

**Universidade da Beira Interior**  
**Departamento de Informática**



**Mensagens Codificadas: Sequencias Equidistantes  
de Símbolos**

Unidade Curricular:

**Algoritmos e Estruturas de Dados**

Orientador:

**Professor Hugo Proença**

11 de Janeiro de 2018

# Conteúdo

<b>Conteúdo</b>	<b>0</b>
<b>Lista de Tabelas</b>	<b>1</b>
<b>1 Objetivos</b>	<b>3</b>
<b>2 Implementação</b>	<b>4</b>
2.1 Motivação . . . . .	4
2.2 A. Construção de um Analisador Léxico: . . . . .	4
2.2.1 A1: Implementar em <i>LEX</i> o algoritmo standard dum Analisador Léxico para reconhecer os <i>tokens</i> que compõem um endereço URL. . . . .	4
2.2.2 A2:Desenhar o autómato para reconhecimento dos <i>tokens</i> que compõem os endereços URL. . . . .	5
2.2.3 A3:Implementar em C o algoritmo standard dum Analisador Léxico com tratamento de erros para reconhecer os <i>tokens</i> que compõem um endereço URL. . . . .	8
2.3 B – Construção de um Analisador Sintático . . . . .	10
2.3.1 1 – Gramática independente de contexto geradora da linguagem . . . . .	10
2.3.2 2 – Tabela LL da Linguagem . . . . .	11
2.3.3 Conjunto dos First's, Follow's e Lookaheads: . . . . .	11
2.3.4 3 – Analisador Sintático em C . . . . .	13
2.4 Conclusões . . . . .	17
<b>3 Reflexão crítica e problemas encontrados</b>	<b>18</b>
3.1 Objetivos Propostos vs Alcançados . . . . .	18
3.2 Divisão de Tarefas . . . . .	19
<b>4 Conclusões</b>	<b>20</b>
4.1 Conclusões Principais . . . . .	20

# Lista de Tabelas

2.1	Tabela de transições . . . . .	7
2.2	Tabela LL da linguagem . . . . .	11

## List of Listings

1	Código usado para definir os vários <i>tokens</i> . . . . .	5
2	<i>Switch</i> usado para analisar e guardar os <i>tokens</i> reconhecidos e respectivos códigos. . . . .	9
3	Chamada da função do analisador sintático na função <i>int main()</i> .	14
4	Código de inicialização da pilha e conversão do código do token .	15
5	Parte do código responsável pela consulta e movimentação dos estados da pilha . . . . .	16

# Capítulo 1

## Objetivos

No âmbito da unidade curricular de Algoritmos e Estruturas de Dados foi pedido o desenvolvimento de um projeto cujo objetivo foi o de planear e implementar um sistema que leia os ficheiros de mensagens e de palavras chave e teste todos os espaçamentos possíveis, de modo a encontrar palavras-chave em algumas das mensagens existentes. Assim, o output do projeto consistiu no 'ID' das mensagens que contêm alguma(s) das palavras-chave a procurar. Para além disto, foi necessário que a aplicação fosse eficiente de modo a encontrar uma variedade enorme de palavras-chave no menor espaço de tempo possível e ser 'user-friendly'.

# Capítulo 2

## Implementação

### 2.1 Motivação

A motivação para a realização deste trabalho foi de elaborar uma aplicação que pudesse 'combater' organizações terroristas ou outros grupos criminosos que podem causar danos severos à humanidade. Acho que esta aplicação, sem dúvida, poderia ser útil no dia-a-dia para as forças de combate ao terrorismo, facilitando a descoberta de palavras-chave importantes e assim obter vantagens no combate a inúmeras organizações terroristas.

### 2.2 A. Construção de um Analisador Léxico:

#### 2.2.1 A1: Implementar em *LEX* o algoritmo standard dum Analisador Léxico para reconhecer os *tokens* que compõem um endereço URL.

Neste exercício A1 era necessário criar um analisador léxico em *LEX* que reconhecesse os vários *tokens* que compõem um endereço URL. Para reconhecer os vários *tokens* o analisador precisa do carácter que o inicia.

