

# COMPILADORES

Prof. Me. Stéfano Schwenck

## Introdução ao Flex

### 1.1 Analisadores léxicos

Um analisador léxico (*scanner*) é um programa que permite ler os caracteres de um ficheiro de texto (*e.g.*, programa-fonte) e produzir uma sequência de componentes léxicos (*tokens*) que serão utilizados pelo analisador sintático (*parser*) e/ou identificar erros léxicos na entrada. Além de sua função básica, o analisador léxico em geral está encarregue de fazer algumas tarefas secundárias, designadamente, a eliminação de comentários, espaços em branco e “tabulações”.

Um *token* representa um conjunto de cadeias de entrada possível e por sua vez, um lexema é uma determinada cadeia de entrada associada a um *token*. Considere os exemplos apresentados na tabela 1.1.

O FLEX é uma ferramenta que permite gerar analisadores léxicos. Estes analisadores são capazes de reconhecer padrões léxicos em texto (*e.g.*, números, identificadores e palavras-chave de uma determinada linguagem de programação). O analisador é construído com base num conjunto de regras. Uma regra é constituída por um par, padrão-ação, o padrão (expressão regular) descreve o elemento a reconhecer e acção (ou conjunto de acções) define o procedimento que será realizado no caso de identificação positiva do padrão.

Tabela 1.1: *Tokens* e *lexemas*

Tokens	Lexemas
FOR	for
IF	if
WHILE	while
NÚMERO	1089, 142857, 0.2, 3.14159
IDENTIFICADOR	i, j, contador, nomeAluno
OP_SOMA	+
OP_MAIOR_IGUAL	>=
ABRE_PAR	(

## 1.2 Modo de utilização do FLEX

O ciclo de vida de um programa FLEX obedece à estrutura apresentada na figura 1.1.

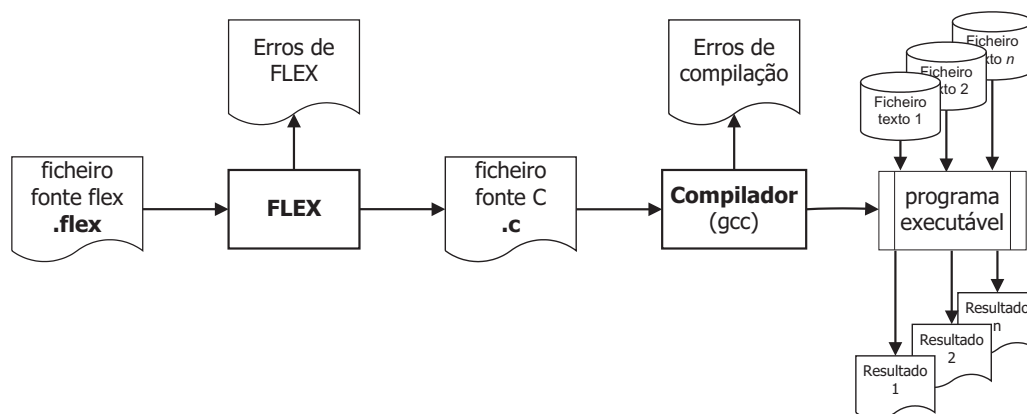


Figura 1.1: Ciclo de vida de um programa FLEX

Com base num ficheiro fonte escrito de acordo com a sintaxe do FLEX, o programa FLEX gerará um analisador léxico descrito na linguagem C. Em caso de existirem erros de codificação, o FLEX gerará uma listagem de erros. O ficheiro fonte em C terá de ser compilado para a plataforma em utilização utilizando um compilador da linguagem C adequado (neste caso o GCC). O resultado final da compilação será um programa executável capaz de identificar os padrões definidos pelo programado e levar o conjunto de acções previsto. Como entrada para o analisador gerado podem ser fornecidos ficheiros de texto ou alternativamente fornecer os dados directamente pelo *standard* de entrada.

