Capítulo 11

Ficheiros

'They're putting down their names,' the Gryphon whispered in reply, 'for fear they forget them before the end of the trial.'

Lewis Carroll, *Alice's Adventures in Wonderland*

Os programas que temos desenvolvido até aqui têm uma única fonte de dados, o teclado, e um único destino para os seus resultados, o ecrã. Para além disso, quando o programa termina, os dados usados e produzidos pelo programa desaparecem.

A maior parte dos programas desenvolvidos na vida real têm múltiplas fontes de dados e múltiplos destinos para os seus resultados. Muitos dos dados dos programas utilizados na vida real são *persistentes*, no sentido em que eles existem, independentemente da execução do programa, armazenados no disco do computador, num local existente na "núvem", numa memória USB ou num CD. A estrutura tipicamente utilizada para armazenar esta informação é chamada um ficheiro.

Um *ficheiro*[1] é um tipo estruturado de dados constituído por uma sequência de elementos, todos do mesmo tipo. Nos ficheiros que consideramos neste capítulo, os *ficheiros sequenciais*, os elementos são acedidos sequencialmente, ou seja, para aceder ao n-ésimo elemento do ficheiro, teremos primeiro de aceder aos $n-1$ elementos que se encontram antes dele.

[1] Em inglês, "file".

Os ficheiros diferem dos outros objetos computacionais considerados até aqui em dois aspetos:

1. Os seus valores poderem existir independentemente de qualquer programa, um ficheiro pode existir antes do início da execução de um programa e manter a sua existência após o fim da sua execução;
2. Um ficheiro encontra-se, em qualquer instante, num de dois estados possíveis, ou está a ser utilizado para a entrada de dados (estão a ser lidos valores do ficheiro, caso em que se diz que o ficheiro se encontra em *modo de leitura*) ou está a ser utilizado para a saída de dados (estão a ser escritos valores no ficheiro, caso em que se diz que o ficheiro se encontra em *modo de escrita*).

Ao utilizarmos um ficheiro, temos de dizer ao Python em que modo queremos que esse ficheiro se encontre e qual a localização física dos ficheiros em que os dados se encontram.

11.1 O tipo ficheiro

Sendo os ficheiros entidades que existem fora do nosso programa, antes de utilizar um ficheiro é necessário identificar qual a localização física deste e o modo como o queremos utilizar, ou seja se queremos ler a informação contida no ficheiro ou se queremos escrever informação no ficheiro.

A operação através da qual identificamos a localização do ficheiro e o modo como o queremos utilizar é conhecida por *operação de abertura do ficheiro* e é realizada em Python recorrendo à função embutida `open`. A função `open` tem a seguinte sintaxe:

`open(`⟨expressão⟩`,` ⟨modo⟩{`, encoding =` ⟨tipo⟩}`)`

Esta função tem dois argumentos obrigatórios e um opcional:

- o primeiro argumento, representado por ⟨expressão⟩, é uma expressão cujo valor é uma cadeia de carateres que corresponde ao nome externo do ficheiro;

- o segundo argumento, representado por ⟨modo⟩, é uma expressão cujo valor é uma das cadeias de carateres `'r'`, `'w'` ou `'a'`[2]. Ou seja,

  ⟨modo⟩ ::= `'r'` | `'w'` | `'a'`

  significando a primeira alternativa que o ficheiro é aberto para leitura[3], a segunda alternativa que o ficheiro é aberto para escrita a partir do início do ficheiro[4] e a terceira alternativa que o ficheiro é aberto para escrita a partir do fim do ficheiro[5];

- o terceiro argumento, o qual é opcional, é da forma `encoding =` ⟨tipo⟩, em que tipo é uma expressão que representa o tipo de codificação de carateres utilizado no ficheiro.

O valor da função `open` corresponde à entidade no programa que está associada ao ficheiro.