1. **carácter '/'** - Inicia os *tokens* *USER* ou *PATH*.
2. **carácter ':'** - Inicia os *tokens* *PASSWORD* ou *PORT*.
3. **carácter '@'** - Inicia o *token* *HOST*.
4. **carácter ';' - Inicia o token FTPTYPE.**

---

```

1 CHARACTER [a-zA-Z0-9]
2 SMB [_$]
3 IDENTIFICADOR ({CHARACTER}[_]{$})(CHARACTER[_]{$}[-])*
4 USER [a-zA-Z]{CHARACTER}{3}{CHARACTER}*
5 PASS {SMB}{IDENTIFICADOR}{5}{IDENTIFICADOR}*|
    ↳ {IDENTIFICADOR}{SMB}{IDENTIFICADOR}{4}{IDENTIFICADOR}*|
    ↳ {IDENTIFICADOR}{2}{SMB}{IDENTIFICADOR}{3}{IDENTIFICADOR}*|
    ↳ {IDENTIFICADOR}{3}{SMB}{IDENTIFICADOR}{2}{IDENTIFICADOR}*|
    ↳ {IDENTIFICADOR}{4}{SMB}{IDENTIFICADOR}{IDENTIFICADOR}*|
    ↳ {IDENTIFICADOR}{5}{SMB}{IDENTIFICADOR}*|
    ↳ {IDENTIFICADOR}{5}{IDENTIFICADOR}+{SMB}{IDENTIFICADOR}*
6 HOST {IDENTIFICADOR}[_]{IDENTIFICADOR}[_]{IDENTIFICADOR}*
7 PORT [0-9]+
8 PATH {IDENTIFICADOR}[/]{IDENTIFICADOR}([/]{IDENTIFICADOR})*
9 FTPTYPE [#]{PORT}

```

---

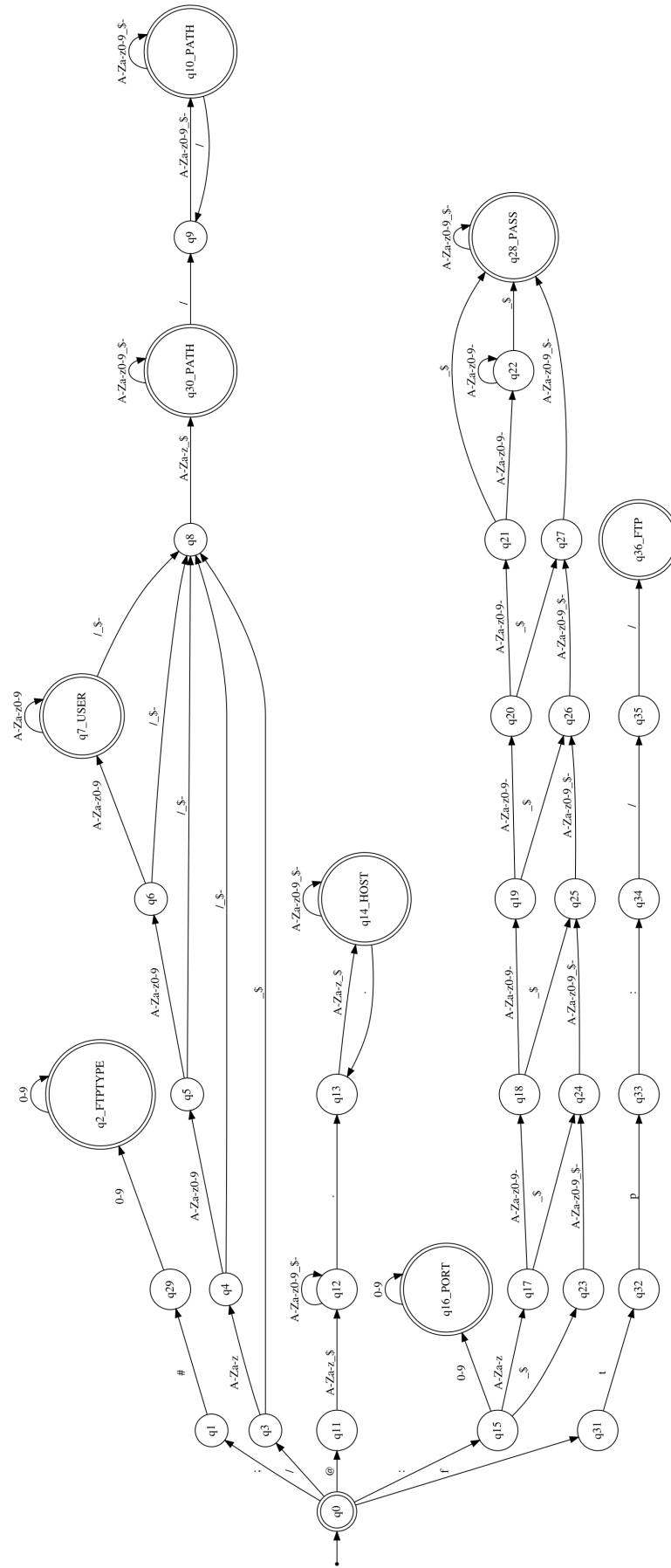
Listing 1: Código usado para definir os vários *tokens*.

