToken id;
744 } tTkReconhecido;
745
746 typedef struct
747 {
```

```
748         int    indice_ult_registro;
749     size_t tam_tabela;
750     tTkReconhecido *token;
751 } tTabTkReconhecido;
752 
753 /**
754  * ------------------------------------------------
755  * PROTÓTIPOS DE FUNÇÕES
756  * ------------------------------------------------
757  */
758 
759 char   * devolvaNomeToken          (tToken *token);
760 int    inicializeTabTkReconhecido (tTabTkReconhecido *tab_token);
761 int    instaleToken                  (tTabTkReconhecido *tab_token, tTkReconhecido *token);
762 void   finalizeTabTKReconhecido   (tTabTkReconhecido *tab_token);
763 tToken palavraReservadaOuIden     (char *lexema);
764 
765 /**
766  * ------------------------------------------------
767  * DECLARAÇÕES DE FUNÇÕES
768  * ------------------------------------------------
769  */
770 
771 /**
772  * Função devolve o nome do seu respectivo tToken.
773  * Retorna NULL caso não exista o token informado.
774  * @param tToken *token
775  * @return char *
776  */
777 
778 char * devolvaNomeToken (tToken *token)
779 {
780         switch (*token)
781         {
782                 case tk_EOF:           return "tk_EOF";
783                 case tk_IDEN:          return "tk_IDEN";
784                 case tk_INTEIRO:       return "tk_INTEIRO";
785                 case tk_DECIMAL:       return "tk_DECIMAL";
786                 case tk_CADEIA:        return "tk_CADEIA";
787                 case tk_inicio:        return "tk_inicio";
788                 case tk_fim:           return "tk_fim";
789                 case tk_int:           return "tk_int";
790                 case tk_dec:           return "tk_dec";
791                 case tk_leia:          return "tk_leia";
792                 case tk_imprima:       return "tk_imprima";
793                 case tk_para:          return "tk_para";
794                 case tk_de:            return "tk_de";
795                 case tk_ate:           return "tk_ate";
796                 case tk_passo:                return "tk_passo";
797                 case tk_fim_para:      return "tk_fim_para";
798                 case tk_se:            return "tk_se";
799                 case tk_entao:                return "tk_entao";
800                 case tk_senao:                return "tk_senao";
801                 case tk_fim_se:        return "tk_fim_se";
802                 case tk_e:                    return "tk_e";
803                 case tk_ou:            return "tk_ou";
804                 case tk_nao:           return "tk_nao";
805                 case tk_virg:          return "tk_virg";
806                 case tk_pt_virg:       return "tk_pt_virg";
807                 case tk_dois_pts:      return "tk_dois_pts";
808                 case tk_abre_par:      return "tk_abre_par";
809                 case tk_fecha_par:    return "tk_fecha_par";
810                 case tk_menor:                return "tk_menor";
811                 case tk_menor_igual: return "tk_menor_igual";
812                 case tk_maior:                return "tk_maior";
813                 case tk_maior_igual: return "tk_maior_igual";
814                 case tk_diferente:     return "tk_diferente";
815                 case tk_igual:                return "tk_igual";
816                 case tk_incr:          return "tk_incr";
817                 case tk_decr:          return "tk_decr";
818                 case tk_atrib:                return "tk_atrib";
819                 case tk_mais:          return "tk_mais";
820                 case tk_menos:                return "tk_menos";
821                 case tk_vezes:         return "tk_vezes";
822                 case tk_dividido:      return "tk_dividido";
```

```
823         }
824
825         return NULL;
826
827 } // end-devolvaNomeToken (tToken *token)
828
829 /**
830  * Função para inicializar tabela de tokens reconhecido
831  * com TAM_INICIAL_TAB_TOKEN definido. Retorna 0 caso a
832  * inicialização ocorra com sucesso ou -1 caso contrário.
833  * @param tTabTkReconhecido *tab_token
834  * @return int
835  */
836
837 int inicializeTabTkReconhecido (tTabTkReconhecido *tab_token)
838 {
839         tab_token->indice_ult_registro = -1;
840     tab_token->tam_tabela = TAM_INICIAL_TAB_TOKEN;
841     tab_token->token = (tTkReconhecido *) malloc (tab_token->tam_tabela * size-
842 of(tTkReconhecido));
843
844     if (tab_token->token == NULL)
845        return -1;
846
847     return 0;
848
849 } // end-inicializeTabTkReconhecido (tTabTkReconhecido *tab_token)
850
851 /**
852  * Função para instalar um token reconhecido (tTkReconhecido) na
853  * tabela de token reconhecido (tTabTkReconhecido), realocando
854  * espaço na tabela caso não exista. Retorna o índice onde o
855  * último token foi inserido na tabela ou -1 se erro durante a
856  * inserção na tabela.
857  * @param tTabTkReconhecido *tab_token, tTkReconhecido *token
858  * @return int
859  */
860
861 int instaleToken (tTabTkReconhecido *tab_token, tTkReconhecido *token)
862 {
863         if ((tab_token->indice_ult_registro + 1) == tab_token->tam_tabela)
864         {
865                 tab_token->tam_tabela <<= 1;
866
867                 tab_token->token = (tTkReconhecido *) realloc (tab_token->token, (tab_token-
868 >tam_tabela * sizeof(tTkReconhecido)));
869                 if (tab_token->token == NULL)
870                         return -1;
871         }
872
873         tab_token->token[ ++ tab_token->indice_ult_registro ] = *token;
874
875         return tab_token->indice_ult_registro;
876
877 } // end-instaleToken (tTabTkReconhecido *tab_token, tTkReconhecido *token)
878
879 /**
880  * Função finaliza a tabela de tokens reconhecidos desalocando
881  * o espaço de memória reservado para o mesmo.
882  * @param tTabTkReconhecido *tab_token
883  * @return void
884  */
885
886 void finalizeTabTKReconhecido (tTabTkReconhecido *tab_token)
887 {
888         free(tab_token->token);
889
890 } // end-finalizeTabTKReconhecido (tTabTkReconhecido *tab_token)
891
892 /**
893  * Função verifica se o identificador é uma palavra reservada através
894  * do lexema de entrada e retorna o token respectivo.
895  * @param char *lexema
896  * @return tToken
897  */
```

```
898
899 tToken palavraReservadaOuIden (char *lexema)
900 {
901         tToken token;
902         int    i;
903         size_t tam_lexema = strlen(lexema);
904         // Variável auxiliar p/ ignorar caixa alta/baixa em palavras reservadas
905         char   *aux = (char *) malloc (tam_lexema * sizeof(char) + 1);
906
907         if (aux == NULL)
908                 return -1;
909
910         for (i = 0; i <= tam_lexema; i++)
911                 aux[i] = tolower(lexema[i]);
912
913         // Verifica se eh palavra reservada ou identificador
914         if              (strcmp(aux,   "inicio") == 0)      token = tk_inicio;
915         else if (strcmp(aux,    "fim") == 0)token = tk_fim;
916         else if (strcmp(aux,    "int") == 0)token = tk_int;
917         else if (strcmp(aux,    "dec") == 0)token = tk_dec;
918         else if (strcmp(aux,   "leia") == 0)token = tk_leia;
919         else if (strcmp(aux,  "imprima") == 0)     token = tk_imprima;
920         else if (strcmp(aux,   "para") == 0)token = tk_para;
921         else if (strcmp(aux,     "de") == 0)token = tk_de;
922         else if (strcmp(aux,    "ate") == 0)token = tk_ate;
923         else if (strcmp(aux,    "passo") == 0)     token = tk_passo;
924         else if (strcmp(aux, "fim_para") == 0)     token = tk_fim_para;
925         else if (strcmp(aux,     "se") == 0)token = tk_se;
926         else if (strcmp(aux,    "entao") == 0)     token = tk_entao;
927         else if (strcmp(aux,    "senao") == 0)     token = tk_senao;
928         else if (strcmp(aux,   "fim_se") == 0)     token = tk_fim_se;
929         else if (strcmp(aux,        "e") == 0)     token = tk_e;
930         else if (strcmp(aux,     "ou") == 0)token = tk_ou;
931         else if (strcmp(aux,    "nao") == 0)token = tk_nao;
932         else token = tk_IDEN;
933
934         free(aux);
935         return token;
936
937 } // end-palavraReservadaOuIden (char *lexema)
938
939 #endif
940
941
942 /*
943  *---------------------------------------------------------------------
944  *
945  *   File    : tabSimbolos.h
946  *   Created : 2017-10-28
947  *   Modified: 2017-10-28
948  *
949  *   Header para manipulação da tabela de símbolos do analisador
950  *      léxico da linguagem Portugol
951  *
952  *---------------------------------------------------------------------
953  */
954
955 #ifndef _TABSIMBOLOS_H
956 #define _TABSIMBOLOS_H
957
958 /**
959  * ------------------------------------------------
960  * INCLUDES
961  * ------------------------------------------------
962  */
963
964 #include <stdlib.h>
965 #include <string.h>
966 #include "tokens.h"
967
968 /**
969  * ------------------------------------------------
970  * MACROS
971  * ------------------------------------------------
972  */
```

```
973
974  #define TAM_TAB_SIMBOLOS 257
975  #define TAM_INICIAL_OCOR 10
976
977  /**
978   * ------------------------------------------------
979   * TIPOS
980   * ------------------------------------------------
981   */
982
983  typedef struct
984  {
985         int lin;
986         int col;
987  } tPos;
988
989  typedef struct tIdentificador
990  {
991         char   *lex_char;
992         tToken token;
993         int    indice_ult_ocor;
994         int    lex_int;
995         float  lex_float;
996         size_t tam_ocor;
997         tPos   *ocor;
998         struct tIdentificador *prox;
999  } tIdentificador;
1000
1001 typedef tIdentificador * tTabSimbolo[TAM_TAB_SIMBOLOS];
1002
1003 /**
1004  * ------------------------------------------------
1005  * PROTÓTIPOS DE FUNÇÕES
1006  * ------------------------------------------------
1007  */
1008
1009 int  instaleIdenTabSimbolo          (tTabSimbolo tab_simbolo, tIdentificador *iden);
1010 int  instaleOcorIdentificador       (tIdentificador *iden_dest, tPos *ocor);
1011 int  gereIdentificador                     (tIdentificador *iden, char *lex, tToken *tk, tPos
1012 *ocor);
1013 void finalizeTabSimbolo             (tTabSimbolo tab_simbolo);
1014 unsigned int hash                          (char *string);
1015 tIdentificador * aloqueECopieIdentificador (tIdentificador *iden_src);
1016
1017 /**
1018  * ------------------------------------------------
1019  * DECLARAÇÕES DE FUNÇÕES
1020  * ------------------------------------------------
1021  */
1022
1023 /**
1024  * Função para instalar um identificador c/ lexema (tIdentificador)
1025  * na tabela de simbolos (tTabSimbolo), realocando espaço tPos *ocor
1026  * caso não exista. Retorna -1 caso ocorra erro durante alocação,
1027  * caso contrário retorna a posição em que o identificador foi
1028  * instalado na tabela de simbolos.
1029  * @param tTabSimbolo tab_simbolo, tIdentificador *iden
1030  * @return int
1031  */
1032
1033 int instaleIdenTabSimbolo (tTabSimbolo tab_simbolo, tIdentificador *iden)
1034 {
1035        tIdentificador *iden_atual;
1036        tIdentificador *iden_anterior;
1037        unsigned int   cod_hash = hash(iden->lex_char);
1038
1039        if ((tab_simbolo[cod_hash]) == NULL)
1040        {
1041               if ((tab_simbolo[cod_hash] = aloqueECopieIdentificador(iden)) == NULL)
1042                      return -1;
1043        }
1044        else
1045        {
1046
1047               iden_atual = tab_simbolo[cod_hash];
```