No exemplo seguinte são apresentados os passos necessários à compilação e utilização de um programa FLEX. Considere-se a existência de um ficheiro `ficheiro.flex` com o programa FLEX já escrito.

```
> flex ficheiro.l  
> gcc lex.yy.c -lfl
```

O comando `flex` gera, por omissão, um ficheiro com o nome `lex.yy.c` que deverá ser compilado, por exemplo com o `gcc`. Na utilização do `gcc` é necessário indicar a utilização da biblioteca FLEX adicionando o parâmetro `-lfl`. Por sua vez, o compilador de C gera, por omissão, um ficheiro com o nome `a.out`. Por último, para a execução deste programa basta a evocação do seu nome na linha de comandos. Neste caso, a introdução dos dados terá de ser realizada via consola (terminando obrigatoriamente com `Ctrl+D`).

```
> gcc lex.yy.c -o Programa -lfl  
> Programa < Dados.txt
```

Neste exemplo, o comando `flex` gera a partir do ficheiro `ficheiro.l`, o ficheiro com o nome `lex.yy.c` que deverá ser compilado. Nesta utilização apresentada do `gcc`, é indicado o nome do executável a ser gerado, neste caso, `Programa`. Na execução do `Programa`, a introdução dos dados é realizada a partir do ficheiro `Dados.txt`.

## 1.3 Formato de um ficheiro FLEX

Um programa em FLEX é constituído por três secções, a saber, declarações, regras e rotinas auxiliares. A separação entre as secções é feita inserindo uma linha com o símbolo “%%”.

Considere-se o seguinte exemplo que será discutido nas secções seguintes.

```
1  %{  
2      int numChars=0;  
3  %}  
4  
5  %%  
6  
7  . {  
8      numChars++;  
9      printf("%s",yytext);  
10 }  
11  
12 \n {  
13     numChars++;  
14     printf("\n");  
15 }
```

```

16
17 %%
18
19 main()
20 {
21     yylex();
22     printf("Número_de_caracteres_%d\n", numChars);
23
24 }

```

### 1.3.1 Declarações

Esta secção compreende duas partes:

- **Instruções C** – nesta parte, delimitada pelos símbolos “%{” e “%}”, são colocadas as instruções da linguagem C que posteriormente serão incluídas no início do ficheiro C a gerar pelo FLEX. Os exemplos mais comuns são a inclusão de ficheiros de cabeçalhos ( *headers*, *.h*), declarações de variáveis e constantes.

```

1 /* Definição da variável numChars */
2 %{
3     int numChars=0;
4 %}

```

- **Expressões regulares** – nesta parte, podem ser declaradas macros para as expressões regulares mais comuns como por exemplo algarismo ou letra do alfabeto.

```

1 /* Definição de macros */
2 ALGARISMO [0-9] /* Algarismo */
3 ALFA [a-zA-Z] /* Letra do alfabeto */

```

A utilização de macros para expressões regulares será explicada com detalhe mais adiante.

### 1.3.2 Regras (definição de padrões e acções)

Nesta secção são definidas as expressões regulares (padrões) e as respectivas acções que se pretendem realizar no caso da identificação positiva (*pattern matching*) do referido padrão.

No caso de um qualquer carácter excepto mudança de linha (representado por “.”) é incrementada a variável `num_chars` e impresso o referido carácter no *standard* de saída. A mudança de linha (representado por “\n”) é também contabilizada como um carácter e escrita no *standard* de saída.

As expressões regulares têm de ser **obrigatoriamente** escritas na primeira coluna do ficheiro.

```
1  .  {
2      numChars++;
3      printf ("%s", yytext );
4  }
5
6  \n  {
7      numChars++;
8      printf ("\n");
9  }
```

Na secção 1.8 são apresentados alguns dos padrões mais relevantes utilizados pelo FLEX.

o analisador léxico gerado funciona de acordo com as seguintes regras:

- Apenas uma regra é aplicada à entrada de dados;
- A acção executada corresponde à expressão que consome o maior número de caracteres;
- Caso existam duas ou mais expressões que consumam igual número de caracteres, tem precedência a regra que aparece em primeiro no ficheiro.

Quando um padrão é reconhecido, a sequência de caracteres consumida (*token*) na identificação do padrão é guardada na variável `yytext` (do tipo `char *`). Para além disso, o comprimento da referida sequência é guardado na variável `yylen`<sup>1</sup> (do tipo `int`).

### 1.3.3 Rotinas em C de suporte

Nesta secção pode ser escrito o código C que se pretende que seja adicionado ao programa a gerar pelo FLEX. Tipicamente este código inclui o corpo do programa, designadamente, a função `main()` da linguagem C.

```
1 main()
2 {
3     yylex();
4     printf("Número_de_caracteres_%d\n", numChars);
5 }
```

A função `yylex()` evoca o analisador léxico gerado pelo flex que processará as expressões regulares anteriormente descritas (ver secção 1.3.2).