Para além de reconhecer os vários *tokens* o analisador léxico em *LEX* também conta o numero total de *tokens* que encontrou.

Usando esta definições o analisador consegue detetar a maioria dos *tokens*, mas comete alguns erros. Um dos erros que o analisador apresenta é detetar *tokens PATH* em que um dos identificadores começa por um dígito, o que não devia acontecer.

### 2.2.2 A2:Desenhar o autómato para reconhecimento dos *tokens* que compõem os endereços URL.

Para reconhecer os diferentes *tokens* construímos um autómato finito não determinístico.

Figura 2.1: Autômato para reconhecimento de *tokens*



A partir desde autômato foi possível criar a tabela de transições usada para criar a matriz da função *delta* necessária ao funcionamento do analisador léxico em C.

A tabela de transições gerada é a seguinte.

Tabela 2.1: Tabela de transições

Estado	A-Za-z	0-9	-	_	@	:	;	/	#	.
0						11	15	1	3	
1										29
2		2								
3	4			8	8					
4	5	5	8	8	8			8		
5	6	6	8	8	8			8		
6	7	7	8	8	8			8		
7	7	7	8	8	8	-2	-2	8		
8	30			30	30					
9	10	10	10	10	10					
10	10	10	10	10	10			-2	9	
11	12			12	12					
12	12	12	12	12	12					13
13	14			14	14					
14	14	14	14	14	14		-2	-2		13
15	17	16		23	23					
16		16						-2		
17	18	18	18	24	24					
18	19	19	19	25	25					
19	20	20	20	26	26					
20	21	21	21	27	27					
21	22	22	22	28	28					
22	22	22	22	28	28					
23	24	24	24	24	24					
24	25	25	25	25	25					
25	26	26	26	26	26					
26	27	27	27	27	27					
27	28	28	28	28	28					
28	28	28	28	28	28	-2				
29		2								
30	30	30	30	30	30			9		

Os espaços em branco representam erros, enquanto que os espaços assinalados com -2" representam o início de um novo *token*. Podemos também ver assinalado a cor diferente os vários estados finais.

1. **2** - Estado final para o *Token FTPTYPE*.
2. **7** - Estado final para o *Token USER*.
3. **10 e 30** - Estados finais para o *Token PATH*.
4. **14** - Estado final para o *Token HOST*.
5. **16** - Estado final para o *Token PORT*.
6. **28** - Estado final para o *token PASSWORD*.

### 2.2.3 A3: Implementar em C o algoritmo standard dum Analisador Léxico com tratamento de erros para reconhecer os *tokens* que compõem um endereço URL.

O analisador léxico em C recolhe os *URLs* do ficheiro "*EnderecosURL.txt*" e analisa os vários *tokens* que encontra. Para o conseguir precisa da função *int delta(int q, char ch)* que recebe o estado atual e um carácter do ficheiro lido e com recurso à tabela de transições calcula o novo estado atual e devolve-o. Este novo estado é depois usado na função *int AnalisadorLexico (TOKEN \*TOK, int \*tam\_TOK)* que trata de reconhecer os *tokens* em si.