11.2 Leitura de ficheiros

Ao abrir um ficheiro para leitura, está subentendido que esse ficheiro existe, pois queremos ler a informação que ele contém. Isto significa que se o Python for instruído para abrir para leitura um ficheiro que não existe, irá gerar um erro como mostra a seguinte interação:

```
>>> f = open('nada', 'r')
builtins.IOError: [Errno 2] No such file or directory: 'nada'
```

Suponhamos que a diretoria (ou pasta) do nosso computador que contém os ficheiros utilizados pelo Python continha um ficheiro cujo nome é `teste.txt` e cujo o conteúdo corresponde ao seguinte texto:

```
Este é um teste
que mostra como o Python
lê ficheiros de carateres
```

[2] Existem outras alternativas que não são tratadas neste livro.
[3] `'r'` é a primeira letra da palavra inglesa "read" (lê).
[4] `'w'` é a primeira letra da palavra inglesa "write" (escreve).
[5] `'a'` é a primeira letra da palavra inglesa "append" (junta).

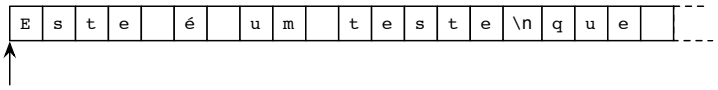

Figura 11.1: Representação do ficheiro `teste.txt`.

A execução pelo Python da instrução

```
t = open('teste.txt', 'r', encoding = 'UTF-16')
```

cria uma variável no nosso programa cujo nome é `t`, variável essa que está associada ao ficheiro `teste.txt`, o qual pode ser lido pelo Python, sabendo o Python que o texto neste ficheiro está codificado usando o código `UTF-16`.

Ao ler informação de um ficheiro, o Python mantém um indicador, o *indicador de leitura*, que indica qual o próximo elemento a ser lido do ficheiro (este indicador é representado nas nossas figuras por uma seta colocada imediatamente por baixo da posição em que se encontra no ficheiro). O indicador de leitura é colocado no início do ficheiro quando o ficheiro é aberto para leitura e movimenta-se no sentido do início para o fim do ficheiro sempre que se efetua uma leitura, sendo colocado imediatamente após o último símbolo lido, de cada vez que a leitura é feita.

Na Figura 11.1 mostramos parte do conteúdo do ficheiro `teste.txt`. Cada um dos elementos deste ficheiro é um caráter, e está representado na figura dentro de um quadrado. Este ficheiro corresponde a uma sequência de carateres e contém carateres que não são por nós visíveis quando o inspecionamos num ecrã e que indicam o fim de cada uma das linhas. Apresentámos na Tabela 2.8 alguns destes carateres, sendo o fim de linha representado por `\n`. Assim, no ficheiro, imediatamente após a cadeia de carateres `'Este é um teste'`, surge o caráter `\n` que corresponde ao fim da primeira linha do ficheiro.

A partir do momento que é criada uma variável, que designaremos por ⟨fich⟩, associada a um ficheiro aberto para leitura, passam a existir quatro novas funções no nosso programa para efetuar operações sobre esse ficheiro[6]:

[6]Novamente, surge-nos aqui uma situação semelhante à que apareceu na função `s.lower` apresentada na página 243, em que o nome das funções correspondem a um ⟨nome composto⟩ em que um dos seus constituintes é o nome de uma variável. Este aspeto é abordado na Secção 12.5.

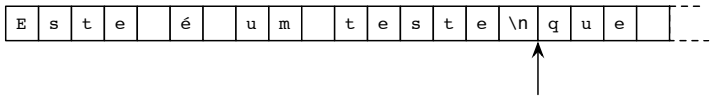

Figura 11.2: Ficheiro `teste.txt` após a execução de `t.readline()`.

1. ⟨fich⟩`.readline()`. Esta função lê a linha do ficheiro ⟨fich⟩ que se encontra imediatamente a seguir ao indicador de leitura, tendo como valor a cadeia de carateres correspondente à linha que foi lida. Se o indicador de leitura se encontrar no fim do ficheiro, esta função tem o valor `''` (a cadeia de carateres vazia). No nosso exemplo, a função que lê uma linha do ficheiro corresponde a `t.readline()`;

2. ⟨fich⟩`.readlines()`. Esta função lê todos os carateres do ficheiro ⟨fich⟩ que se encontram depois do indicador de leitura, tendo como valor uma lista em que cada um dos elementos é a cadeia de carateres correspondente a cada uma das linhas que foi lida. Se o indicador de leitura se encontrar no fim do ficheiro, esta função tem o valor `[]` (a lista vazia). No nosso exemplo, esta função corresponde a `t.readlines()`;

