



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:

Eberty Alves da Silva
Philippe Alexandre R. Kramer

Ilhéus, BA
13/11/2017

Sumário

1. Código fonte	1
2. Autômato.....	22
2.1. Diagrama de estados	22
2.2. Ações	23
2.3. Tabela de transições.....	25
3. Resultados dos testes	26
3.1. Teste 1	26
3.1.1. Arquivo de entrada.....	26
3.1.2. Erros léxicos.....	27
3.1.3. Tokens reconhecidos.....	28
3.1.4. Tabela de símbolos.....	32
3.2. Teste 2	33
3.2.1. Arquivo de entrada.....	33
3.2.2. Erros léxicos.....	34
3.2.3. Tokens reconhecidos.....	35
3.2.4. Tabela de símbolos.....	38
3.3. Teste 3	39
3.3.1. Arquivo de entrada.....	39
3.3.2. Erros léxicos.....	40
3.3.3. Tokens reconhecidos.....	41
3.3.4. Tabela de símbolos.....	43
4. Formulário de pré-avaliação.....	44

1. Código fonte

```
1  /* ----- */
2  /*                      ARQUIVO: Portugol.c                      */
3  /* ----- */
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <ctype.h>
7  #include <string.h>
8
9  #include "lexema.h"
10 #include "tokens.h"
11 #include "automato.h"
12 #include "erros.h"
13 #include "resultados.h"
14
15
16 ///FUNCAO PRINCIPAL
17 int main (int argc, char *argv[]){
18     tToken token_da_vez;
19
20     if (argc < 2){
21         printf("Exemplo de execucao: ./Portugol prog01.ptg\nTente novamente\n");
22     } else {
23         iniciar_Lexema();
24         // Roda os arquivos que sao digitados na entrada
25         for (int i=1; i < argc; i++){
26             printf("\nArquivo: %s \n", argv[i]);
27
28             if ((arquivo_de_entrada = fopen(argv[i], "r")) == NULL){
29                 printf("Erro ao abrir o arquivo!!! \nPor favor, verifique a existen-
30 cia do mesmo e tente novamente.\n");
31             } else {
32                 //Inicia listas de tokens, erros, tabelas e arquivo
33                 iniciar_Lista_De_Erros();
34                 iniciar_Lista_De_Tokens();
35                 iniciar_Tabela_de_Simbolos();
36                 iniciar_Ordem_Tab();
37                 linha_arquivo = 1, coluna_arquivo = 1;
38
39                 //Recebe tokens
40                 do {
41                     token_da_vez = analizador_Lexico();
42                     adicionar_Token_Na_Lista_De_Tokens(token_da_vez, li-
43 nha_token, coluna_token);
44                 } while(token_da_vez != tk_EOF);
45
46                 //Imprime nos arquivos
47                 imprimir_Lista_De_Erros_Lexicos(argv[i]);
48                 imprimir_Lista_De_Tokens_Reconhecidos_E_Resumo(argv[i], lis-
49 ta_de_erros.tamanho_lista);
50                 imprimir_Tabela_De_Simbolos(argv[i]);
51
52                 printf("Os seguintes arquivos gerados:\n");
53                 printf("    %s.err com o conteúdo do arquivo de entrada e os erros
54 léxicos devidamente marcados\n", argv[1]);
55                 printf("    %s.tbl com a lista de tokens reconhecidos\n", argv[1]);
56                 printf("    %s.tok com o conteúdo da tabela de símbolos após proces-
57 samento.\n\n", argv[1]);
58
59                 //Libera Memoria
60                 free(lista_de_erros.id_erro);
61                 free(lista_de_tokens.id_token);
62                 free(ordem_de_entrada.ordem_de_entrada_da_tab_simbolos);
63                 liberar_Tabela_Simbolos();
64
65                 //Fecha Arquivo de entrada
66                 fclose(arquivo_de_entrada);
67             }
68         }
69         //Libera Memoria para tString lexema
70         free(lexema.string);
71     }
72     return 0;
```

```
73 }
74
75
76
77
78
79 /* ----- */
80 /*          ARQUIVO: lexema.h          */
81 /*          */
82 /* Certifique-se que este arquivo não é incluído mais de uma vez */
83 /* ----- */
84 #ifndef _LEXEMA_H_
85 #define _LEXEMA_H_
86
87 /* ----- */
88 /*          CONSTANTES, TIPOS E VARIÁVEIS          */
89 /* ----- */
90 #ifndef LIMITE_INICIAL_DE_ALOCACAO
91     #define LIMITE_INICIAL_DE_ALOCACAO 30
92 #endif
93
94 //Definicao de Estruturas - String
95 typedef struct{
96     int tamanho_string;
97     int limite_string;
98     char * string;
99 } tSring;
100
101 //VARIÁVEIS
102 tSring lexema;
103
104
105 /* ----- */
106 /*          PROTÓTIPOS DAS FUNÇÕES          */
107 /* ----- */
108 void iniciar_Lexema(void);
109 void realocar_Lexema(void);
110 void reiniciar_Lexema(void);
111 void inserir_Caractere_No_Lexema(const char);
112
113
114 /* ----- */
115 /*          IMPLEMENTAÇÃO DAS FUNÇÕES          */
116 /* ----- */
117 void iniciar_Lexema(void){
118     //Define tamanho do lexema como zero (vazio) e tamanho maximo permitido para a insercao
119     de caracteres, esse tamanho maximo pode ser alterado posteriormente
120     lexema.limite_string = LIMITE_INICIAL_DE_ALOCACAO;
121     lexema.tamanho_string = 0;
122     lexema.string = (char*) malloc(lexema.limite_string * sizeof(char));
123     if (lexema.string == NULL){
124         printf("Erro durante a alocação da string d lexema!!! \nInfelizmente o programa tra-
125     vou\n");
126         exit(-1);
127     }
128     lexema.string[0] = '\0';
129 }
130
131
132 void realocar_Lexema(void){
133     //Ao atingir o tamanho maximo, é necessario realocar o tamanho da string do lexema, essa
134     funcao é responsavel por isso
135     lexema.limite_string *= 2;
136     lexema.string = (char*) realloc (lexema.string, lexema.limite_string * sizeof(char));
137     if (lexema.string == NULL){
138         printf("Erro durante a realocação da string do lexema!!! \nInfelizmente o programa tra-
139     vou\n");
140         exit(-1);
141     }
142 }
143
144
145 void reiniciar_Lexema(void){
146     //Define tamanho do lexema como zero, se tornando uma string vazia
147     lexema.tamanho_string = 0;
```

```
148     lexema.string[0] = '\0';
149 }
150
151
152 void inserir_Caractere_No_Lexema(const char prox_Simb){
153     //O nome da funcao é auto-explicativa
154     if (lexema.tamanho_string == (lexema.limite_string - 2)) //Ao atingir o tamanho maximo, é
155     nescessario realocar o tamanho da string do lexema
156         realocar_Lexema();
157     lexema.string[lexema.tamanho_string] = prox_Simb;
158     lexema.tamanho_string++;
159     lexema.string[lexema.tamanho_string] = '\0';
160 }
161
162 #endif
163
164
165
166
167
168
169 /* ----- */
170 /*          ARQUIVO: automato.h          */
171 /* ----- */
172 /* Certifique-se que este arquivo não é incluído mais de uma vez */
173 /* ----- */
174 #ifndef _AUTOMATO_H_
175 #define _AUTOMATO_H_
176
177 /* ----- */
178 /* BIBLIOTECAS E INCLUDE FILES */
179 /* ----- */
180 #include "erros.h"
181
182
183 /* ----- */
184 /*          CONSTANTES, TIPOS E VARIÁVEIS          */
185 /* ----- */
186 #define TOTAL_CLASSES_CARACTERES 21
187 #define QUANTIDADE_DE_ESTADOS 45
188 #define QUANTIDADE_DE_TOKENS 41
189
190 //Definicao de Tipos (ENUM) - Classe de carcteres
191 typedef enum{
192     tc_branco,
193     tc_quebra_linha,
194     tc_letra,
195     tc_digito,
196     tc_underline,
197     tc_aspas,
198     tc_ponto,
199     tc_virgula,
200     tc_ponto_virgula,
201     tc_dois_pontos,
202     tc_abre_parenteses,
203     tc_fecha_parenteses,
204     tc_menor,
205     tc_igual,
206     tc_maior,
207     tc_mais,
208     tc_menos,
209     tc_vezes,
210     tc_dividido,
211     tc_EOF,
212     tc_outro
213 } tClasse_caractere;
214
215 //VARIÁVEIS
216 FILE *arquivo_de_entrada;
217 char tabela_Transicoes[QUANTIDADE_DE_ESTADOS][TOTAL_CLASSES_CARACTERES] = {
218     {0, 0, 1, 3, 44, 11, 9, 15, 16, 17, 18, 24, 25, 30, 31, 34, 37, 40, 41, 14, 44},
219     //Estado 0
220     {2, 2, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2}, //Estado 1
221     { 0 }, //Estado 2
222     {4, 4, 5, 3, 4, 4, 6, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4}, //Estado 3
```

```
223     { 0 }, //Estado 4
224     { 0 }, //Estado 5
225     {7, 7, 8, 6, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7}, //Estado 6
226     { 0 }, //Estado 7
227     { 0 }, //Estado 8
228     {10, 10, 10, 6, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10},
229 //Estado 9
230     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, //Estado 10
231     {11, 13, 11, 11, 11, 12, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 13, 11},
232 //Estado 11
233     { 0 }, //Estado 12
234     { 0 }, //Estado 13
235     { 0 }, //Estado 14
236     { 0 }, //Estado 15
237     { 0 }, //Estado 16
238     { 0 }, //Estado 17
239     {20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 19, 20, 20, 20},
240 //Estado 18
241     {19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 21, 19, 23, 19},
242 //Estado 19
243     { 0 }, //Estado 20
244     {19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 22, 19, 19, 19, 19, 21, 19, 23, 19},
245 //Estado 21
246     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, //Estado 22
247     {42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42},
248 //Estado 23
249     { 0 }, //Estado 24
250     {29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 28, 27, 29, 26, 29, 29, 29, 29},
251 //Estado 25
252     { 0 }, //Estado 26
253     { 0 }, //Estado 27
254     { 0 }, //Estado 28
255     { 0 }, //Estado 29
256     { 0 }, //Estado 30
257     {33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 32, 33, 33, 33, 33, 33, 33},
258 //Estado 31
259     { 0 }, //Estado 32
260     { 0 }, //Estado 33
261     {36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 35, 36, 36, 36, 36},
262 //Estado 34
263     { 0 }, //Estado 35
264     { 0 }, //Estado 36
265     {39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 38, 39, 39, 39},
266 //Estado 37
267     { 0 }, //Estado 38
268     { 0 }, //Estado 39
269     { 0 }, //Estado 40
270     {43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 42, 43, 43, 43},