---

```
1  int AnalisadorLexico (TOKEN *TOK, int *tam_TOK) {
2
3  //(...)
4
5  switch (q_ant)
6  {
7      //FTPTYPE
8      case 2 : TOK[*tam_TOK].COD = 2; strcpy(TOK[*tam_TOK].VALOR, st);
9          ↪ (*tam_TOK)++; break;
10     //USER
11     case 7 : TOK[*tam_TOK].COD = 7; strcpy(TOK[*tam_TOK].VALOR, st);
12         ↪ (*tam_TOK)++; break;
13     //PATH
14     case 10 : TOK[*tam_TOK].COD = 10; strcpy(TOK[*tam_TOK].VALOR, st);
15         ↪ (*tam_TOK)++; break;
16     //PATH
17     case 30 : TOK[*tam_TOK].COD = 30; strcpy(TOK[*tam_TOK].VALOR, st);
18         ↪ (*tam_TOK)++; break;
19     //HOST
20     case 14 : TOK[*tam_TOK].COD = 14; strcpy(TOK[*tam_TOK].VALOR, st);
21         ↪ (*tam_TOK)++; break;
22     //PORT
23     case 16 : TOK[*tam_TOK].COD = 16; strcpy(TOK[*tam_TOK].VALOR, st);
24         ↪ (*tam_TOK)++; break;
25     //PASSWORD
26     case 28 : TOK[*tam_TOK].COD = 28; strcpy(TOK[*tam_TOK].VALOR, st);
27         ↪ (*tam_TOK)++; break;
28 }
29 }
30
31 //(...)
```

---

Listing 2: *Switch* usado para analisar e guardar os *tokens* reconhecidos e respetivos códigos.

Depois de correr a função *AnalisadorLexico()* a função *main()* pode mostrar no ecrã todos os *tokens* reconhecidos no ficheiro.

## 2.3 B – Construção de um Analisador Sintático

### 2.3.1 1 – Gramática independente de contexto geradora da linguagem

$$G = (\Sigma, T, P, S)$$

$$S = \{S, A, B, C, D, F, X, Y\}$$

$$P = \{$$

$$\cdot S \rightarrow ABCDFXY$$

$$\cdot A \rightarrow ftp : //$$

$$\cdot B \rightarrow userC@|\epsilon$$

$$\cdot C \rightarrow :password|\epsilon$$

$$\cdot D \rightarrow host$$

$$\cdot F \rightarrow :port|\epsilon$$

$$\cdot X \rightarrow /path$$

$$\cdot Y \rightarrow ;ftptype|\epsilon$$

$$\}$$

$$T = \{ftp, :, /, user, @, password, host, port, path, ;, , ftptype\}$$

### 2.3.2 2 – Tabela LL da Linguagem

Tabela 2.2: Tabela LL da linguagem

	ftp	:	/	user	@	pass	host	port	path	;	ftptype	\$
ftp	SKIP	E	E	E	E	E	E	E	E	E	E	E
:	E	SKIP	E	E	E	E	E	E	E	E	E	E
/	E	E	SKIP	E	E	E	E	E	E	E	E	E
user	E	E	E	SKIP	E	E	E	E	E	E	E	E
@	E	E	E	E	SKIP	E	E	E	E	E	E	E
password	E	E	E	E	E	SKIP	E	E	E	E	E	E
host	E	E	E	E	E	E	SKIP	E	E	E	E	E
port	E	E	E	E	E	E	E	SKIP	E	E	E	E
path	E	E	E	E	E	E	E	E	SKIP	E	E	E
;	E	E	E	E	E	E	E	E	E	SKIP	E	E
ftptype	E	E	E	E	E	E	E	E	E	E	SKIP	E
\$	E	E	E	E	E	E	E	E	E	E	E	AC
S	P1	E	E	E	E	E	E	E	E	E	E	E
A	P2	E	E	E	E	E	E	E	E	E	E	E
B	E	P4	E	P3	P4	E	P4	E	E	E	E	E
C	E	P5	E	E	E	E	P6	E	E	E	E	E
D	E	E	E	E	E	E	P7	E	E	E	E	E
F	E	P8	P9	E	E	E	E	E	E	E	E	E
X	E	E	P10	E	E	E	E	E	E	E	E	E
Y	E	E	P12	E	E	E	E	E	E	E	P11	E

Como podemos observar pela tabela e visto que não existem conflitos, trata-se de uma tabela do tipo LL.