```
1048
1049                while(iden_atual != NULL)
1050                {
1051                        if (iden_atual->token == iden->token && strcmp(iden_atual->lex_char, iden-
1052 >lex_char) == 0)
1053                                return instaleOcorIdentificador (iden_atual, iden->ocor);
1054
1055                        iden_anterior = iden_atual;
1056                        iden_atual = iden_atual->prox;
1057                }
1058
1059                if ((iden_atual = aloqueECopieIdentificador(iden)) == NULL)
1060                        return -1;
1061                iden_anterior->prox = iden_atual;
1062
1063        }
1064
1065        return cod_hash;
1066
1067 } // end-instaleIdenTabSimbolo (tTabSimbolo tab_simbolo, tIdentificador *iden)
1068
1069 /**
1070  * Função para instalar uma nova ocorrência do identificador. Recebe
1071  * por parametro o identificador de destino (iden_dest) e a posição
1072  * da ocorrência. Retorna -1 se erro ou 0, caso contrário.
1073  * @param tIdentificador *iden_dest, tPos *ocor
1074  * @return int
1075  */
1076
1077 int instaleOcorIdentificador (tIdentificador *iden_dest, tPos *ocor)
1078 {
1079        if ((iden_dest->indice_ult_ocor + 1) == iden_dest->tam_ocor)
1080        {
1081                iden_dest->tam_ocor <<= 1;
1082
1083                iden_dest->ocor = (tPos *) realloc (iden_dest->ocor, (iden_dest->tam_ocor * size-
1084 of(tPos)));
1085                if (iden_dest->ocor == NULL)
1086                        return -1;
1087        }
1088
1089        iden_dest->ocor[ ++ iden_dest->indice_ult_ocor ] = ocor[0];
1090
1091        return 0;
1092
1093 } // end-instaleOcorIdentificador (tIdentificador *iden_dest, tPos *ocor)
1094
1095 /**
1096  * Função para gerar um tIdentificador *iden a partir dos
1097  * dados passados por parâmetro. Retorna -1 caso erro ou 0,
1098  * caso contrário.
1099  * @param tIdentificador *iden, char *lex, tToken *tk, tPos *ocor
1100  * @return int
1101  */
1102
1103 int gereIdentificador (tIdentificador *iden, char *lex, tToken *tk, tPos *ocor)
1104 {
1105        size_t tam_lex = strlen(lex);
1106
1107        iden->lex_char = (char *) malloc (tam_lex * sizeof(char) + 1);
1108        if (iden->lex_char == NULL)
1109                return -1;
1110
1111        iden->ocor = (tPos *) malloc (sizeof(tPos));
1112        if (iden->ocor == NULL)
1113                return -1;
1114
1115        if (*tk == tk_INTEIRO)
1116        {
1117                iden->lex_int   = atoi(lex);
1118                iden->lex_float = -1;
1119        }
1120        else if (*tk == tk_DECIMAL)
1121        {
1122                iden->lex_float = atof(lex);
```

```
1123                iden->lex_int   = -1;
1124        }
1125
1126        strcpy(iden->lex_char, lex);
1127        iden->token            = *tk;
1128        iden->tam_ocor                 = TAM_INICIAL_OCOR;
1129        iden->indice_ult_ocor = 0;
1130        iden->ocor[0]          = ocor[0];
1131
1132        return 0;
1133
1134 } // end-gereIdentificador (tIdentificador *iden, char *lex, tToken *tk, tPos *ocor)
1135
1136 /**
1137  * Função finaliza a tabela de simbolos desalocando
1138  * o espaço de memória reservado para o mesmo.
1139  * @param tTabSimbolo tab_simbolo
1140  * @return void
1141  */
1142
1143 void finalizeTabSimbolo (tTabSimbolo tab_simbolo)
1144 {
1145        tIdentificador *anterior;
1146        tIdentificador *atual;
1147        int i;
1148
1149     for (i = 0; i < TAM_TAB_SIMBOLOS; i++)
1150        {
1151               anterior = atual = tab_simbolo[i];
1152               while (atual != NULL)
1153               {
1154                      atual = atual->prox;
1155                      free(anterior->lex_char);
1156                      free(anterior->ocor);
1157                      free(anterior);
1158                      anterior = atual;
1159               }
1160        }
1161
1162 } // end-finalizeTabSimbolo (tTabSimbolo tab_simbolo)
1163
1164 /**
1165  * Função para hash c/ shift e compressão multiplique,
1166  * adicione e divida (MAD), em uma string recebida por
1167  * parametro, retornando o código hash (unsigned int).
1168  * @param char *string
1169  * @return unsigned int
1170  */
1171
1172 unsigned int hash (char *string)
1173 {
1174        unsigned int i;
1175        unsigned int h = 0;
1176        size_t tam = strlen(string);
1177
1178     for (i = 0; i < tam; i++)
1179        h = (h << 3) + string[i];
1180
1181     return (((7*h) + 5) % TAM_TAB_SIMBOLOS);
1182
1183 } // end-hash (char *string)
1184
1185 /**
1186  * Função para alocar memória e copiar dados de um identificador a outro.
1187  * Recebe por parametro o identificador de origem (iden_src), que possui
1188  * os dados. Retorna NULL se ocorrer erro na alocação ou retorna o endereço
1189  * inicial do tIdentificador alocado, caso contrário.
1190  * @param tIdentificador *iden_src
1191  * @return tIdentificador *
1192  */
1193
1194 tIdentificador * aloqueECopieIdentificador (tIdentificador *iden_src)
1195 {
1196        tIdentificador * iden;
1197        size_t tam_inicial_ocor = TAM_INICIAL_OCOR;
```

```
1198        size_t tam_lexema       = strlen(iden_src->lex_char);
1199
1200        iden = (tIdentificador *) malloc (sizeof(tIdentificador));
1201        if (iden == NULL)
1202               return NULL;
1203
1204        iden->lex_char = (char *) malloc (tam_lexema * sizeof(char) + 1);
1205        if (iden->lex_char == NULL)
1206               return NULL;
1207
1208        iden->ocor = (tPos *) malloc (tam_inicial_ocor * sizeof(tPos));
1209        if (iden->ocor == NULL)
1210               return NULL;
1211
1212        strcpy(iden->lex_char, iden_src->lex_char);
1213        iden->token            = iden_src->token;
1214        iden->tam_ocor             = iden_src->tam_ocor;
1215        iden->indice_ult_ocor = iden_src->indice_ult_ocor;
1216        iden->lex_int          = iden_src->lex_int;
1217        iden->lex_float        = iden_src->lex_float;
1218        iden->ocor[0]          = iden_src->ocor[0];
1219        iden->prox                    = NULL;
1220
1221        return iden;
1222
1223 } // end-aloqueECopieIdentificador (tIdentificador *iden_src)
1224
1225 #endif
1226
1227
1228 /*
1229  *--------------------------------------------------------------------
1230  *
1231  *   File    : errosLexicos.h
1232  *   Created : 2017-10-28
1233  *   Modified: 2017-10-28
1234  *
1235  *   Header para manipulação dos Erros léxicos possíveis
1236  *      da linguagem Portugol
1237  *
1238  *--------------------------------------------------------------------
1239  */
1240
1241 #ifndef _ERROSLEXICOS_H
1242 #define _ERROSLEXICOS_H
1243
1244 /**
1245  * ------------------------------------------------
1246  * MACROS
1247  * ------------------------------------------------
1248  */
1249
1250 #define QNT_ERROS_LEXICOS 5
1251 #define TAM_INICIAL_TAB_ERRO 20
1252
1253 /**
1254  * ------------------------------------------------
1255  * TIPOS
1256  * ------------------------------------------------
1257  */
1258
1259 typedef enum
1260 {
1261        tErro_ponto_isolado = 1,             // 1
1262        tErro_caractere_invalido,            // 2
1263        tErro_cadeia_nao_fechada,            // 3
1264        tErro_delimitador_esperado,          // 4
1265        tErro_comentario_bloco_nao_fechado// 5
1266 } tErroLex;
1267
1268 typedef struct
1269 {
1270        char simbolo;
1271        int  lin;
1272        int  col;
```

```
1273        tErroLex id;
1274  } tErroLexIdentificado;
1275
1276  typedef struct
1277  {
1278        int    indice_ult_registro;
1279        size_t tam_tabela;
1280        tErroLexIdentificado *erro;
1281  } tTabErroLexIdentificado;
1282
1283  /**
1284   * ------------------------------------------------
1285   * PROTÓTIPOS DE FUNÇÕES
1286   * ------------------------------------------------
1287   */
1288
1289  char * devolvaNomeErro    (tErroLex *erro);
1290  int  inicializeTabErroLex (tTabErroLexIdentificado *tab_erro);
1291  int  instaleErroLex    (tTabErroLexIdentificado *tab_erro, tErroLexIdentificado *erro);
1292  void finalizeTabErroLex   (tTabErroLexIdentificado *tab_erro);
1293
1294  /**
1295   * ------------------------------------------------
1296   * DECLARAÇÕES DE FUNÇÕES
1297   * ------------------------------------------------
1298   */
1299
1300  /**
1301   * Função devolve o nome do erro ao seu respectivo tErroLex.
1302   * NULL caso não exista o erro informado.
1303   * @param tErroLex *erro
1304   * @return char *
1305   */
1306
1307  char * devolvaNomeErro (tErroLex *erro)
1308  {
1309        switch (*erro)
1310        {
1311              case tErro_ponto_isolado:                          return "Ponto isolado";
1312              case tErro_caractere_invalido:                     return "Caractere invalido";
1313              case tErro_cadeia_nao_fechada:                     return "Cadeia nao fechada";
1314              case tErro_delimitador_esperado:            return "Delimitador esperado";
1315              case tErro_comentario_bloco_nao_fechado: return "Comentario de bloco nao fechado";
1316        }
1317
1318        return NULL;
1319
1320  } // end-devolvaNomeErro (tErroLex *erro)
1321
1322  /**
1323   * Função para inicializar tabela de erros identificados
1324   * com TAM_INICIAL_TAB_ERRO definido. Retorna 0 caso a
1325   * inicialização ocorra com sucesso ou -1, caso contrário.
1326   * @param tTabErroLexIdentificado *tab_erro
1327   * @return int
1328   */
1329
1330  int inicializeTabErroLex (tTabErroLexIdentificado *tab_erro)
1331  {
1332        tab_erro->indice_ult_registro = -1;
1333      tab_erro->tam_tabela = TAM_INICIAL_TAB_ERRO;
1334
1335      tab_erro->erro = (tErroLexIdentificado *) malloc (tab_erro->tam_tabela * size-
1336  of(tErroLexIdentificado));
1337      if (tab_erro->erro == NULL)
1338         return -1;
1339
1340      return 0;
1341
1342  } // end-inicializeTabErro (tTabErroLexIdentificado *tab_erro)
1343
1344  /**
1345   * Função para instalar um Erro léxico identificado
1346   * (tErroLexIdentificado) na tabela de erro identificado
1347   * (tTabErroLexIdentificado), realocando espaço na tabela
```