271 //Estado 41
272     {42, 0, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 14, 42},
273 //Estado 42
274     { 0 }, //Estado 43
275     { 0 }, //Estado 44
276 };
277 int linha_arquivo = 1, coluna_arquivo = 1;
278
279
280 /* ----- */
281 /*     PROTÓTIPOS DAS FUNÇÕES     */
282 /* ----- */
283 tToken analizador_Lexico(void);
284 void mostrar_Tabela_Transicoes (void);
285 char ler_Proximo_Caractere(void);
286 void retroceder_Caracteres(const int, const char);
287 tClasse_caractere caractere_2_tClasse_caractere(const char);
288 void retroceder_Ate(const int, const int, const int);
289
290
291 /* ----- */
292 /*     IMPLEMENTAÇÃO DAS FUNÇÕES     */
293 /* ----- */
294 tToken analizador_Lexico(void){
295     int estado = 0, id_token, contador_de_bloco = 0;
296     char prox_Simb = '-';
297 }
```

```
298     reiniciar_Lexema();
299     while (1){
300         switch (estado){
301             case 0: ///Estado Inicial
302                 linha_token = linha_arquivo;
303                 coluna_token = coluna_arquivo;
304                 break;
305
306             case 1: ///Estado identificador ou palavra reservada
307                 inserir_Caractere_No_Lexema(prox_Simb);
308                 break;
309
310             case 2: /// Estado de verificacao Palavra reservada ou identificador (FI-
311 NAL)
312                 retroceder_Caracteres(1, prox_Simb);
313                 id_token = identificar_Token();
314                 if (id_token == -1){ ///Identificador
315                     adicionar_Na_Tabela_De_Simbolos(tk_IDEN);
316                     return (tk_IDEN);
317                 } else { ///Palavra reservada
318                     return (id_token);
319                 }
320                 break;
321
322             case 3: /// Estado Digito (inteiro ou decimal)
323                 inserir_Caractere_No_Lexema(prox_Simb);
324                 break;
325
326             case 4: /// Estado Digito Inteiro (FINAL)
327                 retroceder_Caracteres(1, prox_Simb);
328                 adicionar_Na_Tabela_De_Simbolos(tk_INTEIRO);
329                 return (tk_INTEIRO);
330                 break;
331
332             case 5: ///Estado de erro lexico apos inteiro (FINAL)
333                 retroceder_Caracteres(1, prox_Simb);
334                 adicionar_Erro_Na_Lista_De_Erros(er_delimitador_esperado, prox_Simb,
335 linha_arquivo, coluna_arquivo);
336                 adicionar_Na_Tabela_De_Simbolos(tk_INTEIRO);
337                 return (tk_INTEIRO);
338                 break;
339
340             case 6: ///Estado Digito Decimal apos o ponto
341                 inserir_Caractere_No_Lexema(prox_Simb);
342                 break;
343
344             case 7: /// Estado Digito Decimal (FINAL)
345                 retroceder_Caracteres(1, prox_Simb);
346                 adicionar_Na_Tabela_De_Simbolos(tk_DECIMAL);
347                 return (tk_DECIMAL);
348                 break;
349
350             case 8: ///Estado de erro lexico apos decimal (FINAL)
351                 retroceder_Caracteres(1, prox_Simb);
352                 adicionar_Erro_Na_Lista_De_Erros(er_delimitador_esperado, prox_Simb,
353 linha_arquivo, coluna_arquivo);
354                 adicionar_Na_Tabela_De_Simbolos(tk_DECIMAL);
355                 return (tk_DECIMAL);
356                 break;
357
358             case 9: /// Estado Digito Decimal Iniciando com ponto
359                 inserir_Caractere_No_Lexema(prox_Simb);
360                 break;
361
362             case 10: ///Estado de erro lexico Ponto Isolado (FINAL)
363                 reiniciar_Lexema();
364                 retroceder_Caracteres(2, prox_Simb); ///Para ser lido novamente e
365 mudar para o estado 0
366                 adicionar_Erro_Na_Lista_De_Erros(er_ponto_isolado, prox_Simb, li-
367 nha_arquivo, coluna_arquivo);
368                 break;
369
370             case 11: ///Estado Cadeia
371                 inserir_Caractere_No_Lexema(prox_Simb);
372                 break;
```

```
373
374         case 12: ///Estado Cadeia (FINAL)
375             inserir_Caractere_No_Lexema(prox_Simb);
376             adicionar_Na_Tabela_De_Simbolos(tk_CADEIA);
377             return (tk_CADEIA);
378             break;
379
380         case 13: ///Estado de erro lexico nao fechameno da cadeia (FINAL)
381             retroceder_Caracteres(1, prox_Simb);
382             adicionar_Erro_Na_Lista_De_Erros(er_cadeia_nao_fechada, prox_Simb,
383 linha_token, coluna_token);
384             adicionar_Na_Tabela_De_Simbolos(tk_CADEIA);
385             return (tk_CADEIA);
386             break;
387
388         case 14: ///Estado End of File (FINAL)
389             return (tk_EOF);
390             break;
391
392         case 15: ///Estado Virgula (FINAL)
393             return (tk_virg);
394             break;
395
396         case 16: ///Estado Ponto-e-virgula (FINAL)
397             return (tk_pt_virg);
398             break;
399
400         case 17: ///Estado Dois-Pontos (FINAL)
401             return (tk_dois_pts);
402             break;
403
404         case 18: ///Estado Abre Parenteses
405             contador_de_bloco = 0;
406             break;
407
408         case 19: ///Comentario de bloco
409             contador_de_bloco++;
410             break;
411
412         case 20: ///Estado Abre Parenteses (FINAL)
413             retroceder_Caracteres(1, prox_Simb);
414             return (tk_abre_par);
415             break;
416
417         case 21: ///Possivel fim de comentario de bloco
418             contador_de_bloco++;
419             break;
420
421         case 22: ///Fim do comentario de bloco
422             retroceder_Caracteres(1, prox_Simb); //Para ser lido novamente e
423 mudar para o estado 0
424             break;
425
426         case 23: ///Estado de erro lexico Comentario de bloco não fechado
427             retroceder_Ate(contador_de_bloco, linha_token, coluna_token);
428 //Muitos para serem lidos novamenteem outro estado
429             adicionar_Erro_Na_Lista_De_Erros(er_comentario_de_bloco_nao_fechado,
430 prox_Simb, linha_arquivo, coluna_arquivo);
431             break;
432
433         case 24: ///Estado Fecha Parenteses (FINAL)
434             return (tk_fecha_par);
435             break;
436
437         case 25: ///Estado <
438             break;
439
440         case 26: ///Estado Atribuicao (FINAL)
441             return (tk_atrib);
442             break;
443
444         case 27: ///Estado Diferente (FINAL)
445             return (tk_diferente);
446             break;
447
```



```
448         case 28: ///Estado Menor Igual (FINAL)
449             return (tk_menor_igual);
450             break;
451
452         case 29: ///Estado Menor (FINAL)
453             retroceder_Caracteres(1, prox_Simb);
454             return (tk_menor);
455             break;
456
457         case 30: ///Estado Igual (FINAL)
458             return (tk_igual);
459             break;
460
461         case 31: ///Estado >
462             break;
463
464         case 32: ///Estado Maior Igual (FINAL)
465             return (tk_maior_igual);
466             break;
467
468         case 33: ///Estado Maior (FINAL)
469             retroceder_Caracteres(1, prox_Simb);
470             return (tk_maior);
471             break;
472
473         case 34: ///Estado +
474             break;
475
476         case 35: ///Estado Incremento (FINAL)
477             return (tk_incr);
478             break;
479
480         case 36: ///Estado Mais (FINAL)
481             retroceder_Caracteres(1, prox_Simb);
482             return (tk_mais);
483             break;
484
485         case 37: ///Estado -
486             break;
487
488         case 38: ///Estado Decremento (FINAL)
489             return (tk_decr);
490             break;
491
492         case 39: ///Estado Menos (FINAL)
493             retroceder_Caracteres(1, prox_Simb);
494             return (tk_menos);
495             break;
496
497         case 40: ///Estado Vezes (FINAL)
498             return (tk_vezes);
499             break;
500
501         case 41: ///Estado /
502             break;
503
504         case 42: ///Estado Comentario de linha
505             break;
506
507         case 43: ///Estado Dividido (FINAL)
508             retroceder_Caracteres(1, prox_Simb);
509             return (tk_dividido);
510             break;
511
512         default: ///Estado Default: Apresnta erro e ignora o caractere
513             retroceder_Caracteres(1, prox_Simb); //Para ser lido novamente e
514 mudar para o estado 0
515             adicionar_Erro_Na_Lista_De_Erros(er_caracter_invalido, prox_Simb,
516 linha_arquivo, coluna_arquivo);
517             break;
518     }
519     prox_Simb = ler_Proximo_Caractere();
520     estado = tabela_Transições[estado][carctere_2_tClasse_caractere(prox_Simb)];
521 }
522 return 0;
```

```
523 }
524
525
526 void mostrar_Tabela_Transicoes (void){
527     printf("      b \\n l d _ \" . , ; : ( ) < = > + - * / e ot\\n");
528     for (int i = 0; i < QUANTIDADE_DE_ESTADOS; i++){
529         printf("%3d |", i);
530         for (int j = 0; j < TOTAL_CLASSES_CARACTERES; j++){
531             printf ("%3d", tabela_Transicoes[i][j]);
532         }
533         printf("\\n");
534     }
535 }
536
537
538 char ler_Proximo_Caractere(void){
539     char prox_Simb = getc(arquivo_de_entrada); //Obtem caractere do arquivo
540     coluna_arquivo++; //Nova linha
541     if(prox_Simb == '\\n'){ // Se o caracter for uma quebra de linha: Novos valores para linha
542         e coluna
543             linha_arquivo++;
544             coluna_arquivo = 1;
545     }
546     return prox_Simb;
547 }
548
549