### 2.3.3 Conjunto dos First's, Follow's e Lookaheads:

#### First's:

$$First(S) = \{ftp\}$$

$$First(A) = \{ftp\}$$

$$First(B) = \{user, \epsilon\}$$

$$First(C) = \{:, \epsilon\}$$

$$First(D) = \{host\}$$

$$First(F) = \{:, \epsilon\}$$

$$First(X) = \{/ \}$$

$$First(Y) = \{:, ftptype\}$$

#### Follow's:

$$Follow(S) = \{\$ \}$$

$$Follow(A) = \{user, :, host\}$$

$$Follow(B) = \{@, :, host\}$$

$$\begin{aligned}
Follow(C) &= \{host\} \\
First(D) &= \{:, /\} \\
First(F) &= \{ /\} \\
First(X) &= \{;, ftptype\} \\
First(Y) &= \{\$\}
\end{aligned}$$

**Calculo dos First's:**

$$\begin{aligned}
First(S \rightarrow ABCDFXY) &= \{ftp\} \subset First(S) \\
First(A \rightarrow ftp : /\) &: \{ftp\} \subset First(A) \\
First(B \rightarrow userC@) &: \{user\} \subset First(B) \\
First(B \rightarrow \epsilon) &: \{\epsilon\} \subset First(B) \\
First(C \rightarrow : password) &: \{:\} \subset First(C) \\
First(C \rightarrow \epsilon) &: \{\epsilon\} \subset First(C) \\
First(D \rightarrow host) &: \{host\} \subset First(D) \\
First(F \rightarrow : port) &: \{:\} \subset First(F) \\
First(F \rightarrow \epsilon) &: \{\epsilon\} \subset First(F) \\
First(X \rightarrow /path) &: \{ /\} \subset First(X) \\
First(Y \rightarrow ; ftptype) &: \{;\} \subset First(Y)
\end{aligned}$$

**Calculo dos Follow's:**

$$\begin{aligned}
Follow(B \rightarrow userC@) &: \{@\} \subset Follow(B) \\
Follow(S \rightarrow ABCDFXY) &: \\
First(BCDFXY) - \{\epsilon\} &\subset Follow(A) : \\
\cdot \{user\} &\subset Follow(A) \\
First(CDFXY) - \{\epsilon\} &\subset Follow(B) : \\
\cdot \{:, host\} &\subset Follow(B) \\
First(DFX Y) - \{\epsilon\} &\subset Follow(C) : \\
\cdot \{host\} &\subset Follow(C) \\
First(FXY) - \{\epsilon\} &\subset Follow(D) : \\
\cdot \{:, /\} &\subset Follow(D) \\
First(XY) - \{\epsilon\} &\subset Follow(F) : \\
\cdot \{ /\} &\subset Follow(F) \\
First(Y) - \{\epsilon\} &\subset Follow(X) : \\
\cdot \{;, ftptype\} &\subset Follow(X) \\
Follow(S) &\subset Follow(Y)
\end{aligned}$$

**Calculo dos Lookaheads:**

$$\begin{aligned}
L(S \rightarrow ABCDFXY) &: \{ftp\} \\
\cdot M[S, ftp] &= P1
\end{aligned}$$