```
1348  * caso não exista. Retorna o índice onde o último erro
1349  * foi inserido na tabela ou -1 p/ erro durante a inserção
1350  * na tabela.
1351  * @param tTabErroLexIdentificado *tab_erro, tErroLexIdentificado *erro
1352  * @return int
1353  */
1354
1355 int instaleErroLex (tTabErroLexIdentificado *tab_erro, tErroLexIdentificado *erro)
1356 {
1357        if ((tab_erro->indice_ult_registro + 1) == tab_erro->tam_tabela)
1358        {
1359               tab_erro->tam_tabela <<= 1;
1360
1361               tab_erro->erro = (tErroLexIdentificado *) realloc (tab_erro->erro, (tab_erro-
1362 >tam_tabela * sizeof(tErroLexIdentificado)));
1363               if (tab_erro->erro == NULL)
1364                      return -1;
1365        }
1366
1367        tab_erro->erro[ ++ tab_erro->indice_ult_registro ] = *erro;
1368
1369        return tab_erro->indice_ult_registro;
1370
1371 } // end-instaleErroLex (tTabErroLexIdentificado *tab_erro, tErroLexIdentificado *erro)
1372
1373 /**
1374  * Função finaliza a tabela de erros identificados desalocando
1375  * o espaço de memória reservado para o mesmo.
1376  * @param tTabErroLexIdentificado *tab_erro
1377  * @return void
1378  */
1379
1380 void finalizeTabErroLex (tTabErroLexIdentificado *tab_erro)
1381 {
1382        free(tab_erro->erro);
1383
1384 } // end-finalizeTabErroLex (tTabErroLexIdentificado *tab_erro)
1385
1386 #endif
1387
1388 /*
1389  *---------------------------------------------------------------------
1390  *
1391  *   File    : tabTransicoes.h
1392  *   Created : 2017-10-28
1393  *   Modified: 2017-10-28
1394  *
1395  *   Header para manipulação da tabela de transições dos estados para
1396  *   analisador léxico de Portugol
1397  *
1398  *---------------------------------------------------------------------
1399  */
1400
1401 #ifndef _TABTRANSICOES_H
1402 #define _TABTRANSICOES_H
1403
1404 /**
1405  * ------------------------------------------------
1406  * INCLUDES
1407  * ------------------------------------------------
1408  */
1409
1410 #include <ctype.h>
1411
1412 /**
1413  * ------------------------------------------------
1414  * MACROS
1415  * ------------------------------------------------
1416  */
1417
1418 #define QNT_ESTADOS  44
1419 #define QNT_SIMBOLOS 21
1420
1421 /**
1422  * ------------------------------------------------
```

```
1423  * TIPOS
1424  * ------------------------------------------------
1425  */
1426
1427 typedef enum
1428 {
1429     s_digito,   // 0
1430     s_pt,       // 1
1431     s_letra,    // 2
1432     s_under,    // 3
1433     s_aspas,    // 4
1434     s_menor,    // 5
1435     s_maior,    // 6
1436     s_menos,    // 7
1437     s_mais,     // 8
1438     s_igual,    // 9
1439     s_vezes,    // 10
1440     s_pt_virg,  // 11
1441     s_virg,     // 12
1442     s_dois_pts, // 13
1443     s_fecha_par,// 14
1444     s_abre_par, // 15
1445     s_dividido, // 16
1446     s_EOF,      // 17
1447     s_barra_n,  // 18
1448     s_branco,   // 19
1449     s_outro     // 20
1450 } tSimbolo;
1451
1452 /**
1453  * ------------------------------------------------
1454  * VARIÁVEIS GLOBAIS
1455  * ------------------------------------------------
1456  */
1457
1458 unsigned int transicao[QNT_ESTADOS][QNT_SIMBOLOS] = {
1459 //    \D | . |\L | _ | " | < | > | - | + | = | * | ; | , | : | ) | ( | / |EOF|\n |\b |outro
1460     { 1,  7,  9, 43, 11, 14, 19, 22, 25, 28, 29, 30, 31, 32, 33, 34, 39, 42,  0,  0, 43}, //
1461 estado  0
1462     { 1,  4,  3,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2}, //
1463 estado  1
1464     { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0}, //
1465 estado  2
1466     { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0}, //
1467 estado  3
1468     { 4,  6,  5,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6}, //
1469 estado  4
1470     { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0}, //
1471 estado  5
1472     { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0}, //
1473 estado  6
1474     { 4,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8}, //
1475 estado  7
1476     { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0}, //
1477 estado  8
1478     { 9, 10,  9,  9, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10}, //
1479 estado  9
1480     { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0}, //
1481 estado 10
1482     {11, 11, 11, 11, 13, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 12, 12, 11, 11}, //
1483 estado 11
1484     { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0}, //
1485 estado 12
1486     { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0}, //
1487 estado 13
1488     {18, 18, 18, 18, 18, 18, 17, 16, 18, 15, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18}, //
1489 estado 14
1490     { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0}, //
1491 estado 15
1492     { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0}, //
1493 estado 16
1494     { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0}, //
1495 estado 17
1496     { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0}, //
1497 estado 18
```

```
1498      {21, 21, 21, 21, 21, 21, 21, 21, 21, 20, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21}, //
1499  estado 19
1500      { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0}, //
1501  estado 20
1502      { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0}, //
1503  estado 21
1504      {24, 24, 24, 24, 24, 24, 24, 23, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24}, //
1505  estado 22
1506      { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0}, //
1507  estado 23
1508      { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0}, //
1509  estado 24
1510      {27, 27, 27, 27, 27, 27, 27, 27, 26, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27}, //
1511  estado 25
1512      { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0}, //
1513  estado 26
1514      { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0}, //
1515  estado 27
1516      { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0}, //
1517  estado 28
1518      { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0}, //
1519  estado 29
1520      { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0}, //
1521  estado 30
1522      { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0}, //
1523  estado 31
1524      { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0}, //
1525  estado 32
1526      { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0}, //
1527  estado 33
1528      {35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 36, 35, 35, 35, 35, 35, 35, 35, 35, 35}, //
1529  estado 34
1530      { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0}, //
1531  estado 35
1532      {36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 37, 36, 36, 36, 36, 36, 36, 38, 36, 36, 36}, //
1533  estado 36
1534      {36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 37, 36, 36, 36,  0, 36, 36, 38, 36, 36, 36}, //
1535  estado 37
1536      { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0}, //
1537  estado 38
1538      {40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 41, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40}, //
1539  estado 39
1540      { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0}, //
1541  estado 40
1542      {41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 42,  0, 41, 41, 41}, //
1543  estado 41
1544      { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0}, //
1545  estado 42
1546      { 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0}, //
1547  estado 43
1548  };
1549
1550  /**
1551   * ---------------------------------------------------
1552   * PROTÓTIPOS DE FUNÇÕES
1553   * ---------------------------------------------------
1554   */
1555
1556  tSimbolo char2Simbolo (char *caractere);
1557
1558  /**
1559   * ---------------------------------------------------
1560   * DECLARAÇÕES DE FUNÇÕES
1561   * ---------------------------------------------------
1562   */
1563
1564  /**
1565   * Função retorna o simbolo da tabela de transições
1566   * correspondente ao caractere de entrada.
1567   * @param char *caractere
1568   * @return tSimbolo
1569   */
1570
1571  tSimbolo char2Simbolo (char *caractere)
1572  {
```

```
1573      switch (*caractere)
1574      {
1575          case '.' :  return s_pt;
1576          case '_' :  return s_under;
1577          case '"' :  return s_aspas;
1578          case '<' :  return s_menor;
1579          case '>' :  return s_maior;
1580          case '-' :  return s_menos;
1581          case '+' :  return s_mais;
1582          case '=' :  return s_igual;
1583          case '*' :  return s_vezes;
1584          case ';' :  return s_pt_virg;
1585          case ',' :  return s_virg;
1586          case ':' :  return s_dois_pts;
1587          case ')' :  return s_fecha_par;
1588          case '(' :  return s_abre_par;
1589          case '/' :  return s_dividido;
1590          case EOF :  return s_EOF;
1591          case '\n':  return s_barra_n;
1592          default:
1593              if (isalpha(*caractere)) return s_letra;
1594              if (isdigit(*caractere)) return s_digito;
1595              if (isspace(*caractere)) return s_branco;
1596              return s_outro;
1597      }
1598
1599  } // end-char2Simbolo (char *caractere)
1600
1601  #endif
1602
1603  /*
1604   *--------------------------------------------------------------------
1605   *
1606   *   File    : geradorSaidas.h
1607   *   Created : 2017-10-28
1608   *   Modified: 2017-10-28
1609   *
1610   *   Header para manipulação das saídas do analisador léxico
1611   *      da linguagem Portugol
1612   *
1613   *--------------------------------------------------------------------
1614   */
1615
1616  #ifndef _GERADORSAIDAS_H
1617  #define _GERADORSAIDAS_H
1618
1619  /**
1620   * -------------------------------------------------
1621   * INCLUDES
1622   * -------------------------------------------------
1623   */
1624
1625  #include <stdio.h>
1626  #include <stdlib.h>
1627  #include <string.h>
1628  #include "lexema.h"
1629  #include "tokens.h"
1630  #include "tabSimbolos.h"
1631  #include "errosLexicos.h"
1632
1633  /**
1634   * -------------------------------------------------
1635   * PROTÓTIPOS DE FUNÇÕES
1636   * -------------------------------------------------
1637   */
1638
1639  int gereArquivoErro       (FILE *f_in, char *nome_f_in, tTabErroLexIdentificado *tab_erro_lex);
1640  int gereArquivoToken      (char *nome_f_in, tTabTkReconhecido *tab_token, tTabSimbolo
1641  tab_simbolo, int *maior_lex, int qnt_erros);
1642  int gereArquivoTabSimbolo (char *nome_f_in, tTabSimbolo tab_token, int *maior_lex);
1643
1644  /**
1645   * -------------------------------------------------
1646   * DECLARAÇÕES DE FUNÇÕES
1647   * -------------------------------------------------
```