550 void retroceder_Caracteres(const int n, const char prox_Simb){
551     fseek(arquivo_de_entrada, -n*sizeof(char), SEEK_CUR); //Retocede n caracteres
552     coluna_arquivo -= n; //Diminui o numero de caracteres da coluna
553     if (prox_Simb == '\\n') //Isso nao influencia na coluna, uma vez que o prox a ser lido
554     será '\\n, o valor da coluna é zerado
555         linha_arquivo--;
556     if (prox_Simb == EOF) //Tratamento de EOF (que nao é considerado no fseek)
557         fseek(arquivo_de_entrada, 1, SEEK_CUR);
558 }
559
560
561 tClasse_caractere carctere_2 tClasse_caractere(const char prox_Simb){
562     //Funcao Responsavel pr converter o carctere lido em uma classe de simbolos (indice) da
563     tabela de transissoes
564     if (isalpha(prox_Simb)) return tc_letra;
565     if (isdigit(prox_Simb)) return tc_digito;
566     if (prox_Simb == '\\n') return tc_quebra_linha;
567     if (isspace(prox_Simb)) return tc_branco;
568     switch (prox_Simb){
569         case '_': return tc_underline;
570         case '\\': return tc_aspas;
571         case '.': return tc_ponto;
572         case ',': return tc_virgula;
573         case ';': return tc_ponto_virgula;
574         case ':': return tc_dois_pontos;
575         case '(': return tc_abre_parenteses;
576         case ')': return tc_fecha_parenteses;
577         case '<': return tc_menor;
578         case '=': return tc_igual;
579         case '>': return tc_maior;
580         case '+': return tc_mais;
581         case '-': return tc_menos;
582         case '*': return tc_vezes;
583         case '/': return tc_dividido;
584         case EOF: return tc_EOF;
585         default: return tc_outro;
586     }
587 }
588
589
590 void retroceder_Ate(const int n, const int linha, const int coluna){
591     //Similar a funcao retroceder_Caracteres, mas lida com um alcance muito maior para o va-
592     lor de n
593     fseek(arquivo_de_entrada, (-1)*n*sizeof(char), SEEK_CUR);
594     linha_arquivo = linha;
595     coluna_arquivo = coluna;
596 }
597
```

```
598 #endif
599
600
601
602
603
604
605 /* ----- */
606 /*          ARQUIVO: tokens.h          */
607 /* ----- */
608 /* Certifique-se que este arquivo não é incluído mais de uma vez */
609 /* ----- */
610
611 #ifndef _TOKENS_H_
612 #define _TOKENS_H_
613
614 /* ----- */
615 /* BIBLIOTECAS E INCLUDE FILES */
616 /* ----- */
617 #include "lexema.h"
618
619 /* ----- */
620 /*          CONSTANTES, TIPOS E VARIÁVEIS          */
621 /* ----- */
622 #ifndef LIMITE_INICIAL_DE_ALOCACAO
623     #define LIMITE_INICIAL_DE_ALOCACAO 30
624 #endif
625
626 #define TAM_TAB_HASH SIMBOLOS 139
627 #define hash(v) ((2*v)+3)%139 //Multiplique, Adicione e Divida (MAD)
628
629 //Definicao de Tipos (ENUM) - Tokens
630 typedef enum {
631     tk_EOF,
632     tk_IDEN,
633     tk_INTEIRO,
634     tk_DECIMAL,
635     tk_CADEIA,
636     tk_inicio,
637     tk_fim,
638     tk_int,
639     tk_dec,
640     tk_leia,
641     tk_imprima,
642     tk_para,
643     tk_de,
644     tk_ate,
645     tk_passo,
646     tk_fim_para,
647     tk_se,
648     tk_entao,
649     tk_senao,
650     tk_fim_se,
651     tk_e,
652     tk_ou,
653     tk_nao,
654     tk_virg,
655     tk_pt_virg,
656     tk_dois_pts,
657     tk_abre_par,
658     tk_fecha_par,
659     tk_menor,
660     tk_menor_igual,
661     tk_maior,
662     tk_maior_igual,
663     tk_diferente,
664     tk_igual,
665     tk_incr,
666     tk_decr,
667     tk_atrib,
668     tk_mais,
669     tk_menos,
670     tk_vezes,
671     tk_dividido
672 } tToken;
```

```
673
674 //Definicao de Estruturas - Identificador de token, Lista de tokens
675 typedef struct{
676     int LIN, COL;
677     tToken TOKEN;
678     int posicao_na_tabela_de_simbolos;
679 } tIdentificador_De_Token;
680
681 typedef struct{
682     int tamanho_lista;
683     int limite_lista;
684     tIdentificador_De_Token * id_token;
685 } tLista_de_tokens;
686
687 //Tabela Hash + Definicao dos Atributos da tabela de simbolos
688 typedef struct{
689     int LIN, COL;
690 } tPos;
691
692 typedef struct simbolo{
693     tToken COD;
694     int posicao;
695
696     char * lexema_cadeia;
697     int lexema_inteiro;
698     float lexema_decimal;
699
700     tPos *ocorrencias;
701     int tamanho_ocorrencias;
702     int limite_ocorrencias;
703
704     struct simbolo * proximo; //Colisao: endereçamento separado
705 } tSimbolo;
706
707 //Mecanismo que conecta, para cada par token-lexema, a ordem em que ele ocorre na entrada e sua
708 posição na tabela de símbolos
709 typedef struct {
710     int tab_simb_count;
711     int limite_tab_simb_count;
712     tSimbolo** ordem_de_entrada_da_tab_simbolos;
713 } tOrdem;
714
715 //VARIÁVEIS
716 tLista_de_tokens lista_de_tokens;
717 int linha_token, coluna_token;
718 tSimbolo ** tab_simbolos;
719 tOrdem ordem_de_entrada;
720
721
722 /* ----- */
723 /*     PROTÓTIPOS DAS FUNÇÕES     */
724 /* ----- */
725 //TOKENS
726 int identificar_Token(void);
727 void iniciar_Lista_De_Tokens(void);
728 void adicionar_Token_Na_Lista_De_Tokens(const tToken, const int, const int);
729 const char * obter_Nome_Do_Token(const tToken);
730 //TABELA DE SIMBOLOS
731 void iniciar_Tabela_de_Simbolos(void);
732 void liberar_Tabela_Simbolos(void);
733 void iniciar_Ordem_Tab(void);
734 int hash_Com_Shift(void);
735 void adicionar_Na_Tabela_De_Simbolos(const tToken);
736 tSimbolo * buscar_Na_Tabela_De_Simbolos(const tToken, const int);
737 void adiciona_Ocorrencia(tSimbolo *);
738
739
740 /* ----- */
741 /*     IMPLEMENTAÇÃO DAS FUNÇÕES     */
742 /* ----- */
743 int identificar_Token(void){
744     //Ao encontrar uma palavra qualquer, verifique se é palavra reservada com uma função que
745     retorna o código da palavra reservada
746     //ou -1 se o identificador não for palavra reservada.
747     //Tal função deve conhecer as palavras reservadas da linguagem.
```

```
748     const char * palavras_reservadas[] = {"inicio", "fim", "int", "dec", "leia", "imprima",
749 "para", "de", "ate",
750                                     "passo", "fim_para",
751 "se", "entao", "senao", "fim_se", "e", "ou", "nao"};
752
753     char * aux = (char *) malloc(lexema.tamanho_string * sizeof(char));
754     strcpy(aux, lexema.string); //Auxiliar, Letras maiúsculas e minúsculas são distinguidas
755 em nomes de identificadores, mas, é necessario observar o que vem a seguir:
756
757     for(unsigned int i = 0; i < strlen(aux); i++) // Torna todas as letras minusculas Letras
758 maiúsculas e minúsculas nao são distinguidas em palavras reservadas
759         aux[i] = tolower(aux[i]);
760
761     for(int i = 0; i < 18; i++) { //18 = numero de palavras_reservadas
762         if(strcmp(aux, palavras_reservadas[i]) == 0)
763             return (i + tk_inicio);
764     }
765
766     return -1;
767 }
768
769 void iniciar_Lista_De_Tokens(void){
770     //Define tamanho da lista de tokens reconhecidos e suas ocorrencias como zero (vazio) e
771 tamanho maximo permitido para a insercao, esse tamanho maximo pode ser alterado posteriormente
772     lista_de_tokens.tamanho_lista = 0;
773     lista_de_tokens.limite_lista = LIMITE_INICIAL_DE_ALOCACAO;
774     lista_de_tokens.id_token = (tIdentificador_De_Token *) malloc (lis-
775 ta_de_tokens.limite_lista * sizeof(tIdentificador_De_Token));
776     if (lista_de_tokens.id_token == NULL){
777         printf("Erro durante a alocao da lista de tokens!!! \nInfelizmente o programa tra-
778 vou\n");
779         exit(-1);
780     }
781 }
782
783 void adicionar_Token_Na_Lista_De_Tokens(const tToken token, const int linha, const int coluna){
784     //Adicionar token
785     lista_de_tokens.id_token[lista_de_tokens.tamanho_lista].LIN = linha;
786     lista_de_tokens.id_token[lista_de_tokens.tamanho_lista].COL = coluna;
787     if (token == tk_EOF)
788         lista_de_tokens.id_token[lista_de_tokens.tamanho_lista].COL--;
789     lista_de_tokens.id_token[lista_de_tokens.tamanho_lista].TOKEN = token;
790
791     //Posicao na tabela de simbolos
792     if(token == tk_INTEIRO || token == tk_DECIMAL || token == tk_CADEIA || token == tk_IDEN)
793         lis-
794 ta_de_tokens.id_token[lista_de_tokens.tamanho_lista].posisao_na_tabela_de_simbolos =
795 hash_Com_Shift();
796     else
797         lis-
798 ta_de_tokens.id_token[lista_de_tokens.tamanho_lista].posisao_na_tabela_de_simbolos = -1; //Nao
799 existe
800
801     //Novo tamanho
802     lista_de_tokens.tamanho_lista++;
803
804     //Verificar tamanho da alocao
805     if (lista_de_tokens.tamanho_lista == lista_de_tokens.limite_lista-1){
806         lista_de_tokens.limite_lista *= 2;
807         lista_de_tokens.id_token = (tIdentificador_De_Token *) re-
808 alloc(lista_de_tokens.id_token, lista_de_tokens.limite_lista * size-
809 of(tIdentificador_De_Token));
810         if (lista_de_tokens.id_token == NULL){
811             printf("Erro durante a realocacao da lista de tokens!!! \nInfelizmente o
812 programa travou\n");
813             exit(-1);
814         }
815     }
816 }
817
818 const char * obter_Nome_Do_Token(const tToken id_token){
819     const char * NOMES[] = {
```

```
823         "tk_EOF",
824         "tk_IDEN",
825         "tk_INTEIRO",
826         "tk_DECIMAL",
827         "tk_CADEIA",
828         "tk_inicio",
829         "tk_fim",
830         "tk_int",
831         "tk_dec",
832         "tk_leia",
833         "tk_imprima",
834         "tk_para",
835         "tk_de",
836         "tk_ate",
837         "tk_passo",
838         "tk_fim_para",
839         "tk_se",
840         "tk_entao",
841         "tk_senao",
842         "tk_fim_se",
843         "tk_e",
844         "tk_ou",
845         "tk_nao",
846         "tk_virg",
847         "tk_pt_virg",
848         "tk_dois_pts",