$$L(A \rightarrow ftp : /\) : \{ftp\}$$

$$\cdot M[A, ftp] = P2$$

$$L(B \rightarrow userC@) : \{user\}$$

$$\cdot M[B, user] = P3$$

$$L(B \rightarrow \epsilon) : \{ @, :, host \}$$

$$\cdot M[B, @] = M[B, :] = M[B, host] = P4$$

$$L(C \rightarrow : password) : \{ : \}$$

$$\cdot M[C, :] = P5$$

$$L(C \rightarrow \epsilon) : \{ host \}$$

$$\cdot M[C, host] = P6$$

$$L(D \rightarrow host) : \{ host \}$$

$$\cdot M[D, host] = P7$$

$$L(F \rightarrow : port) : \{ host \}$$

$$\cdot M[F, :] = P8$$

$$L(F \rightarrow \epsilon) : \{ / \}$$

$$\cdot M[F, /] = P9$$

$$L(X \rightarrow /path) : \{ / \}$$

$$\cdot M[X, /] = P10$$

$$L(Y \rightarrow ; ftptype) : \{ ; \}$$

$$\cdot M[Y, ;] = P11$$

$$L(Y \rightarrow \epsilon) : \{ / \}$$

$$\cdot M[Y, /] = P12$$

$$L(S \rightarrow ABCDFXY) : \{ ftp \}$$

$$\cdot M[S, ftp] = P13$$

### 2.3.4 3 – Analisador Sintático em C

Após estar concluída a análise léxica dos *tokens* do ficheiro de URLs o programa continua a sua execução e é chamada a função do analisador sintático. Caso uma cadeia seja reconhecida a função devolve um valor diferente de -1 e é impressa

uma mensagem a dar a indicação do resultado do reconhecimento da frase.

---

```
1 if (AnalisadorSintatico(TOK, tam_TOK) == -1)
2     printf("ERRO: Sequencia de tokens nao aceite.\n");
3 else
4     printf("Sequencia de tokens aceite.\n");
5 }
```

---

Listing 3: Chamada da função do analisador sintático na função *int main()*

Na função do analisador sintático propriamente dita existem duas matrizes:

- **Prod[N][M]** – Matriz das produções da gramática – Possui **N** linhas que correspondem ao número total de produções da gramática e **M** colunas, valor que corresponde ao maior número de símbolos de todas as produções. A cada linha corresponde uma produção e em vez de introduzirmos na matriz os caracteres desta produção colocamos números, que correspondem aos caracteres da gramática. Caso a produção tenha um tamanho inferior a **M** preenchemos o resto dessa linha da matriz com -1.
- **Oraculo[N][M]** – Matriz Oráculo – Matriz que corresponde à tabela LL. Não existe nenhuma diferença significativa entre as duas.

Temos também nesta função uma pilha, contador de *tokens* e o carácter de lookahead que corresponde ao código do *token* que está a ser analisado.

Como no nosso caso o código do *token* não corresponde a um índice válido da tabela Oráculo temos também de fazer a conversão entre o código do *token* para o respetivo número do símbolo na matriz Oráculo.



---

```
1 printf("Topo(Z) -- lookahead -- accao \n");
2   Z[0] = 11; // Push(Z, $)
3   Z[1] = 12; // Push(Z, S)
4
5   topo = 1; //topo da pilha (indice da lista Z)
6   k = 0;    //nmro do token da lista
7   lookahead = TOK[k].COD;
8   switch (lookahead){
9       case 2 : //ftptype
10          lookahead = 10; break;
11       case 7 : //user
12          lookahead = 3; break;
13       case 10: //path
14          lookahead = 8; break;
15       case 14: //host
16          lookahead = 6; break;
17       case 16: //port
18          lookahead = 7; break;
19       case 28: //password
20          lookahead = 5; break;
21       case 30: //path
22          lookahead = 8; break;
23   }
```

---

Listing 4: Código de inicialização da pilha e conversão do código do token

Após a conclusão da inicialização da pilha é então feito o reconhecimento da cadeia de *tokens*, nesta parte do código é consultada a matriz Oráculo com o valor no topo da pilha e o número do símbolo do *lookahead* (sendo aqui novamente feita a conversão) sendo então removidos, introduzidos novos valores na pilha (quando é chamada a matriz das Produções) ou atingido um estado de aceitação da sequência de tokens.