```
1648  */
1649
1650 /**
1651  * Função gera o arquivo contendo a entrada e seu erros
1652  * léxicos informados viasualmente a partir do arquivo
1653  * de entrada. Recebe por parametro o fluxo p/ arquivo
1654  * de entrada, o nome do arquivo e a tabela de erros
1655  * léxicos identificados pelo analisador léxico. Retorna
1656  * -1 para erro e 0 caso contrário.
1657  * @param FILE *f_in, char *nome_f_in, tTabErroLexIdentificado *tab_erro_lex
1658  * @return int
1659  */
1660
1661 int gereArquivoErro (FILE *f_in, char *nome_f_in, tTabErroLexIdentificado *tab_erro_lex)
1662 {
1663     FILE *f_erro;
1664     char simbolo;
1665     char *nome_arq_erro;
1666     int  col;
1667     int  i = 0;
1668     int  linha = 1;
1669     size_t tam_linha = 0;
1670
1671     nome_arq_erro = (char *) malloc (strlen(nome_f_in) * sizeof(char) + 5);
1672     sprintf(nome_arq_erro, "%s.err", nome_f_in);
1673
1674     if ((f_erro = fopen(nome_arq_erro, "w")) == NULL)
1675            return -1;
1676
1677     rewind(f_in);
1678     fprintf(f_erro, "LISTA DE ERROS LEXICOS EM \"%s\"\n\n", nome_f_in);
1679
1680     while (feof(f_in) == 0)
1681     {
1682         if ((simbolo = fgetc(f_in)) != EOF)
1683         {
1684             fprintf(f_erro, "[%5d] ", linha);
1685             tam_linha ++;
1686
1687             while (simbolo != '\n' && simbolo != EOF)
1688             {
1689                 fprintf(f_erro, "%c", simbolo);
1690                 simbolo = fgetc(f_in);
1691                 tam_linha ++;
1692             }
1693
1694             if (tab_erro_lex->erro[i].lin == linha && i <= tab_erro_lex->indice_ult_registro)
1695             {
1696
1697                 fprintf(f_erro, "\n        ");
1698                 for (col = 1; col < tab_erro_lex->erro[i].col; col++)
1699                     fprintf(f_erro, "-");
1700                 fprintf(f_erro, "^\n");
1701
1702                 fprintf(f_erro, "        Erro lexico na linha %d coluna %d: %s '%c'",
1703                     tab_erro_lex->erro[i].lin,
1704                     tab_erro_lex->erro[i].col,
1705                     devolvaNomeErro(&tab_erro_lex->erro[i].id),
1706                     tab_erro_lex->erro[i].simbolo);
1707
1708                 i ++;
1709                 if (tab_erro_lex->erro[i].lin == linha)
1710                 {
1711                     fseek(f_in, -tam_linha, SEEK_CUR);
1712                     linha --;
1713                 }
1714
1715             }
1716
1717             fprintf(f_erro, "\n");
1718             linha ++;
1719             tam_linha = 0;
1720         }
1721     }
1722
```

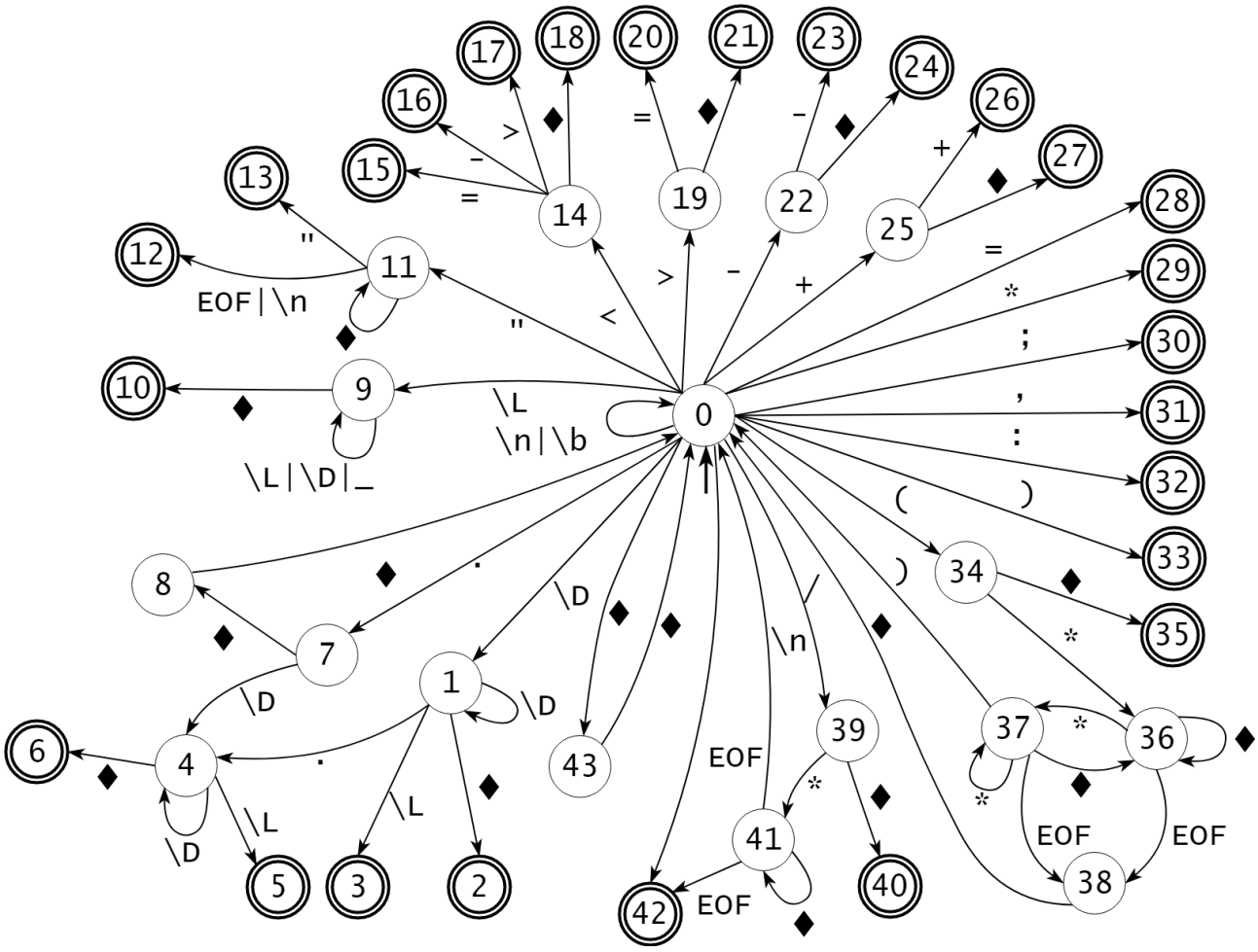
```
1723      fprintf(f_erro, "\nTOTAL DE ERROS: %d\n", tab_erro_lex->indice_ult_registro + 1);
1724
1725      free(nome_arq_erro);
1726      fclose(f_erro);
1727
1728      return 0;
1729
1730  } // end-gereArquivoErro (FILE *f_in, char *nome_f_in, tTabErroLexIdentificado *tab_erro_lex)
1731
1732  /**
1733   * Função gera o arquivo contendo os tokens reconhecidos
1734   * durante a analise léxica do arquivo de entrada. Recebe
1735   * por parametro o nome do arquivo, a tabela de tokens
1736   * reconhecidos pelo analisador léxico, a tabela de simbolos
1737   * geradas pelo analisado lexico e o tamanho do maior lexema.
1738   * Retorna -1 caso erro e 0 caso contrário.
1739   * @param char *nome_f_in, tTabTkReconhecido *tab_token, tTabSimbolo tab_simbolo, int
1740  *maior_lex, int qnt_erros
1741   * @return int
1742   */
1743
1744  int gereArquivoToken (char *nome_f_in, tTabTkReconhecido *tab_token, tTabSimbolo tab_simbolo,
1745  int *maior_lex, int qnt_erros)
1746  {
1747      FILE *f_token;
1748      char *nome_arq_token;
1749      int  i;
1750      int  j;
1751      int  linha = 0;
1752      int  total_token = 0;
1753      int  qnt_token[QNT_TOKENS + 1] = {0};
1754      tToken token;
1755      tIdentificador *iden;
1756
1757      nome_arq_token = (char *) malloc (strlen(nome_f_in) * sizeof(char) + 5);
1758      sprintf(nome_arq_token, "%s.tok", nome_f_in);
1759
1760      if ((f_token = fopen(nome_arq_token, "w")) == NULL)
1761             return -1;
1762
1763      fprintf(f_token, "LISTA DE TOKENS RECONHECIDOS EM \"%s\"\n\n", nome_f_in);
1764
1765      fprintf(f_token, "+-----+-----+-----+----------------+-");
1766      for (i = 0; i < *maior_lex; i++)
1767         fprintf(f_token, "-");
1768      fprintf(f_token, "-+-----------------+\n");
1769
1770      fprintf(f_token, "| LIN | COL | COD | %-14s | %-*s |  POS TAB SIMB  |\n",
1771         "TOKEN",
1772         *maior_lex,
1773         "LEXEMA");
1774
1775      fprintf(f_token, "+-----+-----+-----+----------------+-");
1776      for (i = 0; i < *maior_lex; i++)
1777         fprintf(f_token, "-");
1778      fprintf(f_token, "-+-----------------+\n");
1779
1780      for (i = 0; i <= tab_token->indice_ult_registro; i++)
1781      {
1782         qnt_token[tab_token->token[i].id] ++;
1783
1784         if (tab_token->token[i].lin == linha)
1785                fprintf(f_token,"|     |");
1786             else
1787                fprintf(f_token,"| %3d |", tab_token->token[i].lin);
1788
1789             fprintf(f_token," %3d | %3d | %-14s ", tab_token->token[i].col, tab_token-
1790  >token[i].id,
1791          devolvaNomeToken(&tab_token->token[i].id));
1792
1793             if (tab_token->token[i].pos_tab_simbolo == -1)
1794                fprintf(f_token, "| %-*s |                |\n", *maior_lex, " ");
1795       else
1796       {
1797          iden = tab_simbolo[tab_token->token[i].pos_tab_simbolo];
```

```
1798
1799            while(iden != NULL)
1800            {
1801                if (tab_token->token[i].id == iden->token)
1802                {
1803                    for (j = 0; j <= iden->indice_ult_ocor; j++)
1804                    {
1805                        if (tab_token->token[i].lin == iden->ocor[j].lin
1806                            && tab_token->token[i].col == iden->ocor[j].col)
1807                        {
1808                            fprintf(f_token, "| %-*s |      %4d      |\n", *maior_lex,
1809                                iden->lex_char, tab_token->token[i].pos_tab_simbolo);
1810                        }
1811                    }
1812                }
1813                iden = iden->prox;
1814            }
1815
1816        }
1817
1818        if (tab_token->token[i].lin != linha)
1819            linha = tab_token->token[i].lin;
1820    }
1821
1822    fprintf(f_token, "+-----+-----+-----+----------------+-");
1823    for (i = 0; i < *maior_lex; i++)
1824        fprintf(f_token, "-");
1825    fprintf(f_token, "-+----------------+\n");
1826
1827    fprintf(f_token, "RESUMO\n\n");
1828    fprintf(f_token, "+-----+----------------+------+\n");
1829    fprintf(f_token, "| COD | TOKEN          | USOS |\n");
1830    fprintf(f_token, "+-----+----------------+------+\n");
1831
1832    for (token = 1; token <= QNT_TOKENS; token++)
1833    {
1834        fprintf(f_token, "| %3d | %-14s | %4d |\n", token, devolvaNomeToken(&token),
1835 qnt_token[token]);
1836        total_token += qnt_token[token];
1837    }
1838
1839    fprintf(f_token, "+-----+----------------+------+\n");
1840    fprintf(f_token, "|   0 | TOTAL          | %4d |\n", total_token);
1841    fprintf(f_token, "+-----+----------------+------+\n\n");
1842
1843    fprintf(f_token, "TOTAL DE ERROS: %d\n\n", qnt_erros);
1844
1845    free(nome_arq_token);
1846    fclose(f_token);
1847
1848    return 0;
1849
1850 } // end-gereArquivoToken (char *nome_f_in, tTabTkReconhecido *tab_token, tTabSimbolo
1851 tab_simbolo, int *maior_lex, int qnt_erros)
1852
1853 /**
1854  * Função gera o arquivo contendo a tabela de simbolos criada
1855  * durante a analise léxica do arquivo de entrada. Recebe por
1856  * parametro o nome do arquivo, e a tabela de simbolos.
1857  * Retorna -1 caso erro e 0 caso contrário.
1858  * @param char *nome_f_in, tTabSimbolo tab_simbolo
1859  * @return int
1860  */
1861
1862 int gereArquivoTabSimbolo (char *nome_f_in, tTabSimbolo tab_simbolo, int *maior_lex)
1863 {
1864     FILE *f_simbolo;
1865     char *nome_arq_simbolo;
1866     int  i;
1867     int  j;
1868     int  maior_ocor = 0;
1869     tIdentificador *iden;
1870     tToken token;
1871
1872     nome_arq_simbolo = (char *) malloc (strlen(nome_f_in) * sizeof(char) + 5);
```