849         "tk_abre_par",
850         "tk_fecha_par",
851         "tk_menor",
852         "tk_menor_igual",
853         "tk_maior",
854         "tk_maior_igual",
855         "tk_diferente",
856         "tk_igual",
857         "tk_incr",
858         "tk_decr",
859         "tk_atrib",
860         "tk_mais",
861         "tk_menos",
862         "tk_vezes",
863         "tk_dividido"
864     };
865     return (NOMES[id_token]);
866 }
867
868 void iniciar_Tabela_de_Simbolos(void){
869     //inicia a tabela de simbolos com tamanho máximo do numero primo definido
870     tab_simbolos = (tSimbolo **) malloc(TAM_TAB_HASH_SIMBOLOS * sizeof(tSimbolo *));
871     if (tab_simbolos == NULL) {
872         printf("Erro durante a alocação da tabela de simbolos!!! \nInfelizmente o programa
873 travou\n");
874         exit(-1);
875     }
876
877     //Inicia todas as posições desocupadas
878     for (int i = 0; i < TAM_TAB_HASH_SIMBOLOS; i++)
879         tab_simbolos[i] = NULL;
880 }
881
882 void liberar_Tabela_Simbolos(void){
883     //Função para liberar memória alocada dos itens da tabela de simbolo (incluindo o vetor
884 de ocorrencias)
885     for(int i=0; i < TAM_TAB_HASH_SIMBOLOS; i++){
886         tSimbolo *ant, *atual;
887         ant = atual = tab_simbolos[i];
888         while (atual != NULL){
889             atual = atual->proximo;
890             free(ant->ocorrencias);
891             free(ant);
892             ant = atual;
893         }
894     }
895     free(tab_simbolos);
896 }
```

```
898 }
899
900 void iniciar_Ordem_Tab(void){
901     //Inicia o Mecanismo que conecta, para cada par token-lexema, a ordem em que ele ocorre
902     na entrada e sua posição na tabela de símbolos
903     //definindo o tamanho maximo permitido para a insercao, esse tamanho maximo pode ser al-
904     terado posteriormente
905     ordem_de_entrada.tab_simb_count = 0;
906     ordem_de_entrada.limite_tab_simb_count = LIMITE_INICIAL_DE_ALOCACAO;
907     ordem_de_entrada.ordem_de_entrada_da_tab_simbolos = (tSimbolo**) malloc (or-
908     dem_de_entrada.limite_tab_simb_count * sizeof(tSimbolo*));
909     if (ordem_de_entrada.ordem_de_entrada_da_tab_simbolos == NULL){
910         printf("Erro durante a alocação da lista de ordenacao!!! \nInfelizmente o programa tra-
911         vou\n");
912         exit(-1);
913     }
914 }
915
916 int hash_Com_Shift(void){
917     //Funcao HASH com shift: após ela é realizada a compressao (o valor deve ser positivo
918     pois rdefinir o indice do Hash)
919     int h = 0;
920
921     for (int i = 0; i < lexema.tamanho_string; i++){
922         h += lexema.string[i];
923         h <= 2; //shift de 2 bits na soma atual
924     }
925     return abs(hash(h)); //Compressao do valor h obtido
926 }
927
928 void adicionar_Na_Tabela_De_Simbolos(const tToken tk){
929     int posicao = hash_Com_Shift();
930     tSimbolo * simb = buscar_Na_Tabela_De_Simbolos(tk, posicao);
931
932     if(simb != NULL){ //se o token já está instalado na tab simbolos, adicionar ocorrencia
933         adiciona_Ocorrencia(simb);
934     } else {
935         //Setar as caracteristicas do novo simbolo
936         simb = (tSimbolo *) malloc(sizeof(tSimbolo));
937         if (simb == NULL){
938             printf("Erro durante a alocação de um novo simbolo na tabela hash!!!
939             \nInfelizmente o programa travou\n");
940             exit(-1);
941         }
942
943         simb->COD = tk;
944         simb->posicao = posicao;
945         simb->lexema_cadeia = (char *) malloc(lexema.tamanho_string * sizeof(char));
946         strcpy(simb->lexema_cadeia, lexema.string);
947
948         if (tk == tk_INTEIRO)
949             simb->lexema_inteiro = atoi(lexema.string);
950         else if (tk == tk_DECIMAL)
951             simb->lexema_decimal = atof(lexema.string);
952
953         simb->tamanho_ocorrencias = 0;
954         simb->limite_ocorrencias = LIMITE_INICIAL_DE_ALOCACAO;
955         simb->ocorrencias = (tPos*) malloc (simb->limite_ocorrencias * sizeof(tPos));
956         if (simb->ocorrencias == NULL){
957             printf("Erro durante a alocação da lista de ocorrencias!!! \nInfelizmente o
958             programa travou\n");
959             exit(-1);
960         }
961         adiciona_Ocorrencia(simb);
962
963         //Adicionar na tabela com Insercao no inicio
964         simb->proximo = tab_simbolos[posicao];
965         tab_simbolos[posicao] = simb;
966
967         //Adicionando
968         ordem_de_entrada.ordem_de_entrada_da_tab_simbolos[ordem_de_entrada.tab_simb_count]
969         = simb;
```

```
973         ordem_de_entrada.tab_simb_count++;
974         if (ordem_de_entrada.tab_simb_count == ordem_de_entrada.limite_tab_simb_count-1){
975             ordem_de_entrada.limite_tab_simb_count *= 2;
976             ordem_de_entrada.ordem_de_entrada_da_tab_simbolos = (tSimbolo**) realloc
977 (ordem_de_entrada.ordem_de_entrada_da_tab_simbolos, ordem_de_entrada.limite_tab_simb_count *
978 sizeof(tSimbolo*));
979             if (ordem_de_entrada.ordem_de_entrada_da_tab_simbolos == NULL){
980                 printf("Erro durante a realocacao da lista de ordenacao!!!
981 \nInfelizmente o programa travou\n");
982                 exit(-1);
983             }
984         }
985     }
986 }
987
988
989 tSimbolo * buscar_Na_Tabela_De_Simbolos(const tToken tk, const int pos){
990     //combinação token-lexema deve ser incluída uma unica vez na tabela
991     //É realizada uma busca pela combinacao
992     tSimbolo * simb = tab_simbolos[pos];
993     while (simb != NULL) {
994         if (simb->COD == tk && (strcmp(lexema.string, simb->lexema_cadeia) == 0)) {
995             return simb;
996         }
997         simb = simb->proximo;
998     }
999     return NULL;
1000 }
1001
1002
1003 void adiciona_Ocorrencia(tSimbolo * simb){
1004     //Função que armazena a linha e coluna da ocorrencia de um token com lexema na tabela de
1005     simbolos
1006     simb->ocorrencias[simb->tamanho_ocorrencias].LIN = linha_token;
1007     simb->ocorrencias[simb->tamanho_ocorrencias].COL = coluna_token;
1008     simb->tamanho_ocorrencias++;
1009
1010     //Verificar tamanho da alocação
1011     if(simb->tamanho_ocorrencias == simb->limite_ocorrencias-1){
1012         simb->limite_ocorrencias *= 2;
1013         simb->ocorrencias = (tPos*) realloc (simb->ocorrencias, simb->limite_ocorrencias *
1014 sizeof(tPos));
1015         if (simb->ocorrencias == NULL){
1016             printf("Erro durante a realocacao da lista de ocorrencias!!! \nInfelizmente
1017 o programa travou\n");
1018             exit(-1);
1019         }
1020     }
1021 }
1022
1023 #endif
1024
1025
1026
1027
1028
1029
1030 /* ----- */
1031 /*          ARQUIVO: erros.h          */
1032 /* ----- */
1033 /* Certifique-se que este arquivo não é incluído mais de uma vez */
1034 /* ----- */
1035 #ifndef _ERROS_H_
1036 #define _ERROS_H_
1037
1038 /* ----- */
1039 /*          CONSTANTES, TIPOS E VARIÁVEIS          */
1040 /* ----- */
1041 #ifndef LIMITE_INICIAL_DE_ALOCACAO
1042     #define LIMITE_INICIAL_DE_ALOCACAO 30
1043 #endif
1044
1045 //Definicao de Tipos (ENUM) - Erros
1046 typedef enum{
1047     er_delimitador_esperado,
```



```
1048     er_ponto_isolado,
1049     er_cadeia_nao_fechada,
1050     er_comentario_de_bloco_nao_fechado,
1051     er_caracter_invalido
1052 } tErro;
1053
1054 //Definicao de Estruturas - Identificador de Erros Lexicos, Lista de erros lexicos
1055 typedef struct{
1056     int LIN, COL;
1057     tErro ERRO;
1058     char CARACTER;
1059 } tIdentificador_De_Erro;
1060
1061 typedef struct{
1062     int tamanho_lista;
1063     int limite_lista;
1064     tIdentificador_De_Erro * id_erro;
1065 } tLista_de_erros;
1066
1067 //VARIÁVEIS
1068 tLista_de_erros lista_de_erros;
1069
1070
1071 /* ----- */
1072 /*     PROTÓTIPOS DAS FUNÇÕES     */
1073 /* ----- */
1074 void iniciar_Lista_De_Erros(void);
1075 void adicionar_Erro_Na_Lista_De_Erros(const tErro, const char, const int, const int);
1076 const char * obter_Nome_Do_Erro(const tErro);
1077
1078 /* ----- */
1079 /*     IMPLEMENTAÇÃO DAS FUNÇÕES     */
1080 /* ----- */
1081 void iniciar_Lista_De_Erros(void){
1082     //Define tamanho da lista de erros como zero (vazio) e tamanho maximo permitido para a
1083     //insercao de erros, esse tamanho maximo pode ser alterado posteriormente
1084     lista_de_erros.tamanho_lista = 0;
1085     lista_de_erros.limite_lista = LIMITE_INICIAL_DE_ALOCACAO;
1086     lista_de_erros.id_erro = (tIdentificador_De_Erro *) malloc (lista_de_erros.limite_lista
1087 * sizeof(tIdentificador_De_Erro));
1088     if (lista_de_erros.id_erro == NULL){
1089         printf("Erro durante a alocao da lista de erros!!! \nInfelizmente o programa tra-
1090 vou\n");
1091         exit(-1);
1092     }
1093 }
1094
1095 void adicionar_Erro_Na_Lista_De_Erros(const tErro erro, const char c, const int linha, const int
1096 coluna){
1097     //Setar o erro
1098     lista_de_erros.id_erro[lista_de_erros.tamanho_lista].LIN = linha;
1099     lista_de_erros.id_erro[lista_de_erros.tamanho_lista].COL = coluna;
1100     lista_de_erros.id_erro[lista_de_erros.tamanho_lista].ERRO = erro;
1101     lista_de_erros.id_erro[lista_de_erros.tamanho_lista].CARACTER = c;
1102     lista_de_erros.tamanho_lista++;
1103
1104     //Verificar tamanho da alocao
1105     if (lista_de_erros.tamanho_lista == lista_de_erros.limite_lista-1){
1106         lista_de_erros.limite_lista *= 2;
1107         lista_de_erros.id_erro = (tIdentificador_De_Erro *) re-
1108 alloc(lista_de_erros.id_erro, lista_de_erros.limite_lista * sizeof(tIdentificador_De_Erro));
1109         if (lista_de_erros.id_erro == NULL){
1110             printf("Erro durante a realocacao da lista de erros!!! \nInfelizmente o