---

```
1  do
2  {
3      accao = Oraculo[Z[topo]][lookhead];
4      printf(" %d -- %d -- %d \n", Z[topo], lookhead, accao);
5      switch (acao)
6      {
7          case -2 : break; // Aceitacao
8          case -1 : break; // ERRO
9          case 0 : topo--; // Pop(Z)
10             lookahead = TOK[++k].COD; // Da_Simbolo
11             switch (lookhead){
12                 case 2 :
13                     lookahead = 10; break;
14                 case 7 :
15                     lookahead = 3; break;
16                 case 10:
17                     lookahead = 8; break;
18                 case 14:
19                     lookahead = 6; break;
20                 case 16:
21                     lookahead = 7; break;
22                 case 28:
23                     lookahead = 5; break;
24                 case 30:
25                     lookahead = 8; break;
26             }
27             break;
28         default:
29             topo--; // Pop(Z);
30             for (j = 6; j >= 0; j--)
31                 if (Prod[acao][j] >= 0)
32                     Z[++topo] = Prod[acao][j]; // Push(Z, Yi)
33             break;
34     }
35 } while (acao >= 0);
```

---

Listing 5: Parte do código responsável pela consulta e movimentação dos estados da pilha

**Nota:** Apesar de pensarmos que este código se encontra funcional não conse-

guimos fazer o reconhecimento completo de frases devido à falta da receção do *token* correspondente ao FTP.

## 2.4 Conclusões

Ao longo deste capítulo explicámos de forma extensiva as opções tomadas na realização das duas principais partes do trabalho: o A (Construção de um analisador léxico) e o B (Construção de um analisador sintático). Concluimos então que apesar de na nossa opinião o código das funções se encontrar maioritariamente bem implementado o nosso trabalho possui alguns erros que o impedem de funcionar de acordo com o que nos foi pedido.

## Capítulo 3

# Reflexão crítica e problemas encontrados

### 3.1 Objetivos Propostos vs Alcançados

Tivemos alguns problemas que não ficaram resolvidos e que se traduziram em erros no programa final, tais como:

- **Grupo A:**

Antes de poder ser reconhecido algum token, é preciso ler o seu identificador, isto é, o caráter que o antecede.

Para reconhecer um 'host' é preciso ler um '@', contudo isto não devia ser o caso pois só é necessário ler um '@' se houver o token 'user' que é opcional ao contrario do 'host' que é obrigatório;

Apenas os 'paths' que não têm nenhum identificador começado por dígitos deviam ser validos, mas o programa também pode aceitar 'paths' que não respeitem esta regra;

O analisador léxico não considera o 'ftp://';

- **Grupo B:**

Reconhecimento da palavra 'ftp://' logo no inicio da frase. Uma vez que a nossa função de análise léxica não gera o token correspondente a esta parte a função de análise sintática não consegue funcionar de forma correta. Apesar disto e com base nos nossos testes e análise do código desta função pensamos que ela se encontra correta.

Contudo fomos bem sucedidos em varias tarefas.

- O analisador léxico, tanto em flex como em C é capaz de reconhecer corretamente a maioria dos tokens analisados.
- O autómato apenas reconhece como passwords identificadores com pelos menos um '\_' ou um '\$', o que foi um grande desafio.

## **3.2 Divisão de Tarefas**

Relativamente à divisão de tarefas, decidimos desde o início dividir os grupos por cada elemento do grupo, ou seja o João Dinis ficou encarregue do Grupo A então o João Saraiva e Luís Rodrigues ficaram encarregues do Grupo B. O relatório foi feito em conjunto com todos os elementos do grupo.

# Capítulo 4

## Conclusões

### 4.1 Conclusões Principais

Com a realização deste projeto deparámo-nos com diversos desafios, mais concretamente:

- **Grupo A:**
  - Obter todos os estados do autómato
  - Reconhecer as palavras-passe com um 'underscore' e com um '\$'
  - Os 'path' não poderem ser iniciados com um dígito
- **Grupo B:**
  - Elaborar as produções corretas para a gramática G
  - Conseguir a inicialização da frase, relativamente ao 'ftp'

Para resolver estes problemas tivemos de analisar e implementar várias funcionalidades que foram discutidas e analisadas em detalhe neste relatório. Estamos satisfeitos com o trabalho realizado apesar de não termos conseguido cumprir na totalidade os objetivos que nos foram propostos.

Foi também uma boa forma de aplicar alguns dos conhecimentos adquiridos ao longo da unidade curricular e de trabalhar com uma linguagem de programação com a qual não possuíamos muita prática.