```
1873      sprintf(nome_arq_simbolo, "%s.tbl", nome_f_in);
1874
1875      if ((f_simbolo = fopen(nome_arq_simbolo, "w")) == NULL)
1876            return -1;
1877
1878      // Verifica qual maior ocorrência
1879      for (i = 0; i < TAM_TAB_SIMBOLOS; i++)
1880          if (tab_simbolo[i])
1881              if (tab_simbolo[i]->indice_ult_ocor > maior_ocor)
1882                  maior_ocor = tab_simbolo[i]->indice_ult_ocor;
1883      maior_ocor ++; // conta o indice 0
1884      maior_ocor *= 10; // tam que (lin,col) ocupa na tabela de simbolos
1885
1886      fprintf(f_simbolo, "TABELA DE SIMBOLOS - \"%s\"\n\n", nome_f_in);
1887
1888      fprintf(f_simbolo, "+-----+----------------+-");
1889      for (i = 0; i < *maior_lex; i++)
1890          fprintf(f_simbolo, "-");
1891      fprintf(f_simbolo, "-+-");
1892      for (i = 0; i < maior_ocor; i++)
1893          fprintf(f_simbolo, "-");
1894      fprintf(f_simbolo, "+\n");
1895
1896      fprintf(f_simbolo,"| POS | %-14s | %-*s | %-*s|\n", "TOKEN", *maior_lex,
1897          "LEXEMA", maior_ocor, "POS NA ENTRADA (linha,coluna)");
1898
1899      fprintf(f_simbolo, "+-----+----------------+-");
1900      for (i = 0; i < *maior_lex; i++)
1901          fprintf(f_simbolo, "-");
1902      fprintf(f_simbolo, "-+-");
1903      for (i = 0; i < maior_ocor; i++)
1904          fprintf(f_simbolo, "-");
1905      fprintf(f_simbolo, "+\n");
1906
1907
1908      for (i = 0; i < TAM_TAB_SIMBOLOS; i++)
1909      {
1910          iden = tab_simbolo[i];
1911
1912          while(iden != NULL)
1913          {
1914              token = iden->token;
1915              fprintf(f_simbolo, "| %3d | %-14s | %-*s | ", i, devolvaNomeToken(&token),
1916  *maior_lex,
1917                  iden->lex_char);
1918              for (j = 0; j <= iden->indice_ult_ocor; j++)
1919                  fprintf(f_simbolo, "(%3d,%3d) ", iden->ocor[j].lin, iden->ocor[j].col);
1920              fprintf(f_simbolo, "%*s|\n", maior_ocor - ((iden->indice_ult_ocor + 1)* 10), "");
1921              iden = iden->prox;
1922          }
1923
1924      }
1925
1926      fprintf(f_simbolo, "+-----+----------------+-");
1927      for (i = 0; i < *maior_lex; i++)
1928          fprintf(f_simbolo, "-");
1929      fprintf(f_simbolo, "-+-");
1930      for (i = 0; i < maior_ocor; i++)
1931          fprintf(f_simbolo, "-");
1932      fprintf(f_simbolo, "+\n");
1933
1934      free(nome_arq_simbolo);
1935      fclose(f_simbolo);
1936
1937      return 0;
1938
1939  } // end-gereArquivoTabSimbolo (char *nome_f_in, tTabSimbolo tab_simbolo)
1940
1941  #endif
```

2. Autômato

2.1. Diagrama de estados



2.2. Ações

	Ações a serem efetuadas					
Estado	Devolver n caracteres à entrada	Emitir msg de erro (código do erro)	instalar lexema na tabela de símbolos	Retornar token	Salvar linha e coluna inicial do símbolo	Restaurar lin, col e pos no arquivo
0					sim	
1						
2	1		sim	tk_INTEIRO		sim
3	1	4	sim	tk_INTEIRO		sim
4						
5	1	4	sim	tk_DECIMAL		sim
6	1		sim	tk_DECIMAL		sim
7						
8	2	1				sim
9						
10	1		sim	tk_IDEN		sim
11						
12	1	3	sim	tk_CADEIA		sim
13			sim	tk_CADEIA		
14						
15				tk_igual		
16				tk_atrib		
17				tk_diferente		
18	1			tk_menor		sim
19						
20				tk_maior_igual		
21	1			tk_maior		sim
22						
23				tk_dec		
24	1			tk_menos		sim
25						
26				tk_inc		
27	1			tk_mais		sim
28				tk_igual		
29				tk_vezes		
30				tk_pt_virg		
31				tk_virg		

32				tk_dois_pts		
33				tk_fecha_par		
34						
35	1			tk_abre_par		sim
36						
37						
38	n caracteres até primeira ocorrência do \n após comentário de bloco aberto.	5				sim
39						
40	1			tk_dividido		sim
41						
42				tk_EOF		
43	1	2				sim

Erros léxicos	
Código	**Mensagem**
1	Ponto isolado
2	Caractere inválido
3	Cadeia não fechada
4	Delimitador esperado
5	Comentário de bloco não fechado

2.3. Tabela de transições

Estado atual	Próximo estado																				
	\D	**.**	**\L**	**_**	**“**	**<**	**>**	**-**	**+**	**=**	*****	**;**	**,**	**:**	**)**	**(**	**/**	**EOF**	**\n**	**\b**	**♦**
0	1	7	9	43	11	14	19	22	25	28	29	30	31	32	33	34	39	42	0	0	43
1	1	4	3	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
4	4	6	5	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
7	4	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	9	10	9	9	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
11	11	11	11	11	13	11	11	11	11	11	11	11	11	11	11	11	11	12	12	11	11
14	18	18	18	18	18	18	17	16	18	15	18	18	18	18	18	18	18	18	18	18	18
19	21	21	21	21	21	21	21	21	21	20	21	21	21	21	21	21	21	21	21	21	21
22	24	24	24	24	24	24	24	23	24	24	24	24	24	24	24	24	24	24	24	24	24
25	27	27	27	27	27	27	27	27	26	27	27	27	27	27	27	27	27	27	27	27	27
34	35	35	35	35	35	35	35	35	35	35	36	35	35	35	35	35	35	35	35	35	35
36	36	36	36	36	36	36	36	36	36	36	37	36	36	36	36	36	36	38	36	36	36
37	36	36	36	36	36	36	36	36	36	36	37	36	36	36	0	36	36	38	36	36	36
38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
39	40	40	40	40	40	40	40	40	40	40	41	40	40	40	40	40	40	40	40	40	40
41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	42	0	41	41
43	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Legenda:

- \b Branco (tabulação e espaço)
- \L Letra
- \D Dígito
- E EOF
- ♦ Demais símbolos que não possuam suas respectivas transições no estado.

* Os estados não listados, entre 0 e 43, são estados finais

3. Resultados dos testes

3.1. Teste 1

3.1.1. Arquivo de entrada

```
Inicio
  int: num;
  Int: nUm, NuM;
  INT: Maior;
  imprima ("Digite 3 nros:");
  leia (num);
  Leia (nUm);
  LEIA (NuM);
  se (num>=nUm) e (num>=NuM) entao
    maior <- num;
  senao
  SE (nUm>=num) E (nUm>=NuM) ENTAO
    MAIOR <- nUm;
  SENAO
    Maior <- NuM;
  Fim_Se
  Imprima ("Maior = ");
  IMPRIMA (Maior);
  NuM <-    -.12 - +001 - 1234567890 + -.0 - +0. - 12345.67890 ;
  nUm<--.12-+001-1234567890+-.0-+0.-12345.67890;
  num<-+.;
  para   de 1ate xpasso1 s/**++*/e  2<3
  paraide1 ate9passo 1.2.3.<----(***))
  para i de 1ate 9. 0. passo 1$j
FIM
```

3.1.2. Erros léxicos

```
LISTA DE ERROS LEXICOS EM "teste-01.ptg"

[    1] Inicio
[    2]   int: num;
[    3]   Int: nUm, NuM;
[    4]   INT: Maior;
[    5]   imprima ("Digite 3 nros:");
[    6]   leia (num);
[    7]   Leia (nUm);
[    8]   LEIA (NuM);
[    9]   se (num>=nUm) e (num>=NuM) entao
[   10]     maior <- num;
[   11]   senao
[   12]   SE (nUm>=num) E (nUm>=NuM) ENTAO
[   13]     MAIOR <- nUm;
[   14]   SENAO
[   15]     Maior <- NuM;
[   16]   Fim_Se
[   17]   Imprima ("Maior = ");
[   18]   IMPRIMA (Maior);
[   19]   NuM <-    -.12 - +001 - 1234567890 + -.0 - +0. - 12345.67890 ;
[   20]   nUm<--.12-+001-1234567890+-.0-+0.-12345.67890;
[   21]   num<-+.;
         --------^
         Erro lexico na linha 21 coluna 9: Ponto isolado '.'
[   22]   para   de 1ate xpasso1 s/**++*/e  2<3
         -------------^
         Erro lexico na linha 22 coluna 14: Delimitador esperado 'a'
[   23]   paraide1 ate9passo 1.2.3.<----(***))
         --------------------------^
         Erro lexico na linha 23 coluna 27: Ponto isolado '.'
[   24]   para i de 1ate 9. 0. passo 1$j
         -------------^
         Erro lexico na linha 24 coluna 14: Delimitador esperado 'a'
[   24]   para i de 1ate 9. 0. passo 1$j
         ------------------------------^
         Erro lexico na linha 24 coluna 31: Caractere invalido '$'
[   25] FIM