1111 programa travou\n");
1112             exit(-1);
1113         }
1114     }
1115 }
1116
1117 const char * obter_Nome_Do_Erro(const tErro id_token){
1118     const char * NOMES[] = {
1119         "Delimitador esperado",
1120     }
```

```
1123         "Ponto isolado",
1124         "Cadeia nao fechada",
1125         "Comentario de Bloco nao fechado",
1126         "Caracter Invalido"
1127     };
1128     return (NOMES[id_token]);
1129 }
1130
1131 #endif
1132
1133
1134
1135
1136
1137
1138 /* ----- */
1139 /*          ARQUIVO: resultados.h          */
1140 /* ----- */
1141 /* Certifique-se que este arquivo não é incluído mais de uma vez */
1142 /* ----- */
1143 #ifndef _RESULTADOS_H_
1144 #define _RESULTADOS_H_
1145
1146 /* ----- */
1147 /* BIBLIOTECAS E INCLUDE FILES */
1148 /* ----- */
1149 #include "tokens.h"
1150 #include "automato.h"
1151
1152
1153 /* ----- */
1154 /*          PROTÓTIPOS DAS FUNÇÕES          */
1155 /* ----- */
1156 char reconhecer_Proximo_Simbolo(void);
1157 int imprimir_Linha(FILE *);
1158 void imprimir_Seta(FILE *, const int);
1159 void imprimir_Lista_De_Erros_Lexicos(const char*);
1160 tSimbolo * procurar_Lexema(const tIdentificador_De_Token meuToken);
1161 void imprimir_Lista_De_Tokens_Reconhecidos_E_Resumo(const char*, int);
1162 int obter_Tamanho_Do_Maior_Nome_Token_Reconhecido(void);
1163 int obter_Tamanho_Do_Maior_Lexema(void);
1164 int obter_Tamanho_Do_Maior_Nome_Token_Com_Lexema(void);
1165 int obter_Tamanho_Da_Maior_Qtd_De_Ocorrencias(void);
1166 void imprimir_Tabela_De_Simbolos(Const char*);
1167
1168
1169 /* ----- */
1170 /*          IMPLEMENTAÇÃO DAS FUNÇÕES          */
1171 /* ----- */
1172 char reconhecer_Proximo_Simbolo(void){
1173     char prox_Simb = getc(arquivo_de_entrada);
1174     fseek(arquivo_de_entrada, -sizeof(char), SEEK_CUR);
1175     return prox_Simb;
1176 }
1177
1178
1179 int imprimir_Linha(FILE * arquivo_de_saida){
1180     //Usada durante a impressao de erros lexicos para a entrada portugol
1181     //Essa funcao é responsavel por imprimir em arquivo uma linha do aquivo texto
1182     int caracteres_na_linha = 0;
1183     char prox_Simb = ' ';
1184     while (prox_Simb != '\n' && prox_Simb != EOF){
1185         fprintf(arquivo_de_saida, "%c", prox_Simb);
1186         prox_Simb = ler_Proximo_Caractere();
1187         caracteres_na_linha++;
1188     }
1189     fprintf(arquivo_de_saida, "\n");
1190     return caracteres_na_linha;
1191 }
1192
1193
1194 void imprimir_Seta(FILE * arquivo_de_saida, const int n){
1195     //Essa funcao é responsável por imprimir em arquivo uma seta indicando um erro lexico em
1196     uma determinada linha
1197     fprintf(arquivo_de_saida, "          ");
```

```
1198     for (int i=0; i<n-1; i++)
1199         fprintf(arquivo_de_saida, "-");
1200     fprintf(arquivo_de_saida, "\n");
1201 }
1202
1203
1204 void imprimir_Lista_De_Erros_Lexicos(const char* nomeArquivoEntrada){
1205     int caracteres_na_linha = -1, i = 0; //Variaveis
1206     FILE *arquivo_de_saida;
1207     char * nome_arquivo;
1208
1209     rewind(arquivo_de_entrada); //Voltar ao inicio do arquivo
1210     linha_arquivo = coluna_arquivo = 1; //Zerar linha e coluna
1211
1212     nome_arquivo = (char *) malloc(4 + strlen(nomeArquivoEntrada)); // Alocação para o nome
1213 do arquivo
1214     sprintf(nome_arquivo, "%s.err", nomeArquivoEntrada); // O arquivo esta na pasta testes
1215
1216     if ((arquivo_de_saida = fopen(nome_arquivo, "w")) == NULL){
1217         //Erro ao abrir o arquivo
1218         printf("Erro ao abrir o arquivo para a saida de erros lexicos!!!\n");
1219     } else {
1220         //Inserindo dados no arquivo
1221         fprintf(arquivo_de_saida, "LISTA DE ERROS LEXICOS EM \"%s\" \n\n", nomeArquivoEn-
1222 trada);
1223         while(!feof(arquivo_de_entrada)){
1224             if (reconhecer_Proximo_Simbolo() == EOF)
1225                 break;
1226             fprintf(arquivo_de_saida, "[%4d]", linha_arquivo);
1227             caracteres_na_linha = imprimir_Linha(arquivo_de_saida);
1228             if (lista_de_erros.id_erro[i].LIN == linha_arquivo-1 && i < lis-
1229 ta_de_erros.tamanho_lista){
1230                 imprimir_Seta(arquivo_de_saida, lista_de_erros.id_erro[i].COL);
1231                 if (lista_de_erros.id_erro[i].ERRO == er_caracter_invalido)
1232                     fprintf(arquivo_de_saida, "          Erro lexico na linha %d
1233 coluna %d: %s '%c'\n", lista_de_erros.id_erro[i].LIN, lista_de_erros.id_erro[i].COL, ob-
1234 ter_Nome_Do_Erro(lista_de_erros.id_erro[i].ERRO), lista_de_erros.id_erro[i].CARACTER);
1235                 else
1236                     fprintf(arquivo_de_saida, "          Erro lexico na linha %d
1237 coluna %d: %s \n", lista_de_erros.id_erro[i].LIN, lista_de_erros.id_erro[i].COL, ob-
1238 ter_Nome_Do_Erro(lista_de_erros.id_erro[i].ERRO));
1239                 i++;
1240                 if (lista_de_erros.id_erro[i].LIN == linha_arquivo-1)
1241                     retroceder_Caracteres(caracteres_na_linha, '\n');
1242             }
1243         }
1244
1245         fprintf(arquivo_de_saida, "\nTOTAL DE ERROS: %d\n\n", lis-
1246 ta_de_erros.tamanho_lista);
1247         //Fechando arquivo de saida e liberando memoria alocada
1248         fclose(arquivo_de_saida);
1249         free(nome_arquivo);
1250     }
1251 }
1252
1253
1254 tSimbolo * procurar_Lexema(const tIdentificador_De_Token meuToken){
1255     tSimbolo * simb = tab_simbolos[meuToken.posicao_na_tabela_de_simbolos];
1256     while (simb != NULL) {
1257         if (simb->COD == meuToken.TOKEN) {
1258             for(int i=0; i < simb->tamanho_ocorrencias; i++){
1259                 if (simb->ocorrencias[i].LIN == meuToken.LIN && simb->
1260 >ocorrencias[i].COL == meuToken.COL)
1261                     return simb;
1262             }
1263         }
1264         simb = simb->proximo;
1265     }
1266     return NULL;
1267 }
1268
1269
1270 void imprimir_Lista_De_Tokens_Reconhecidos_E_Resumo(const char* nomeArquivoEntrada, int qtdEr-
1271 ros){
1272     int resumo[QUANTIDADE_DE_TOKENS] = {0};
```

```
1273     int token_max = obter_Tamanho_Do_Maior_Nome_Token_Reconhecido();
1274     int lexema_max = obter_Tamanho_Do_Maior_Lexema();
1275     int linha_Da_Vez = 0;
1276
1277     FILE *arquivo_de_saida;
1278     char * nome_arquivo;
1279
1280     nome_arquivo = (char *) malloc(4 + strlen(nomeArquivoEntrada)); // Alocação para o nome
1281 do arquivo
1282     sprintf(nome_arquivo, "%s.tok", nomeArquivoEntrada); // O arquivo esta na pasta testes
1283
1284     if ((arquivo_de_saida = fopen(nome_arquivo, "w")) == NULL){
1285         //Erro ao abrir o arquivo
1286         printf("Erro ao abrir o arquivo para a saida de erros lexicos!!!\n");
1287     } else {
1288         fprintf(arquivo_de_saida, "LISTA DE TOKENS RECONHECIDOS EM \"%s\" \n\n", nomeArqui-
1289 voEntrada);
1290
1291         fprintf(arquivo_de_saida, "+-----+-----+-----+");
1292         for (int i=0; i<token_max; i++)
1293             fprintf(arquivo_de_saida, "-");
1294         fprintf(arquivo_de_saida, "+-");
1295         for (int i=0; i<lexema_max; i++)
1296             fprintf(arquivo_de_saida, "-");
1297         fprintf(arquivo_de_saida, "-+-----+-----+\\n");
1298
1299         fprintf(arquivo_de_saida, "| LIN | COL | COD | %-*s | %-*s | POS TAB SIMB |\\n",
1300 token_max, "TOKEN", lexema_max, "LEXEMA");
1301
1302         fprintf(arquivo_de_saida, "+-----+-----+-----+");
1303         for (int i=0; i<token_max; i++)
1304             fprintf(arquivo_de_saida, "-");
1305         fprintf(arquivo_de_saida, "+-");
1306         for (int i=0; i<lexema_max; i++)
1307             fprintf(arquivo_de_saida, "-");
1308         fprintf(arquivo_de_saida, "-+-----+-----+\\n");
1309
1310         for (int i=0; i<lista_de_tokens.tamanho_lista; i++){
1311             resumo[lista_de_tokens.id_token[i].TOKEN]++;
1312
1313             if (lista_de_tokens.id_token[i].LIN == linha_Da_Vez){
1314                 fprintf(arquivo_de_saida, "|      | ");
1315             } else{
1316                 fprintf(arquivo_de_saida, "| %3d | ", lis-
1317 ta_de_tokens.id_token[i].LIN);
1318             }
1319
1320             fprintf(arquivo_de_saida, "%3d | %3d | %-*s | ", lis-
1321 ta_de_tokens.id_token[i].COL, lista_de_tokens.id_token[i].TOKEN+1, token_max, ob-
1322 ter_Nome_Do_Token(lista_de_tokens.id_token[i].TOKEN));
1323
1324             if (lista_de_tokens.id_token[i].posicao_na_tabela_de_simbolos == -1){
1325                 fprintf(arquivo_de_saida, "%-*s |      |\\n", lexema_max, "
1326 ");
1327             } else{
1328                 tSimbolo * simb = procurar_Lexema(lista_de_tokens.id_token[i]);
1329
1330                 if (simb->COD == tk_INTEIRO){
1331                     fprintf(arquivo_de_saida, "%-*d |      %3d      |\\n", lexe-
1332 ma_max, simb->lexema_inteiro, lista_de_tokens.id_token[i].posicao_na_tabela_de_simbolos);
1333                 } else if (simb->COD == tk_DECIMAL){
1334                     int aux = strlen(simb->lexema_cadeia);
1335                     int qtd_apos = 1, qtd_antes = 0;
1336                     for (int j=0; j<aux; j++){
1337                         qtd_antes++; if (simb->lexema_cadeia[j]=='.') break;
1338                     }
1339                     qtd_apos = aux - qtd_antes;
1340                     if (qtd_apos == 0) qtd_apos++;
1341                     if (qtd_apos >= lexema_max-1) qtd_apos--;
1342                     fprintf(arquivo_de_saida, "%-*.f |      %3d      |\\n", lexe-
1343 ma_max, qtd_apos, simb->lexema_decimal, lis-
1344 ta_de_tokens.id_token[i].posicao_na_tabela_de_simbolos);
1345                 } else{
1346
```