---

<sup>1</sup>O valor desta variável poderia ser obtido através da instrução da linguagem C `strlen(yytext)`

## 1.4 Exemplo mais elaborado

Considere o seguinte exemplo, no qual é contabilizado a quantidade de números e de linhas existentes no ficheiro. Neste exemplo recorre-se à utilização de uma macro para a definição de algarismo.

```

1  %{
2  int qtdNumeros=0, nLinhas=0;
3  %}
4
5  ALGARISMO    [0-9]
6
7  %%
8
9  /* Se a acção for descrita numa só linha
10     as chavetas podem ser omitidas */
11
12  \n          nLinhas++;
13  {ALGARISMO}+ {printf("d_%s\n",yytext); qtdNumeros++;}
14  .
15
16  %%
17  main()
18  {
19      yylex();
20      printf("#_linhas=%d\n", nLinhas);
21      printf("#_numeros=%d\n", qtdNumeros);
22  }
```

Todos os caracteres não processados pelas duas primeiras expressões regulares são consumidos pela última à qual não corresponde nenhuma acção particular.

## 1.5 Propostas de exercícios

- a) Escrever um programa que permite contar o número de ocorrências de uma cadeia de caracteres;
- b) Escrever um programa que permite substituir uma cadeia de caracteres por outra;
- c) Escrever um programa que dado um ficheiro de texto, mostra:
  - número de algarismos;
  - número de letras do alfabeto;

- número de linha de texto;
  - número de espaços ou tabulações (\t);
  - número de caracteres não identificados nos pontos anteriores;
- d) Escrever um programa que permite identificar números naturais;

Entrada	Saída
123 abc 12.45 s 245 xyz xyz 2 abc 45 cc	123 12 45 245 2 45

- e) Escrever um programa que permite identificar números inteiros (com ou sem sinal);
- f) Escrever um programa que permite identificar números com parte decimal (com ou sem sinal);

## 1.6 Ambiente de trabalho

O acesso a uma máquina LINUX pode ser realizado utilizando o programa `putty` que está instalado em `c:putty` em modo SSH (ver figura 1.2). As máquinas a utilizar deverão ser `ssh`, `ssh1` e `ssh2`.

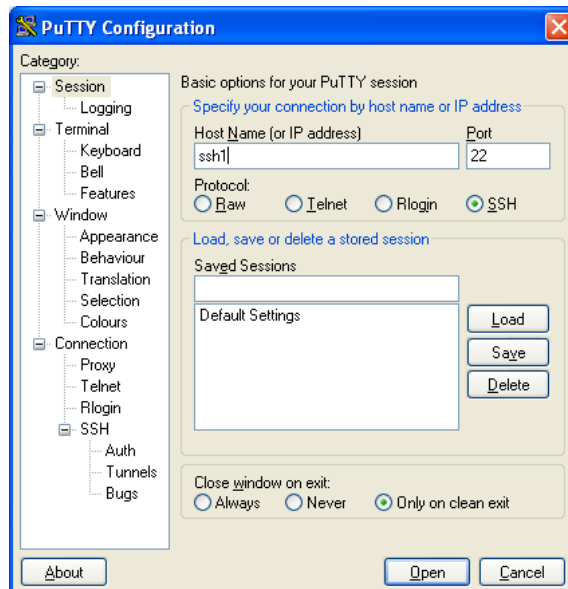


Figura 1.2: Acesso SSH via `putty`

A edição dos ficheiros fonte pode ser realizada a partir de qualquer editor de texto básico (*e.g.*, no ambiente Windows existe o Programmer's Notepad) desde que os ficheiros sejam gravados em formato Unix.

## 1.7 Resumo de comandos UNIX úteis

Na tabela 1.2 são apresentados os comandos que permitem fazer mudança de directório e mostrar qual o directório actual.

Tabela 1.2: Mudar e mostrar directório

Comando	Descrição
<code>cd</code>	Mudar para o directório <code>home</code>
<code>cd ..</code>	Mudar para o directório pai
<code>cd &lt;DIR&gt;</code>	Mudar para o directório DIR
<code>pwd</code>	Mostra o <i>path</i> do directório actual

Na tabela 1.3 são apresentados os comandos que permitem listar o conteúdo de um determinado directório.