TOTAL DE ERROS: 5
```

3.1.3. Tokens reconhecidos

LISTA DE TOKENS RECONHECIDOS EM "teste-01.ptg"

LIN	COL	COD	TOKEN	LEXEMA	POS TAB SIMB
1	1	6	tk_inicio		
2	3	8	tk_int		
	6	26	tk_dois_pts		
	8	2	tk_IDEN	num	60
	11	25	tk_pt_virg		
3	3	8	tk_int		
	6	26	tk_dois_pts		
	8	2	tk_IDEN	nUm	67
	11	24	tk_virg		
	13	2	tk_IDEN	NuM	149
	16	25	tk_pt_virg		
4	3	8	tk_int		
	6	26	tk_dois_pts		
	8	2	tk_IDEN	Maior	130
	13	25	tk_pt_virg		
5	3	11	tk_imprima		
	11	27	tk_abre_par		
	12	5	tk_CADEIA	"Digite 3 nros:"	203
	28	28	tk_fecha_par		
	29	25	tk_pt_virg		
6	3	10	tk_leia		
	8	27	tk_abre_par		
	9	2	tk_IDEN	| 12 | 28	tk_fecha_par
	13	25	tk_pt_virg		
7	3	10	tk_leia		
	8	27	tk_abre_par		
	9	2	tk_IDEN	| 12 | 28	tk_fecha_par
	13	25	tk_pt_virg		
8	3	10	tk_leia		
	8	27	tk_abre_par		
	9	2	tk_IDEN	| 12 | 28	tk_fecha_par
	13	25	tk_pt_virg		
9	3	17	tk_se		
	6	27	tk_abre_par		
	7	2	tk_IDEN	| 10 | 32	tk_maior_igual
	12	2	tk_IDEN	| 15 | 28	tk_fecha_par
	17	21	tk_e		
	19	27	tk_abre_par		
	20	2	tk_IDEN	| 23 | 32	tk_maior_igual
	25	2	tk_IDEN	| 28 | 28	tk_fecha_par
	30	18	tk_entao		
10	5	2	tk_IDEN	maior	144
	11	37	tk_atrib		
	14	2	tk_IDEN	| 17 | 25	tk_pt_virg
11	3	19	tk_senao		
12	3	17	tk_se		
	6	27	tk_abre_par		
	7	2	tk_IDEN	| 10 | 32	tk_maior_igual
	12	2	tk_IDEN	| 15 | 28	tk_fecha_par
	17	21	tk_e		
	19	27	tk_abre_par		
	20	2	tk_IDEN	| 23 | 32	tk_maior_igual
	25	2	tk_IDEN	| 28 | 28	tk_fecha_par
	30	18	tk_entao		

```
|  13 |   5 |   2 | tk_IDEN        | MAIOR            |       160      |                |
|     |  11 |  37 | tk_atrib       |                  |                |
|     |  14 |   2 | tk_IDEN        |     |  17 |  25 | tk_pt_virg     |                |
|
|  14 |   3 |  19 | tk_senao       |                  |                |
|  15 |   5 |   2 | tk_IDEN        |     |  11 |  37 | tk_atrib       |                |
|
|     |  14 |   2 | tk_IDEN        |     |  17 |  25 | tk_pt_virg     |                |
|
|  16 |   3 |  20 | tk_fim_se      |                  |                |
|  17 |   3 |  11 | tk_imprima     |                  |                |
|     |  11 |  27 | tk_abre_par    |                  |                |
|     |  12 |   5 | tk_CADEIA      | "Maior = "       |        73      |
|     |  22 |  28 | tk_fecha_par   |                  |                |
|     |  23 |  25 | tk_pt_virg     |                  |                |
|  18 |   3 |  11 | tk_imprima     |                  |                |
|     |  11 |  27 | tk_abre_par    |                  |                |
|     |  12 |   2 | tk_IDEN        |     |  17 |  28 | tk_fecha_par   |                |
|
|     |  18 |  25 | tk_pt_virg     |                  |                |
|  19 |   3 |   2 | tk_IDEN        |     |   7 |  37 | tk_atrib       |                |
|
|     |  13 |  39 | tk_menos       |                  |                |
|     |  14 |   4 | tk_DECIMAL     | .12              |        63      |
|     |  18 |  39 | tk_menos       |                  |                |
|     |  20 |  38 | tk_mais        |                  |                |
|     |  21 |   3 | tk_INTEIRO     | 001              |       125      |
|     |  25 |  39 | tk_menos       |                  |                |
|     |  27 |   3 | tk_INTEIRO     | 1234567890       |        23      |
|     |  38 |  38 | tk_mais        |                  |                |
|     |  40 |  39 | tk_menos       |                  |                |
|     |  41 |   4 | tk_DECIMAL     | .0               |        90      |
|     |  44 |  39 | tk_menos       |                  |                |
|     |  46 |  38 | tk_mais        |                  |                |
|     |  47 |   4 | tk_DECIMAL     | 0.               |       188      |
|     |  50 |  39 | tk_menos       |                  |                |
|     |  52 |   4 | tk_DECIMAL     | 12345.67890      |       123      |
|     |  64 |  25 | tk_pt_virg     |                  |                |
|  20 |   3 |   2 | tk_IDEN        |     |   6 |  37 | tk_atrib       |                |
|
|     |   8 |  39 | tk_menos       |                  |                |
|     |   9 |   4 | tk_DECIMAL     |     |  12 |  39 | tk_menos       |                |
|
|     |  13 |  38 | tk_mais        |                  |                |
|     |  14 |   3 | tk_INTEIRO     |     |  17 |  39 | tk_menos       |                |
|
|     |  18 |   3 | tk_INTEIRO     |     |  28 |  38 | tk_mais        |                |
|
|     |  29 |  39 | tk_menos       |                  |                |
|     |  30 |   4 | tk_DECIMAL     |     |  32 |  39 | tk_menos       |                |
|
|     |  33 |  38 | tk_mais        |                  |                |
|     |  34 |   4 | tk_DECIMAL     |     |  36 |  39 | tk_menos       |                |
|
|     |  37 |   4 | tk_DECIMAL     |     |  48 |  25 | tk_pt_virg     |                |
|
|  21 |   3 |   2 | tk_IDEN        |     |   6 |  37 | tk_atrib       |                |
|
|     |   8 |  38 | tk_mais        |                  |                |
|     |  10 |  25 | tk_pt_virg     |                  |                |
|  22 |   3 |  12 | tk_para        |                  |                |
|     |  10 |  13 | tk_de          |                  |                |
|     |  13 |   3 | tk_INTEIRO     | 1                |        91      |
|     |  14 |  14 | tk_ate         |                  |                |
|     |  18 |   2 | tk_IDEN        | xpasso1          |        18      |
|     |  26 |   2 | tk_IDEN        | s                |        39      |
|  23 |   3 |   2 | tk_IDEN        | paraide1         |       110      |
|     |  12 |   2 | tk_IDEN        | ate9passo        |        53      |
|     |  22 |   4 | tk_DECIMAL     | 1.2              |       211      |
|     |  25 |   4 | tk_DECIMAL     | .3               |       111      |
|     |  28 |  37 | tk_atrib       |                  |                |
|     |  30 |  36 | tk_decr        |                  |                |
|     |  32 |  39 | tk_menos       |                  |                |
|     |  38 |  28 | tk_fecha_par   |                  |                |
|  24 |   3 |  12 | tk_para        |                  |                |
```

```
|     |   8 |   2 | tk_IDEN        | i                |       226      |
|     |  10 |  13 | tk_de          |                  |                |
|     |  13 |   3 | tk_INTEIRO     |     |  14 |  14 | tk_ate         |                  |
|
|     |  18 |   4 | tk_DECIMAL     | 9.               |       178      |
|     |  21 |   4 | tk_DECIMAL     |     |  24 |  15 | tk_passo       |                  |
|
|     |  30 |   3 | tk_INTEIRO     |     |  32 |   2 | tk_IDEN        | j                |
233      |
|  25 |   1 |   7 | tk_fim         |                  |                |
|  26 |   1 |   1 | tk_EOF         |                  |                |
+-----+-----+-----+----------------+------------------+----------------+
```

RESUMO

COD	TOKEN	USOS
1	tk_EOF	1
2	tk_IDEN	31
3	tk_INTEIRO	7
4	tk_DECIMAL	12
5	tk_CADEIA	2
6	tk_inicio	1
7	tk_fim	1
8	tk_int	3
9	tk_dec	0
10	tk_leia	3
11	tk_imprima	3
12	tk_para	2
13	tk_de	2
14	tk_ate	2
15	tk_passo	1
16	tk_fim_para	0
17	tk_se	2
18	tk_entao	2
19	tk_senao	2
20	tk_fim_se	1
21	tk_e	2
22	tk_ou	0
23	tk_nao	0
24	tk_virg	1
25	tk_pt_virg	15
26	tk_dois_pts	3
27	tk_abre_par	10
28	tk_fecha_par	11
29	tk_menor	0
30	tk_menor_igual	0
31	tk_maior	0
32	tk_maior_igual	4
33	tk_diferente	0
34	tk_igual	0
35	tk_incr	0
36	tk_decr	1
37	tk_atrib	7
38	tk_mais	7
39	tk_menos	13
40	tk_vezes	0
41	tk_dividido	0
0	TOTAL	152

TOTAL DE ERROS: 5

3.1.4. Tabela de símbolos

TABELA DE SIMBOLOS - "teste-01.ptg"

POS	TOKEN	LEXEMA	POS NA ENTRADA (linha,coluna)
18	tk_IDEN	xpasso1	(22, 18)
23	tk_INTEIRO	1234567890	(19, 27) (20, 18)
39	tk_IDEN	s	(22, 26)
53	tk_IDEN	ate9passo	(23, 12)
60	tk_IDEN	num	(2, 8) (6, 9) (9, 7) (9, 20) (10, 14) (12, 12) (21, 3)
63	tk_DECIMAL	.12	(19, 14) (20, 9)
67	tk_IDEN	nUm	(3, 8) (7, 9) (9, 12) (12, 7) (12, 20) (13, 14) (20, 3)
73	tk_CADEIA	"Maior = "	(17, 12)
90	tk_DECIMAL	.0	(19, 41) (20, 30)
91	tk_INTEIRO	1	(22, 13) (24, 13) (24, 30)
110	tk_IDEN	paraide1	(23, 3)
111	tk_DECIMAL	.3	(23, 25)
123	tk_DECIMAL	12345.67890	(19, 52) (20, 37)
125	tk_INTEIRO	001	(19, 21) (20, 14)
130	tk_IDEN	Maior	(4, 8) (15, 5) (18, 12)
144	tk_IDEN	maior	(10, 5)
149	tk_IDEN	NuM	(3, 13) (8, 9) (9, 25) (12, 25) (15, 14) (19, 3)
160	tk_IDEN	MAIOR	(13, 5)
178	tk_DECIMAL	9.	(24, 18)
188	tk_DECIMAL	0.	(19, 47) (20, 34) (24, 21)
203	tk_CADEIA	"Digite 3 nros:"	(5, 12)
211	tk_DECIMAL	1.2	(23, 22)
226	tk_IDEN	i	(24, 8)
233	tk_IDEN	j	(24, 32)

3.2. Teste 2

3.2.1. Arquivo de entrada

```
// Comentário 1
// Comentário 2 //
/* Comentário 3
/* Comentário 4 */
(* Comentário 5 *)

início
  int: x, y, z;  /* Declaração de variáveis
  (* Esse tipo de dado é válido?? *) decimal: média;
  imprima ("Digite um valor para x:");
  leia (x);
  imprima("Digite um valor para y:");
  leia(y);
  imprima ("Digite um valor para z:");
  leia (z);
  media <- (x+y+z)*0.33333;
  imprima ("Média = ");
  imprima   (média);
  imprima    ("\n");
fim
```

3.2.2. Erros léxicos

```
LISTA DE ERROS LEXICOS EM "teste-02.ptg"