3. ⟨fich⟩`.read()`. Esta função lê todos os carateres do ficheiro ⟨fich⟩ que se encontram depois do indicador de leitura, tendo como valor uma cadeia de carateres contendo todos os carateres lidos. Se o indicador de leitura se encontrar no fim do ficheiro, esta função tem o valor `''` (a cadeia de carateres vazia). No nosso exemplo, esta função corresponde a `t.read()`;

4. ⟨fich⟩`.close()`. Esta função fecha o ficheiro ⟨fich⟩. A *operação de fecho de um ficheiro* corresponde a desfazer a ligação entre o programa e o ficheiro. No nosso exemplo, a função que fecha o ficheiro corresponde a `t.close()`.

Voltando ao nosso exemplo, suponhamos que abrimos o ficheiro `teste.txt` com a instrução `t = open('teste.txt', 'r', encoding = 'UTF-16')`. Se executarmos a instrução `ln1 = t.readline()`, a variável `ln1` passa a ter como valor a cadeia de carateres `'Este é um teste\n'`, ficando o indicador de leitura como se mostra na Figura 11.2. Repare-se que o caráter de fim de linha faz parte da cadeia de carateres lida.

A seguinte interação mostra a utilização da função `t.readline()`. É importante notar que ao atingir o fim do ficheiro, a função `t.readline()` tem como

valor a cadeia de carateres vazia, ''. Este aspeto é importante quando o nosso programa está a ler um ficheiro e necessita de determinar quando a leitura chega ao fim.

```
>>> t = open('teste.txt', 'r', encoding = 'UTF-16')
>>> ln1 = t.readline()
>>> ln1
'Este é um teste\n'
>>> ln2 = t.readline()
>>> ln2
'que mostra como o Python\n'
>>> ln3 = t.readline()
>>> ln3
'lê ficheiros de carateres\n'
>>> ln4 = t.readline()
>>> ln4
''
>>> t.close()
```

A seguinte interação mostra a utilização das funções disponíveis para ler ficheiros, indicando o comportamento de cada uma delas ao serem avaliadas com situações diferentes relativas ao indicador de leitura.

```
>>> f = open('teste.txt', 'r', encoding = 'UTF-16')
>>> l1 = f.readline()
>>> l1
'Este é um teste\n'
>>> l2 = f.read()
>>> l2
'que mostra como o Python\nlê ficheiros de carateres\n'
>>> print(l2)
que mostra como o Python
lê ficheiros de carateres

>>> f.close()
>>> g = open('teste.txt', 'r', encoding = 'UTF-16')
>>> lines = g.readlines()
```

```
>>> lines
['Este é um teste\n', 'que mostra como o Python\n',
'lê ficheiros de carateres\n']
```

Vamos agora analisar de uma forma mais detalhada os argumentos da função `open`. O nome do ficheiro utilizado por esta função corresponde à especificação completa do nome do ficheiro a utilizar. No exemplo que apresentámos, o ficheiro `teste.txt` existia na diretoria (ou pasta) do nosso computador que é utilizada por omissão pelo Python. Contudo, se este ficheiro não existisse na diretoria de omissão, mas sim numa diretoria chamada `exemplos` localizada na diretoria de omissão do Python, o nome do ficheiro a especificar na função `open` seria `'exemplos/teste.txt'`. Ou seja, o primeiro argumento da função `open` não é o nome de um ficheiro, mas sim a combinação de um caminho de diretorias e do nome de um ficheiro. Embora diferentes sistemas operativos usem símbolos diferentes para especificar caminhos em diretorias (o Mac OS e o Linux usam o símbolo `/`, ao passo que o Windows usa o símbolo \), na função `open` é sempre utilizado o símbolo `/`.

O argumento da função `open` associado à codificação de carateres é ligeiramente mais complicado. Sabemos que os carateres correspondem a símbolos e que estes são representados internamente em Python utilizando o *Unicode* (este aspeto foi discutido na Secção 4.3). Internamente ao Python, uma cadeia de carateres é uma sequência de zeros e uns correspondente a representações de carateres usando o *Unicode*. Contudo, um ficheiro existente no disco não é uma sequência de símbolos codificados em *Unicode*, mas apenas uma sequência de zeros e uns. Ao ler um ficheiro de texto existente num disco, o Python precisa de saber como "interpretar" a sequência de zeros e uns nele contida. Com base nesta informação, o Python descodifica a sequência de zeros e uns existente no disco, devolvendo uma sequência de carateres em *Unicode*, que é identificada como uma cadeia de carateres.