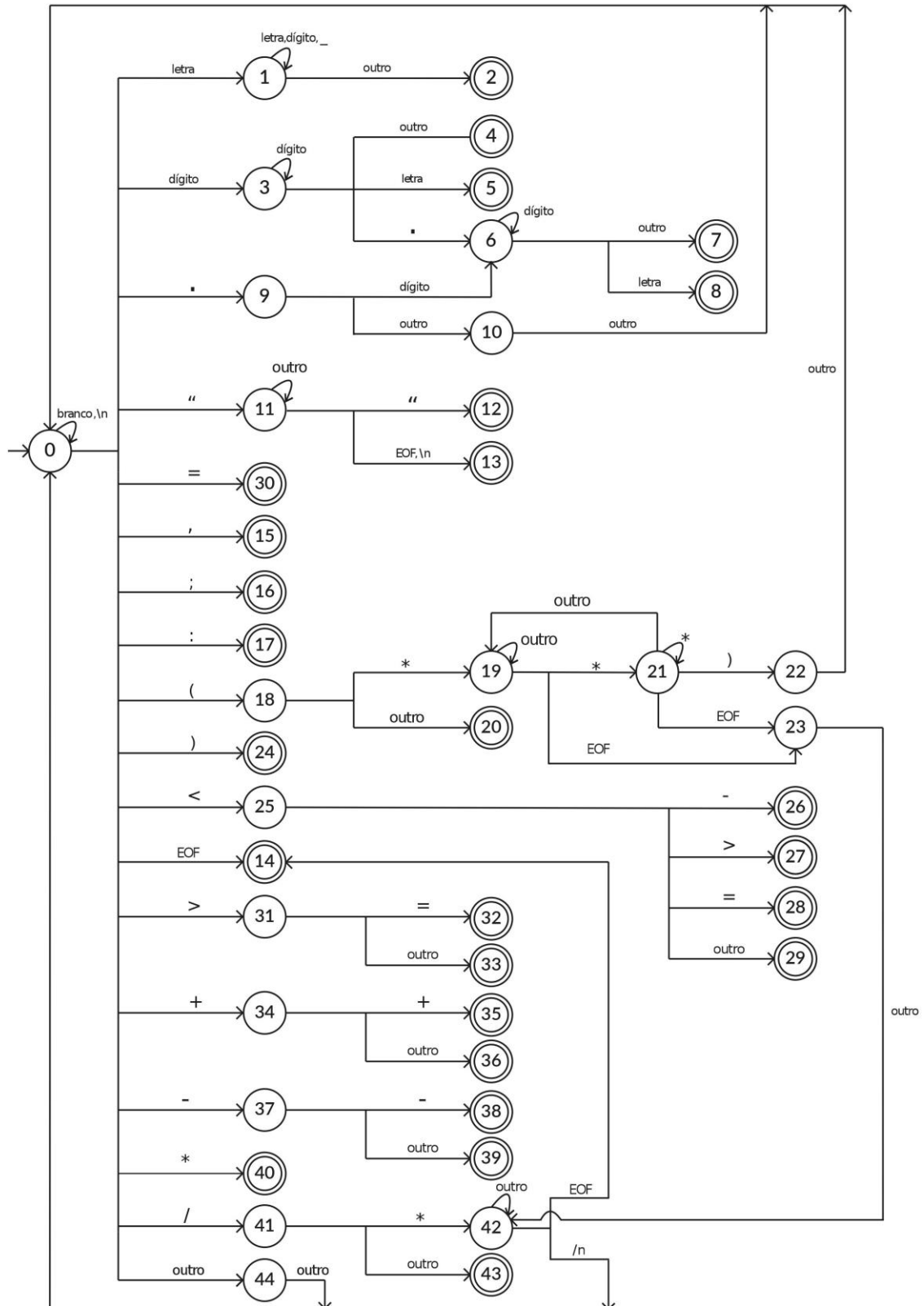
19

```
1422         return maior;
1423     }
1424
1425
1426 int obter_Tamanho_Da_Maior_Qtd_De_Ocorrencias(void){
1427     int maior = 3; //Tamanho de "POS NA ENTRADA (linha,coluna)" /10
1428     for (int i=0; i < ordem_de_entrada.tab_simb_count; i++){
1429         if (ordem_de_entrada.ordem_de_entrada_da_tab_simbolos[i]->tamanho_ocorrencias >
1430 maior)
1431             maior = ordem_de_entrada.ordem_de_entrada_da_tab_simbolos[i]-
1432 >tamanho_ocorrencias;
1433     }
1434     return maior;
1435 }
1436
1437
1438 void imprimir_Tabela_De_Simbolos(const char* nomeArquivoEntrada){
1439     int token_max = obter_Tamanho_Do_Maior_Nome_Token_Com_Lexema();
1440     int lexema_max = obter_Tamanho_Do_Maior_Lexema();
1441     int maior_numero_ocorrencias = obter_Tamanho_Da_Maior_Qtd_De_Ocorrencias() * 10;
1442     FILE *arquivo_de_saida;
1443     char * nome_arquivo;
1444
1445     nome_arquivo = (char *) malloc(4 + strlen(nomeArquivoEntrada)); // Alocação para o nome
1446 do arquivo
1447     sprintf(nome_arquivo,"%s.tbl", nomeArquivoEntrada); // O arquivo esta na pasta testes
1448
1449     if ((arquivo_de_saida = fopen(nome_arquivo, "w")) == NULL){
1450         //Erro ao abrir o arquivo
1451         printf("Erro ao abrir o arquivo para a saida de erros lexicos!!!\n");
1452     } else {
1453         fprintf(arquivo_de_saida,"TABELA DE SIMBOLOS - \"%s\" \n\n", nomeArquivoEntrada);
1454         fprintf(arquivo_de_saida,"+-----+");
1455         for (int i=0; i<token_max; i++)
1456             fprintf(arquivo_de_saida,"-");
1457         fprintf(arquivo_de_saida,"-+");
1458         for (int i=0; i<lexema_max; i++)
1459             fprintf(arquivo_de_saida,"-");
1460         fprintf(arquivo_de_saida,"-+");
1461         for (int i=0; i<maior_numero_ocorrencias; i++)
1462             fprintf(arquivo_de_saida,"-");
1463         fprintf(arquivo_de_saida,"+\n");
1464
1465         fprintf(arquivo_de_saida,"| POS | %-*s | %-*s | %-*s|\n", token_max, "TOKEN",
1466 lexema_max, "LEXEMA", maior_numero_ocorrencias, "POS NA ENTRADA (linha,coluna)");
1467
1468         fprintf(arquivo_de_saida,"+-----+");
1469         for (int i=0; i<token_max; i++)
1470             fprintf(arquivo_de_saida,"-");
1471         fprintf(arquivo_de_saida,"-+");
1472         for (int i=0; i<lexema_max; i++)
1473             fprintf(arquivo_de_saida,"-");
1474         fprintf(arquivo_de_saida,"-+");
1475         for (int i=0; i<maior_numero_ocorrencias; i++)
1476             fprintf(arquivo_de_saida,"-");
1477         fprintf(arquivo_de_saida,"+\n");
1478
1479         for (int i=0; i<ordem_de_entrada.tab_simb_count; i++){
1480             fprintf(arquivo_de_saida,"| %3d | %-*s | ", or-
1481 dem_de_entrada.ordem_de_entrada_da_tab_simbolos[i]->posicao, token_max, ob-
1482 ter_Nome_Do_Token(ordem_de_entrada.ordem_de_entrada_da_tab_simbolos[i]->COD));
1483
1484             //fprintf(arquivo_de_saida,"%-*s", lexema_max, or-
1485 dem_de_entrada.ordem_de_entrada_da_tab_simbolos[i]->lexema_cadeia);
1486             if (ordem_de_entrada.ordem_de_entrada_da_tab_simbolos[i]->COD ==
1487 tk_INTEIRO){
1488                 fprintf(arquivo_de_saida,"%-*d", lexema_max, or-
1489 dem_de_entrada.ordem_de_entrada_da_tab_simbolos[i]->lexema_inteiro);
1490             }else if(ordem_de_entrada.ordem_de_entrada_da_tab_simbolos[i]->COD ==
1491 tk_DECIMAL){
1492                 int aux = str-
1493 len(ordem_de_entrada.ordem_de_entrada_da_tab_simbolos[i]->lexema_cadeia);
1494                 int qtd_apos = 1, qtd_antes = 0;
1495                 for (int j=0; j<aux; j++){
```

```
1496             qtd_antes++; if (or-
1497 dem_de_entrada.ordem_de_entrada_da_tab_simbolos[i]->lexema_cadeia[j]=='.') break;
1498             }
1499             qtd_apos = aux - qtd_antes;
1500             if (qtd_apos == 0) qtd_apos++;
1501
1502             fprintf(arquivo_de_saida,"%-*.f", lexema_max, qtd_apos, or-
1503 dem_de_entrada.ordem_de_entrada_da_tab_simbolos[i]->lexema_decimal);
1504             }else{
1505             fprintf(arquivo_de_saida,"%-*s", lexema_max, or-
1506 dem_de_entrada.ordem_de_entrada_da_tab_simbolos[i]->lexema_cadeia);
1507             }
1508
1509             fprintf(arquivo_de_saida," | ");
1510             for(int j=0; j<ordem_de_entrada.ordem_de_entrada_da_tab_simbolos[i]-
1511 >tamanho_ocorrencias; j++)
1512                 fprintf(arquivo_de_saida,"%3d,%3d ", or-
1513 dem_de_entrada.ordem_de_entrada_da_tab_simbolos[i]->ocorrencias[j].LIN, or-
1514 dem_de_entrada.ordem_de_entrada_da_tab_simbolos[i]->ocorrencias[j].COL);
1515             fprintf(arquivo_de_saida,"%*s\n", maior_numero_ocorrencias - or-
1516 dem_de_entrada.ordem_de_entrada_da_tab_simbolos[i]->tamanho_ocorrencias * 10,"");
1517             }
1518
1519             fprintf(arquivo_de_saida,"+-----+");
1520             for (int i=0; i<token_max; i++)
1521                 fprintf(arquivo_de_saida,"-");
1522             fprintf(arquivo_de_saida,"-+-");
1523             for (int i=0; i<lexema_max; i++)
1524                 fprintf(arquivo_de_saida,"-");
1525             fprintf(arquivo_de_saida,"-+-");
1526             for (int i=0; i<maior_numero_ocorrencias; i++)
1527                 fprintf(arquivo_de_saida,"-");
1528             fprintf(arquivo_de_saida,"+\n\n");
1529
1530             //Fechando arquivo de saida e liberando memoria alocada
1531             fclose(arquivo_de_saida);
1532             free(nome_arquivo);
1533         }
1534     }
1535
1536 #endif
```