[    1] // Comentário 1
        ---------^
        Erro lexico na linha 1 coluna 10: Caractere invalido 'á'
[    2] // Comentário 2 //
        ---------^
        Erro lexico na linha 2 coluna 10: Caractere invalido 'á'
[    3] /* Comentário 3
[    4] /* Comentário 4 */
[    5] (* Comentário 5 *)
[    6]
[    7] início
        --^
        Erro lexico na linha 7 coluna 3: Caractere invalido 'í'
[    8]   int: x, y, z;  /* Declaração de variáveis
[    9]   (* Esse tipo de dado é válido?? *) decimal: média;
        ------------------------------------------------^
        Erro lexico na linha 9 coluna 48: Caractere invalido 'é'
[   10]   imprima ("Digite um valor para x:");
[   11]   leia (x);
[   12]   imprima("Digite um valor para y:");
[   13]   leia(y);
[   14]   imprima ("Digite um valor para z:");
[   15]   leia (z);
[   16]   media <- (x+y+z)*0.33333;
[   17]   imprima ("Média = ");
[   18]   imprima   (média);
        --------------^
        Erro lexico na linha 18 coluna 15: Caractere invalido 'é'
[   19]   imprima    ("\n");
[   20] fim

TOTAL DE ERROS: 5
```

3.2.3. Tokens reconhecidos

```
LISTA DE TOKENS RECONHECIDOS EM "teste-02.ptg"

+-----+-----+-----+----------------+---------------------------+----------------+
| LIN | COL | COD | TOKEN          | LEXEMA                    |  POS TAB SIMB  |
+-----+-----+-----+----------------+---------------------------+----------------+
|   1 |   1 |  41 | tk_dividido    |                           |                |
|     |   2 |  41 | tk_dividido    |                           |                |
|     |   4 |   2 | tk_IDEN        | Coment                    |        80      |
|     |  11 |   2 | tk_IDEN        | rio                       |       166      |
|     |  15 |   3 | tk_INTEIRO     | 1                         |        91      |
|   2 |   1 |  41 | tk_dividido    |                           |                |
|     |   2 |  41 | tk_dividido    |                           |                |
|     |   4 |   2 | tk_IDEN        |     |  11 |   2 | tk_IDEN       |     |  15 |   3 |
tk_INTEIRO     | 2                         |        98      |
|     |  17 |  41 | tk_dividido    |                           |                |
|     |  18 |  41 | tk_dividido    |                           |                |
|   7 |   1 |   2 | tk_IDEN        | in                        |       230      |
|     |   4 |   2 | tk_IDEN        | cio                       |       128      |
|   8 |   3 |   8 | tk_int         |                           |                |
|     |   6 |  26 | tk_dois_pts    |                           |                |
|     |   8 |   2 | tk_IDEN        | x                         |        74      |
|     |   9 |  24 | tk_virg        |                           |                |
|     |  11 |   2 | tk_IDEN        | y                         |        81      |
|     |  12 |  24 | tk_virg        |                           |                |
|     |  14 |   2 | tk_IDEN        | z                         |        88      |
|     |  15 |  25 | tk_pt_virg     |                           |                |
|   9 |  38 |   2 | tk_IDEN        | decimal                   |         3      |
|     |  45 |  26 | tk_dois_pts    |                           |                |
|     |  47 |   2 | tk_IDEN        | m                         |       254      |
|     |  49 |   2 | tk_IDEN        | dia                       |       221      |
|     |  52 |  25 | tk_pt_virg     |                           |                |
|  10 |   3 |  11 | tk_imprima     |                           |                |
|     |  11 |  27 | tk_abre_par    |                           |                |
|     |  12 |   5 | tk_CADEIA      | "Digite um valor para x:" |       110      |
|     |  37 |  28 | tk_fecha_par   |                           |                |
|     |  38 |  25 | tk_pt_virg     |                           |                |
|  11 |   3 |  10 | tk_leia        |                           |                |
|     |   8 |  27 | tk_abre_par    |                           |                |
|     |   9 |   2 | tk_IDEN        |     |  10 |  28 | tk_fecha_par  |
|                |
|     |  11 |  25 | tk_pt_virg     |                           |                |
|  12 |   3 |  11 | tk_imprima     |                           |                |
|     |  10 |  27 | tk_abre_par    |                           |                |
|     |  11 |   5 | tk_CADEIA      | "Digite um valor para y:" |        44      |
|     |  36 |  28 | tk_fecha_par   |                           |                |
|     |  37 |  25 | tk_pt_virg     |                           |                |
|  13 |   3 |  10 | tk_leia        |                           |                |
|     |   7 |  27 | tk_abre_par    |                           |                |
|     |   8 |   2 | tk_IDEN        |     |   9 |  28 | tk_fecha_par  |
|                |
|     |  10 |  25 | tk_pt_virg     |                           |                |
|  14 |   3 |  11 | tk_imprima     |                           |                |
|     |  11 |  27 | tk_abre_par    |                           |                |
|     |  12 |   5 | tk_CADEIA      | "Digite um valor para z:" |       235      |
|     |  37 |  28 | tk_fecha_par   |                           |                |
|     |  38 |  25 | tk_pt_virg     |                           |                |
|  15 |   3 |  10 | tk_leia        |                           |                |
|     |   8 |  27 | tk_abre_par    |                           |                |
|     |   9 |   2 | tk_IDEN        |     |  10 |  28 | tk_fecha_par  |
|                |
|     |  11 |  25 | tk_pt_virg     |                           |                |
|  16 |   3 |   2 | tk_IDEN        | media                     |       220      |
|     |   9 |  37 | tk_atrib       |                           |                |
|     |  12 |  27 | tk_abre_par    |                           |                |
|     |  13 |   2 | tk_IDEN        |     |  14 |  38 | tk_mais       |
|                |
|     |  15 |   2 | tk_IDEN        |     |  16 |  38 | tk_mais       |
|                |
|     |  17 |   2 | tk_IDEN        |     |  18 |  28 | tk_fecha_par  |
|                |
|     |  19 |  40 | tk_vezes       |                           |                |
```

```
|     |  20 |   4 | tk_DECIMAL     | 0.33333                  |        71      |
|     |  27 |  25 | tk_pt_virg     |                          |                |
|  17 |   3 |  11 | tk_imprima     |                          |                |
|     |  11 |  27 | tk_abre_par    |                          |                |
|     |  12 |   5 | tk_CADEIA      | "Média = "               |       122      |
|     |  22 |  28 | tk_fecha_par   |                          |                |
|     |  23 |  25 | tk_pt_virg     |                          |                |
|  18 |   3 |  11 | tk_imprima     |                          |                |
|     |  13 |  27 | tk_abre_par    |                          |                |
|     |  14 |   2 | tk_IDEN        |     |  16 |   2 | tk_IDEN        |     |  19 |  28 |
tk_fecha_par   |                                |                |
|     |  20 |  25 | tk_pt_virg     |                          |                |
|  19 |   3 |  11 | tk_imprima     |                          |                |
|     |  14 |  27 | tk_abre_par    |                          |                |
|     |  15 |   5 | tk_CADEIA      | "\n"                     |       112      |
|     |  19 |  28 | tk_fecha_par   |                          |                |
|     |  20 |  25 | tk_pt_virg     |                          |                |
|  20 |   1 |   7 | tk_fim         |                          |                |
|  21 |   1 |   1 | tk_EOF         |                          |                |
+-----+-----+-----+----------------+--------------------------+----------------+
```

RESUMO

COD	TOKEN	USOS
1	tk_EOF	1
2	tk_IDEN	21
3	tk_INTEIRO	2
4	tk_DECIMAL	1
5	tk_CADEIA	5
6	tk_inicio	0
7	tk_fim	1
8	tk_int	1
9	tk_dec	0
10	tk_leia	3
11	tk_imprima	6
12	tk_para	0
13	tk_de	0
14	tk_ate	0
15	tk_passo	0
16	tk_fim_para	0
17	tk_se	0
18	tk_entao	0
19	tk_senao	0
20	tk_fim_se	0
21	tk_e	0
22	tk_ou	0
23	tk_nao	0
24	tk_virg	2
25	tk_pt_virg	12
26	tk_dois_pts	2
27	tk_abre_par	10
28	tk_fecha_par	10
29	tk_menor	0
30	tk_menor_igual	0
31	tk_maior	0
32	tk_maior_igual	0
33	tk_diferente	0
34	tk_igual	0
35	tk_incr	0
36	tk_decr	0
37	tk_atrib	1
38	tk_mais	2
39	tk_menos	0
40	tk_vezes	1
41	tk_dividido	6
0	TOTAL	87

TOTAL DE ERROS: 5

3.2.4. Tabela de símbolos

TABELA DE SIMBOLOS - "teste-02.ptg"

POS	TOKEN	LEXEMA	POS NA ENTRADA (linha,coluna)
3	tk_IDEN	decimal	(9, 38)
44	tk_CADEIA	"Digite um valor para y:"	(12, 11)
71	tk_DECIMAL	0.33333	(16, 20)
74	tk_IDEN	x	(8, 8) (11, 9) (16, 13)
80	tk_IDEN	Coment	(1, 4) (2, 4)
81	tk_IDEN	y	(8, 11) (13, 8) (16, 15)
88	tk_IDEN	z	(8, 14) (15, 9) (16, 17)
91	tk_INTEIRO	1	(1, 15)
98	tk_INTEIRO	2	(2, 15)
110	tk_CADEIA	"Digite um valor para x:"	(10, 12)
112	tk_CADEIA	"\n"	(19, 15)
122	tk_CADEIA	"Média = "	(17, 12)
128	tk_IDEN	cio	(7, 4)
166	tk_IDEN	rio	(1, 11) (2, 11)
220	tk_IDEN	media	(16, 3)
221	tk_IDEN	dia	(9, 49) (18, 16)
230	tk_IDEN	in	(7, 1)
235	tk_CADEIA	"Digite um valor para z:"	(14, 12)
254	tk_IDEN	m	(9, 47) (18, 14)

3.3. Teste 3

3.3.1. Arquivo de entrada

```
(* teste-03.ptg *))

inicio
  int: i, n, fat;
  imprima ("Digite um nro: ");
  leia (n);
  para i de 1.0 ate n passo (2*.5)    /* Cálculo
    fat = fat*i;                      /* do
  fim_para                            /* fatorial
  imprima ("Fatorial = ");
  imprima (fat);
fim
```

3.3.2. Erros léxicos

```
LISTA DE ERROS LEXICOS EM "teste-03.ptg"

[    1] (* teste-03.ptg *))
[    2]
[    3] inicio
[    4]   int: i, n, fat;
[    5]   imprima ("Digite um nro: ");
[    6]   leia (n);
[    7]   para i de 1.0 ate n passo (2*.5)    /* Cálculo
[    8]     fat = fat*i;                       /* do
[    9]   fim_para                            /* fatorial
[   10]   imprima ("Fatorial = ");
[   11]   imprima (fat);
[   12] fim