Tabela 1.3: Listagem do conteúdo de um directório

Comando	Descrição
<code>ls</code>	Mostra conteúdo do directório actual
<code>ls -a</code>	<i>idem</i> incluindo ficheiros escondidos
<code>ls -la</code>	<i>idem</i> incluindo ficheiros escondidos sob a forma de lista
<code>ls &lt;DIR&gt; -l</code>	Mostra conteúdo do directório DIR sob a forma de lista

Na tabela 1.4 são apresentados os comandos que permitem alterar as permissões de acesso a ficheiros e/ou directórios. As permissões podem ser de três tipos:

- **Leitura (r)** – permite visualizar o conteúdo quando de se trate de ficheiros, no caso de directórios, permite fazer um `ls`;
- **Escrita (w)** – permite alterar o conteúdo quando de se trate de ficheiros e no caso de directórios permite criar ficheiros/directórios no referido directório;
- **Execução (x)** – permite executar um programa, quando de se trate de ficheiros, no caso de directórios, permite aceder aos seus conteúdos;

As permissões de acesso a um ficheiro ou directório estão subdivididas por três grupos de utilizadores:



Tabela 1.4: Alteração de permissões de acesso de ficheiros/directórios

Comando	Descrição
<code>chmod u+rwx,go-rw &lt;FICH&gt;</code>	Concede as permissões <b>rw</b> x para o utilizador; retira as permissões <b>rw</b> para utilizadores do mesmo grupo e outros; as outras permissões não são alteradas
<code>chmod u+rwx,go+rx &lt;DIR&gt;</code>	Concede as permissões <b>rw</b> x para o utilizador; concede as permissões <b>rx</b> para utilizadores do mesmo grupo e outros; as outras permissões não são alteradas
<code>chmod 751 &lt;DIR&gt;</code>	Concede <b>somente</b> as permissões: <b>rw</b> x ( $7 = 4 + 2 + 1$ ) para o utilizador, <b>rx</b> ( $5 = 4 + 1$ ) para utilizadores do mesmo grupo e <b>x</b> ( $1$ ) para outros utilizadores

- o próprio (*owner*);
- o grupo (*group*);
- restantes (*other*).

Considere o seguinte extracto resultado da execução do comando `ls -la` no qual o primeiro campo faz a codificação das permissões leitura, escrita e execução (**rw**x) para cada grupo. Estas permissões podem ser convertidas para um valor numérico somando 4 para **r**, 2 para **w** e 1 para **x**.

```
-rwxr----- 1 user   profs    25 Jan 27  2002 fich
drwx--x---  4 user   profs   4096 Jan 17 16:31 dir/
```

Neste exemplo, o ficheiro **fich** tem permissões de leitura, escrita e execução para o próprio, leitura para o grupo e nenhuma para os restantes utilizadores. Por sua vez, o directório **dir** (o primeiro carácter é um **d** no caso de directórios) tem permissões de leitura, escrita e execução para o próprio, execução para o grupo e nenhuma para os restantes utilizadores.

## 1.8 Padrões utilizados no FLEX

Na tabela 1.5 são apresentados alguns dos padrões mais relevantes utilizados pelo FLEX.

Tabela 1.5: Padrões utilizados no FLEX

Padrão	Descrição
x	O carácter “x”
.	Qualquer carácter excepto mudança de linha
\n	Mudança de linha
[xyz]	Um dos caracteres “x”, “y” ou “z”
xyz	A cadeia de caracteres “xyz”
[a-zA-Z]	Um dos caracteres no intervalo de “a” a “z” ou de “A” a “Z”
[-+*/]	Qualquer um dos operadores “-”, “+”, “*” ou “/”, sendo que o símbolo “-” tem de aparecer em primeiro lugar dada a possibilidade de ambiguidade com a definição de intervalo
[abj-oZ]	Um dos caracteres “a”, “b” ou de “j” a “o” ou “Z”
[^A-Z\n]	Qualquer carácter excepto no intervalo de “A” a “Z” ou mudança de linha
r*	O carácter “r” zero ou mais vezes
r+	O carácter “r” uma ou mais vezes
r?	O carácter “r” zero ou uma vez
r{2,5}	O carácter “r” repetido de duas a cinco vezes
r{2,}	O carácter “r” repetido pelo menos duas vezes
r{4}	O carácter “r” repetido exactamente quatro vezes
{macro}	Substituição/Expansão da macro definida anteriormente
(r)	O carácter “r”, sendo que os parêntesis permitem estipular precedências
xyz*	A sequência “xy” seguida de zero ou mais “z”s
(xyz)*	A sequência “xyz” repetida zero ou mais vezes
r s	O carácter “r” ou “s” (alternativa)
^r	O carácter “r” apenas se no início da linha
r\$	O carácter “r” apenas se no final da linha (não consome o \n)
^xyz\$	Uma linha que contém apenas a cadeia de caracteres “xyz”
<<EOF>>	Fim de ficheiro