2. Autômato

2.1. Diagrama de estados



2.2. Ações

Estado	Ações a serem efetuadas							
	Devolver caracteres à entrada (Quantidade)	Emitir mensagem de erro (código do erro)	Instalar lexema na tabela de símbolos	Retornar token (nome)	Definir linha e coluna de início do token	Inserir caracteres no lexema	Zerar lexema	Contar caracteres lidos para casos de erro
0					Sim			
1						Sim		
2	1		Sim (tk_IDEN)	tk_IDEN ou Palavra reservada				
3						Sim		
4	1		Sim	tk_INTEIRO				
5	1	00	Sim	tk_INTEIRO				
6						Sim		
7	1		Sim	tk_DECIMAL				
8	1	00	Sim	tk_DECIMAL				
9						Sim		
10	2	01					Sim	
11						Sim		
12			Sim	tk_CADEIA		Sim		
13	1	02	Sim	tk_CADEIA				
14				tk_EOF				
15				tk_virg				
16				tk_pt_virg				
17				tk_dois_pts				
18								Sim
19								Sim
20	1			tk_abre_par				
21								Sim
22	1							
23	n	03						
24				tk_fecha_par				
26				tk_atrib				
27				tk_diferente				
28				tk_menor_igual				

29	1			tk_menor				
30				tk_igual				
32				tk_maior_igual				
33	1			tk_maior				
35				tk_incr				
36	1			tk_mais				
38				tk_decr				
39	1			tk_menos				
40				tk_vezes				
43	1			tk_dividido				
44	1	04						

Erros léxicos	
Código	Mensagem
00	Delimitador esperado
01	Ponto isolado
02	Cadeia não fechada
03	Comentário de bloco não fechado
04	Caractere Inválido

2.3. Tabela de transições

Estado atual	Próximo estado																				
	\b	\n	L	D	_	"	.	,	;	:	()	<	=	>	+	-	*	/	E	O
0	0	0	1	3	44	11	9	15	16	17	18	24	25	30	31	34	37	40	41	14	44
1	2	2	1	1	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
3	4	4	5	3	4	4	6	4	4	4	4	4	4	4	4	4	4	4	4	4	4
6	7	7	8	6	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
9	10	10	10	6	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	11	13	11	11	11	12	11	11	11	11	11	11	11	11	11	11	11	11	11	13	11
18	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	19	20	20	20
19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	21	19	23	19
21	19	19	19	19	19	19	19	19	19	19	19	22	19	19	19	19	19	21	19	23	19
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	42	42	42	42	42	42	42	42	42	42	42	42	42	42	42	42	42	42	42	42	42
25	29	29	29	29	29	29	29	29	29	29	29	29	29	28	27	29	26	29	29	29	29
31	33	33	33	33	33	33	33	33	33	33	33	33	33	32	33	33	33	33	33	33	33
34	36	36	36	36	36	36	36	36	36	36	36	36	36	36	36	35	36	36	36	36	36
37	39	39	39	39	39	39	39	39	39	39	39	39	39	39	39	39	38	39	39	39	39
41	43	43	43	43	43	43	43	43	43	43	43	43	43	43	43	43	43	42	43	43	43
42	42	0	42	42	42	42	42	42	42	42	42	42	42	42	42	42	42	42	42	14	42
44	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Legenda:

\b: Branco
L: Letra
D: Dígito
E: EOF
O: Qualquer outro valor char

*Os estados não listados 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 late xpassol s/**++*/e 2<3
  paraidel ate9passo 1.2.3.<----(***)
  para i de late 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 late xpasso1 s/*****/e 2<3
      -----^
      Erro lexico na linha 22 coluna 14: Delimitador esperado
[23]   paraide1 ate9passo 1.2.3.<----(***)
      -----^
      Erro lexico na linha 23 coluna 27: Ponto isolado
[24]   para i de late 9. 0. passo 1$j
      -----^
      Erro lexico na linha 24 coluna 14: Delimitador esperado
[24]   para i de late 9. 0. passo 1$j
      -----^
      Erro lexico na linha 24 coluna 31: Caracter 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	73		
	11	25	tk_pt_virg				
3	3	8	tk_int				
	6	26	tk_dois_pts				
	8	2	tk_IDEN	nUm	22		
	11	24	tk_virg				
	13	2	tk_IDEN	NuM	30		
	16	25	tk_pt_virg				
4	3	8	tk_int				
	6	26	tk_dois_pts				
	8	2	tk_IDEN	Maior	87		
	13	25	tk_pt_virg				
5	3	11	tk_imprima				
	11	27	tk_abre_par				
	12	5	tk_CADEIA	"Digite 3 nros:"	47		
	28	28	tk_fecha_par				
	29	25	tk_pt_virg				
6	3	10	tk_leia				
	8	27	tk_abre_par				
	9	2	tk_IDEN	num	73		
	12	28	tk_fecha_par				
	13	25	tk_pt_virg				
7	3	10	tk_leia				
	8	27	tk_abre_par				
	9	2	tk_IDEN	nUm	22		
	12	28	tk_fecha_par				
	13	25	tk_pt_virg				
8	3	10	tk_leia				
	8	27	tk_abre_par				
	9	2	tk_IDEN	NuM	30		
	12	28	tk_fecha_par				
	13	25	tk_pt_virg				
9	3	17	tk_se				
	6	27	tk_abre_par				
	7	2	tk_IDEN	num	73		
	10	32	tk_maior_igual				
	12	2	tk_IDEN	nUm	22		
	15	28	tk_fecha_par				
	17	21	tk_e				
	19	27	tk_abre_par				
	20	2	tk_IDEN	num	73		
	23	32	tk_maior_igual				
	25	2	tk_IDEN	NuM	30		
	28	28	tk_fecha_par				
	30	18	tk_entao				
10	5	2	tk_IDEN	maior	15		
	11	37	tk_atrib				
	14	2	tk_IDEN	num	73		
	17	25	tk_pt_virg				
11	3	19	tk_senao				
12	3	17	tk_se				
	6	27	tk_abre_par				
	7	2	tk_IDEN	nUm	22		
	10	32	tk_maior_igual				
	12	2	tk_IDEN	num	73		
	15	28	tk_fecha_par				
	17	21	tk_e				
	19	27	tk_abre_par				
	20	2	tk_IDEN	nUm	22		
	23	32	tk_maior_igual				
	25	2	tk_IDEN	NuM	30		
	28	28	tk_fecha_par				
	30	18	tk_entao				