TOTAL DE ERROS: 0
```

3.3.3. Tokens reconhecidos

LISTA DE TOKENS RECONHECIDOS EM "teste-03.ptg"

LIN	COL	COD	TOKEN	LEXEMA	POS TAB SIMB
1	19	28	tk_fecha_par		
3	1	6	tk_inicio		
4	3	8	tk_int		
	6	26	tk_dois_pts		
	8	2	tk_IDEN	i	226
	9	24	tk_virg		
	11	2	tk_IDEN	n	4
	12	24	tk_virg		
	14	2	tk_IDEN	fat	31
	17	25	tk_pt_virg		
5	3	11	tk_imprima		
	11	27	tk_abre_par		
	12	5	tk_CADEIA	"Digite um nro: "	86
	29	28	tk_fecha_par		
	30	25	tk_pt_virg		
6	3	10	tk_leia		
	8	27	tk_abre_par		
	9	2	tk_IDEN	| 10 | 28 | tk_fecha_par	
	11	25	tk_pt_virg		
7	3	12	tk_para		
	8	2	tk_IDEN	| 10 | 13 | tk_de	
	13	4	tk_DECIMAL	1.0	197
	17	14	tk_ate		
	21	2	tk_IDEN	| 23 | 15 | tk_passo	
	29	27	tk_abre_par		
	30	3	tk_INTEIRO	2	98
	31	40	tk_vezes		
	32	4	tk_DECIMAL	.5	125
	34	28	tk_fecha_par		
8	5	2	tk_IDEN	| 9 | 34 | tk_igual	
	11	2	tk_IDEN	| 14 | 40 | tk_vezes	
	15	2	tk_IDEN	| 16 | 25 | tk_pt_virg	
9	3	16	tk_fim_para		
10	3	11	tk_imprima		
	11	27	tk_abre_par		
	12	5	tk_CADEIA	"Fatorial = "	233
	25	28	tk_fecha_par		
	26	25	tk_pt_virg		
11	3	11	tk_imprima		
	11	27	tk_abre_par		
	12	2	tk_IDEN	| 15 | 28 | tk_fecha_par	
	16	25	tk_pt_virg		
12	1	7	tk_fim		
	4	1	tk_EOF		

```
RESUMO

+-----+----------------+------+
| COD | TOKEN          | USOS |
+-----+----------------+------+
|   1 | tk_EOF         |    1 |
|   2 | tk_IDEN        |   10 |
|   3 | tk_INTEIRO     |    1 |
|   4 | tk_DECIMAL     |    2 |
|   5 | tk_CADEIA      |    2 |
|   6 | tk_inicio      |    1 |
|   7 | tk_fim         |    1 |
|   8 | tk_int         |    1 |
|   9 | tk_dec         |    0 |
|  10 | tk_leia        |    1 |
|  11 | tk_imprima     |    3 |
|  12 | tk_para        |    1 |
|  13 | tk_de          |    1 |
|  14 | tk_ate         |    1 |
|  15 | tk_passo       |    1 |
|  16 | tk_fim_para    |    1 |
|  17 | tk_se          |    0 |
|  18 | tk_entao       |    0 |
|  19 | tk_senao       |    0 |
|  20 | tk_fim_se      |    0 |
|  21 | tk_e           |    0 |
|  22 | tk_ou          |    0 |
|  23 | tk_nao         |    0 |
|  24 | tk_virg        |    2 |
|  25 | tk_pt_virg     |    6 |
|  26 | tk_dois_pts    |    1 |
|  27 | tk_abre_par    |    5 |
|  28 | tk_fecha_par   |    6 |
|  29 | tk_menor       |    0 |
|  30 | tk_menor_igual |    0 |
|  31 | tk_maior       |    0 |
|  32 | tk_maior_igual |    0 |
|  33 | tk_diferente   |    0 |
|  34 | tk_igual       |    1 |
|  35 | tk_incr        |    0 |
|  36 | tk_decr        |    0 |
|  37 | tk_atrib       |    0 |
|  38 | tk_mais        |    0 |
|  39 | tk_menos       |    0 |
|  40 | tk_vezes       |    2 |
|  41 | tk_dividido    |    0 |
+-----+----------------+------+
|   0 | TOTAL          |   51 |
+-----+----------------+------+

TOTAL DE ERROS: 0
```

3.3.4. Tabela de símbolos

TABELA DE SIMBOLOS - "teste-03.ptg"

POS	TOKEN	LEXEMA	POS NA ENTRADA (linha,coluna)
4	tk_IDEN	n	(4, 11) (6, 9) (7, 21)
31	tk_IDEN	fat	(4, 14) (8, 5) (8, 11) (11, 12)
86	tk_CADEIA	"Digite um nro: "	(5, 12)
98	tk_INTEIRO	2	(7, 30)
125	tk_DECIMAL	.5	(7, 32)
197	tk_DECIMAL	1.0	(7, 13)
226	tk_IDEN	i	(4, 8) (7, 8) (8, 15)
233	tk_CADEIA	"Fatorial = "	(10, 12)

4. Formulário de pré-avaliação

(imprima-o em branco e preencha a mão, de lápis)

Resumo	
Data da defesa	Auto-avaliação (nota)
Pontos fortes	**Pontos fracos**

Objetos de avaliação	**Evidência**		**Comentários** *(apenas se necessário)*
	Página	Linha	
Nome do arquivo de entrada			
[A01] ( ) Fixo no código			
[A02] ( ) Fornecido na linha de comando na chamada do programa			
[A03] ( ) Fornecido na linha de comando após chamada do programa			
[A04] ( ) Fornecido via interface gráfica			
[A05] ( ) Outro:			
Maiúsculas, minúsculas e acentos			
[B01] ( ) Diferenciadas em palavras reservadas (`ate` ≠ `Ate`)			
[B02] ( ) Diferenciadas em identificadores (media ≠ Media)			
[B03] ( ) Acentos rejeitados em palavras reservadas (`até`)			
[B04] ( ) Acentos rejeitados em identificadores (média)			
Detalhes de implementação			
[C01] ( ) Comentários de linha tratados corretamente			
[C02] ( ) Comentários de bloco tratados corretamente			
[C03] ( ) Retorna *token* para sinalizar um comentário			
[C04] ( ) Retorna *token* para sinalizar um erro léxico			
[C05] ( ) Retorna *token* para sinalizar fim de arquivo (`tkEOF`)			
Arrays, tabelas etc de tamanho arbitrário *(Forneça breve descrição)*			
[D01]			
[D02]			
[D03]			
[D04]			
***Tokens*: declaração**			
[E01] ( ) enum [E04] ( ) const int			
[E02] ( ) #define [E05] ( ) string ou vetor de caracteres			
[E03] ( ) int [E06] ( ) Outro:			
***Tokens*: o que é fornecido ao parser**			
[E07] ( ) Código numérico do *token*			
[E08] ( ) Cadeia do nome do *token*			
[E09] ( ) Posição do lexema na tabela de símbolos			
[E10] ( ) Cadeia do lexema			
[E11] ( ) Outros:			

Tokens: quando e como são fornecidos ao parser			
[E12] ( ) Um por chamada via comando `return`			
[E13] ( ) Um por chamada via variável global, p.ex. `int` ou `array`			
[E14] ( ) Todos de uma vez ao final da análise léxica			
[E15] ( ) Outro:			
Transições			
[F01] ( ) Guiadas por comandos `if` / `switch`			
[F02] ( ) Guiadas por tabela de transições			
[F03] ( ) Tabela de transições implementada *Descreva a estrutura – lista encadeada, array, tabela hash?*			
[F04] ( ) Tabela nunca é consultada			
[F05] ( ) Tabela consultada uma única vez no laço			
[F06] ( ) Tabela consultada várias vezes no laço			
[F07] ( ) Tabela impressa com diagrama do autômato			
[F08] ( ) Tabela congruente com diagrama do autômato			
[F09] ( ) Tabela única para todos os testes			
Tabela de símbolos: implementação			
[G01] ( ) Em nenhuma estrutura de dados			
[G02] ( ) Em estrutura de dados compartilhada			
[G03] ( ) Em estrutura de dados própria e exclusiva *Descreva a estrutura – lista encadeada, array, tabela hash?*			
[G04] ( ) Permite duplicação de lexemas			
Tabela de símbolos: palavras reservadas			
[G05] ( ) *Tokens* não armazenados			
[G06] ( ) *Tokens*: códigos numéricos armazenados – *Como?*			
[G07] ( ) *Tokens*: nomes armazenados – *Como?*			
[G08] ( ) *Tokens*: armazenados de outra forma – *Como?*			
[G09] ( ) Lexemas não armazenados			
[G10] ( ) Lexemas: armazenados – *Como?*			
Tabela de símbolos: operadores			
[G11] ( ) *Tokens* não armazenados			
[G12] ( ) *Tokens*: códigos numéricos armazenados – *Como?*			
[G13] ( ) *Tokens*: nomes armazenados – *Como?*			
[G14] ( ) *Tokens*: armazenados de outra forma – *Como?*			
[G15] ( ) Lexemas não armazenados			
[G16] ( ) Lexemas: armazenados – *Como?*			
Tabela de símbolos: delimitadores			
[G17] ( ) *Tokens* não armazenados			
[G18] ( ) *Tokens*: códigos numéricos armazenados – *Como?*			
[G19] ( ) *Tokens*: nomes armazenados – *Como?*			
[G20] ( ) *Tokens*: armazenados de outra forma – *Como?*			
[G21] ( ) Lexemas não armazenados			
[G22] ( ) Lexemas: armazenados – *Como?*			

Tabela de símbolos: identificadores			
[G23] ( ) *Tokens* não armazenados			
[G24] ( ) *Tokens*: códigos numéricos armazenados – *Como?*			
[G25] ( ) *Tokens*: nomes armazenados – *Como?*			
[G26] ( ) *Tokens*: armazenados de outra forma – *Como?*			
[G27] ( ) Lexemas não armazenados			
[G28] ( ) Lexemas: armazenados – *Como?*			
Tabela de símbolos: constantes inteiras			
[G29] ( ) *Tokens* não armazenados			
[G30] ( ) *Tokens*: códigos numéricos armazenados – *Como?*			
[G31] ( ) *Tokens*: nomes armazenados – *Como?*			
[G32] ( ) *Tokens*: armazenados de outra forma – *Como?*			
[G33] ( ) Lexemas não armazenados			
[G34] ( ) Lexemas: armazenados – *Como?*			
Tabela de símbolos: constantes decimais			
[G35] ( ) *Tokens* não armazenados			
[G36] ( ) *Tokens*: códigos numéricos armazenados – *Como?*			
[G37] ( ) *Tokens*: nomes armazenados – *Como?*			
[G38] ( ) *Tokens*: armazenados de outra forma – *Como?*			
[G39] ( ) Lexemas não armazenados			
[G40] ( ) Lexemas: armazenados – *Como?*			
Tabela de símbolos: constantes *string* (literais)			
[G41] ( ) *Tokens* não armazenados			
[G42] ( ) *Tokens*: códigos numéricos armazenados – *Como?*			
[G43] ( ) *Tokens*: nomes armazenados – *Como?*			
[G44] ( ) *Tokens*: armazenados de outra forma – *Como?*			
[G45] ( ) Lexemas não armazenados			
[G46] ( ) Lexemas: armazenados – *Como?*			
Tabela de símbolos: outro conteúdo			
[G47] ( ) Linha / coluna do *token* no arquivo de entrada			
[G48] ( )			
[G49] ( )			
[G50] ( )			