13	5	2	tk_IDEN	MAIOR	11
	11	37	tk_atrib		
	14	2	tk_IDEN	nUm	22
	17	25	tk_pt_virg		
14	3	19	tk_senao		
15	5	2	tk_IDEN	Maior	87
	11	37	tk_atrib		
	14	2	tk_IDEN	NuM	30
	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	Maior	87
	17	28	tk_fecha_par		
	18	25	tk_pt_virg		
19	3	2	tk_IDEN	NuM	30
	7	37	tk_atrib		
	13	39	tk_menos		
	14	4	tk_DECIMAL	0.12	75
	18	39	tk_menos		
	20	38	tk_mais		
	21	3	tk_INTEIRO	1	13
	25	39	tk_menos		
	27	3	tk_INTEIRO	1234567890	78
	38	38	tk_mais		
	40	39	tk_menos		
	41	4	tk_DECIMAL	0.0	52
	44	39	tk_menos		
	46	38	tk_mais		
	47	4	tk_DECIMAL	0.0	100
	50	39	tk_menos		
	52	4	tk_DECIMAL	12345.67871	27
	64	25	tk_pt_virg		
20	3	2	tk_IDEN	nUm	22
	6	37	tk_atrib		
	8	39	tk_menos		
	9	4	tk_DECIMAL	0.12	75
	12	39	tk_menos		
	13	38	tk_mais		
	14	3	tk_INTEIRO	1	13
	17	39	tk_menos		
	18	3	tk_INTEIRO	1234567890	78
	28	38	tk_mais		
	29	39	tk_menos		
	30	4	tk_DECIMAL	0.0	52
	32	39	tk_menos		
	33	38	tk_mais		
	34	4	tk_DECIMAL	0.0	100
	36	39	tk_menos		
	37	4	tk_DECIMAL	12345.67871	27
	48	25	tk_pt_virg		
21	3	2	tk_IDEN	num	73
	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	117
	14	14	tk_ate		
	18	2	tk_IDEN	xpassol	103
	26	2	tk_IDEN	s	89
23	3	2	tk_IDEN	paraidel	82
	12	2	tk_IDEN	ate9passo	40
	22	4	tk_DECIMAL	1.2	85
	25	4	tk_DECIMAL	0.3	76
	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		9	
		10		13		tk_de					
		13		3		tk_INTEIRO		1		117	
		14		14		tk_ate					
		18		4		tk_DECIMAL		9.0		110	
		21		4		tk_DECIMAL		0.0		100	
		24		15		tk_passo					
		30		3		tk_INTEIRO		1		117	
		32		2		tk_IDEN		j		17	
	25		1		7		tk_fim				
	26		0		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)
73	tk_IDEN	num	(2, 8) (6, 9) (9, 7) (9, 20) (10, 14) (12, 12) (21, 3)
22	tk_IDEN	nUm	(3, 8) (7, 9) (9, 12) (12, 7) (12, 20) (13, 14) (20, 3)
30	tk_IDEN	NuM	(3, 13) (8, 9) (9, 25) (12, 25) (15, 14) (19, 3)
87	tk_IDEN	Maior	(4, 8) (15, 5) (18, 12)
47	tk_CADEIA	"Digite 3 nros:"	(5, 12)
15	tk_IDEN	maior	(10, 5)
11	tk_IDEN	MAIOR	(13, 5)
73	tk_CADEIA	"Maior = "	(17, 12)
75	tk_DECIMAL	0.12	(19, 14) (20, 9)
13	tk_INTEIRO	1	(19, 21) (20, 14)
78	tk_INTEIRO	1234567890	(19, 27) (20, 18)
52	tk_DECIMAL	0.0	(19, 41) (20, 30)
100	tk_DECIMAL	0.0	(19, 47) (20, 34) (24, 21)
27	tk_DECIMAL	12345.67871	(19, 52) (20, 37)
117	tk_INTEIRO	1	(22, 13) (24, 13) (24, 30)
103	tk_IDEN	xpassol	(22, 18)
89	tk_IDEN	s	(22, 26)
82	tk_IDEN	paraidel	(23, 3)
40	tk_IDEN	ate9passo	(23, 12)
85	tk_DECIMAL	1.2	(23, 22)
76	tk_DECIMAL	0.3	(23, 25)
9	tk_IDEN	i	(24, 8)
110	tk_DECIMAL	9.0	(24, 18)
17	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: Character Invalido 'á'
[ 2] // Comentário 2 //
      -----^
      Erro lexico na linha 2 coluna 10: Character Invalido 'á'
[ 3] /* Comentário 3
[ 4] /* Comentário 4 */
[ 5] (* Comentário 5 *)
[ 6]
[ 7] início
      --^
      Erro lexico na linha 7 coluna 3: Character 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: Character 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: Character 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	89		
	11	2	tk_IDEN	rio	78		
	15	3	tk_INTEIRO	1	117		
2	1	41	tk_dividido				
	2	41	tk_dividido				
	4	2	tk_IDEN	Coment	89		
	11	2	tk_IDEN	rio	78		
	15	3	tk_INTEIRO	2	125		
	17	41	tk_dividido				
	18	41	tk_dividido				
7	1	2	tk_IDEN	in	73		
	4	2	tk_IDEN	cio	104		
8	3	8	tk_int				
	6	26	tk_dois_pts				
	8	2	tk_IDEN	x	129		
	9	24	tk_virg				
	11	2	tk_IDEN	y	137		
	12	24	tk_virg				
	14	2	tk_IDEN	z	6		
	15	25	tk_pt_virg				
9	38	2	tk_IDEN	decimal	127		
	45	26	tk_dois_pts				
	47	2	tk_IDEN	m	41		
	49	2	tk_IDEN	dia	120		
	52	25	tk_pt_virg				
10	3	11	tk_imprima				
	11	27	tk_abre_par				
	12	5	tk_CADEIA	"Digite um valor para x:"	54		
	37	28	tk_fecha_par				
	38	25	tk_pt_virg				
11	3	10	tk_leia				
	8	27	tk_abre_par				
	9	2	tk_IDEN	x	129		
	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:"	65		
	36	28	tk_fecha_par				
	37	25	tk_pt_virg				
13	3	10	tk_leia				
	7	27	tk_abre_par				
	8	2	tk_IDEN	y	137		
	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:"	76		
	37	28	tk_fecha_par				
	38	25	tk_pt_virg				
15	3	10	tk_leia				
	8	27	tk_abre_par				
	9	2	tk_IDEN	z	6		
	10	28	tk_fecha_par				
	11	25	tk_pt_virg				
16	3	2	tk_IDEN	media	122		
	9	37	tk_atrib				
	12	27	tk_abre_par				
	13	2	tk_IDEN	x	129		
	14	38	tk_mais				
	15	2	tk_IDEN	y	137		
	16	38	tk_mais				
	17	2	tk_IDEN	z	6		
	18	28	tk_fecha_par				

	19	40	tk vezes			
	20	4	tk_DECIMAL	0.33333	74	
	27	25	tk_pt_virg			
17	3	11	tk_imprima			
	11	27	tk_abre_par			
	12	5	tk_CADEIA	"Média = "	7	
	22	28	tk_fecha_par			
	23	25	tk_pt_virg			
18	3	11	tk_imprima			
	13	27	tk_abre_par			
	14	2	tk_IDEN	m	41	
	16	2	tk_IDEN	dia	120	
	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"	36	
	19	28	tk_fecha_par			
	20	25	tk_pt_virg			
20	1	7	tk_fim			
21	0	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)
89	tk_IDEN	Coment	(1, 4) (2, 4)
78	tk_IDEN	rio	(1, 11) (2, 11)
117	tk_INTEIRO	1	(1, 15)
125	tk_INTEIRO	2	(2, 15)
73	tk_IDEN	in	(7, 1)
104	tk_IDEN	cio	(7, 4)
129	tk_IDEN	x	(8, 8) (11, 9) (16, 13)
137	tk_IDEN	y	(8, 11) (13, 8) (16, 15)
6	tk_IDEN	z	(8, 14) (15, 9) (16, 17)
127	tk_IDEN	decimal	(9, 38)
41	tk_IDEN	m	(9, 47) (18, 14)
120	tk_IDEN	dia	(9, 49) (18, 16)
54	tk_CADEIA	"Digite um valor para x:"	(10, 12)
65	tk_CADEIA	"Digite um valor para y:"	(12, 11)
76	tk_CADEIA	"Digite um valor para z:"	(14, 12)
122	tk_IDEN	media	(16, 3)
74	tk_DECIMAL	0.33333	(16, 20)
7	tk_CADEIA	"Média = "	(17, 12)
36	tk_CADEIA	"\n"	(19, 15)

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	9
	9	24	tk_virg		
	11	2	tk_IDEN	n	49
	12	24	tk_virg		
	14	2	tk_IDEN	fat	133
	17	25	tk_pt_virg		
5	3	11	tk_imprima		
	11	27	tk_abre_par		
	12	5	tk_CADEIA	"Digite um nro: "	20
	29	28	tk_fecha_par		
	30	25	tk_pt_virg		
6	3	10	tk_leia		
	8	27	tk_abre_par		
	9	2	tk_IDEN	n	49
	10	28	tk_fecha_par		
	11	25	tk_pt_virg		
7	3	12	tk_para		
	8	2	tk_IDEN	i	9
	10	13	tk_de		
	13	4	tk_DECIMAL	1.0	69
	17	14	tk_ate		
	21	2	tk_IDEN	n	49
	23	15	tk_passo		
	29	27	tk_abre_par		
	30	3	tk_INTEIRO	2	125
	31	40	tk_vezes		
	32	4	tk_DECIMAL	0.5	92
	34	28	tk_fecha_par		
8	5	2	tk_IDEN	fat	133
	9	34	tk_igual		
	11	2	tk_IDEN	fat	133
	14	40	tk_vezes		
	15	2	tk_IDEN	i	9
	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 = "	76
	25	28	tk_fecha_par		
	26	25	tk_pt_virg		
11	3	11	tk_imprima		
	11	27	tk_abre_par		
	12	2	tk_IDEN	fat	133
	15	28	tk_fecha_par		
	16	25	tk_pt_virg		
12	1	7	tk_fim		
	3	1	tk_EOF		

RESUMO

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)
9	tk_IDEN	i	(4, 8) (7, 8) (8, 15)
49	tk_IDEN	n	(4, 11) (6, 9) (7, 21)
133	tk_IDEN	fat	(4, 14) (8, 5) (8, 11) (11, 12)
20	tk_CADEIA	"Digite um nro: "	(5, 12)
69	tk_DECIMAL	1.0	(7, 13)
125	tk_INTEIRO	2	(7, 30)
92	tk_DECIMAL	0.5	(7, 32)
76	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 <i>token</i> para sinalizar um comentário			
[C04] () Retorna <i>token</i> para sinalizar um erro léxico			
[C05] () Retorna <i>token</i> 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 <i>token</i>			
[E08] () Cadeia do nome do <i>token</i>			
[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 <code>return</code>		
[E13]	() Um por chamada via variável global, p.ex. <code>int</code> ou <code>array</code>		
[E14]	() Todos de uma vez ao final da análise léxica		
[E15]	() Outro:		
Transições			
[F01]	() Guiadas por comandos <code>if / switch</code>		
[F02]	() Guiadas por tabela de transições		
[F03]	() Tabela de transições implementada <i>Descreva a estrutura – lista encadeada, array, tabela hash?</i>		
[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 <i>Descreva a estrutura – lista encadeada, array, tabela hash?</i>		
[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 <i>string</i> (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 <i>token</i> no arquivo de entrada			
[G48] ()			
[G49] ()			
[G50] ()			