Para tornar as coisas ainda mais complicadas, não só a codificação de informação em disco é dependente do tipo de computador, mas também, dentro do mesmo computador existem normalmente ficheiros com diferentes tipos de codificação. Os sistemas operativos lidam com esta diversidade de codificações porque cada ficheiro contém informação sobre a codificação que este utiliza, informação essa que não está acessível ao Python. Por estas razões, ao abrir um ficheiro, é necessário dizer ao Python qual o tipo de codificação utilizada.

11.3 Escrita em ficheiros

De modo a escrevermos informação num ficheiro, teremos primeiro que efetuar a abertura do ficheiro com um dos modos `'w'` ou `'a'`. Ao abrir um ficheiro para escrita, se o ficheiro não existir, ele é criado pelo Python como um ficheiro sem elementos. Tal como no caso da leitura em ficheiros, ao escrever informação num ficheiro, o Python mantém um indicador, o *indicador de escrita*, que indica qual a posição do próximo elemento a ser escrito no ficheiro. Consoante o modo escolhido para a abertura do ficheiro, o Python coloca o indicador de escrita ou no início do ficheiro (ou seja, o ficheiro fica sem quaisquer elementos, e o seu antigo conteúdo, se existir, é perdido), se for utilizado o modo `'w'`, ou no fim do ficheiro, se for utilizado o modo `'a'`.

De um modo semelhante ao que acontece quando abrimos um ficheiro em modo de leitura, a partir do momento que é criada uma variável, que designaremos por ⟨fich⟩, associada a um ficheiro aberto para escrita, passam a existir as seguintes funções no nosso programa para efetuar operações sobre esse ficheiro:

1. ⟨fich⟩`.write(`⟨cadeia de carateres⟩`)`. Esta função escreve, a partir da posição do indicador de escrita, a ⟨cadeia de carateres⟩ no ficheiro ⟨fich⟩. O indicador de escrita é movimentado para a posição imediatamente a seguir à cadeia de carateres escrita. Esta função devolve o número de carateres escritos no ficheiro;

2. ⟨fich⟩`.writelines(`⟨sequência⟩`)`, na qual ⟨sequência⟩ é um tuplo ou uma lista cujos elementos são cadeias de carateres. Esta função escreve, a partir da posição do indicador de escrita, cada um dos elementos da ⟨sequência⟩ no ficheiro ⟨fich⟩, não escrevendo o caráter de fim de linha entre os elementos escritos. O indicador de escrita é movimentado para a posição imediatamente a seguir à última cadeia de carateres escrita. Esta função não devolve nenhum valor;

3. ⟨fich⟩`.close()`. Esta função fecha o ficheiro ⟨fich⟩.

A seguinte interação mostra a utilização de operações de escrita e de leitura num ficheiro. Poderá parecer estranho a não utilização da indicação sobre a codificação dos carateres utilizada no ficheiro. Contudo, como os ficheiros utilizados são criados pelo Python, a não indicação da codificação significa que esta será a codificação usada por omissão pelo Python.

```
>>> f1 = open('teste1', 'w')
>>> f1.write('abc')
3
>>> f1.write('def')
3
>>> f1.close()
>>> f1 = open('teste1', 'r') # 'teste1' é aberto para leitura
>>> cont = f1.read()
>>> print(cont)
>>> abcdef
>>> cont # inspeção do conteúdo do ficheiro f1
'abcdef'
>>> f1.close()
>>> # 'teste1' (já existente) é aberto para escrita
>>> f1 = open('teste1', 'w')
>>> f1.close()
>>> f1 = open('teste1', 'r') # 'teste1' é aberto para leitura
>>> cont = f1.read()
>>> cont # o conteúdo do ficheiro foi apagado
''
>>> f1.close()
```

A interação anterior mostra que se um ficheiro existente é aberto para escrita, o seu conteúdo é apagado com a operação de abertura do ficheiro.

Após a abertura de um ficheiro para escrita, é também possível utilizar a função `print` apresentada na Secção 2.5.2, a qual, na realidade, tem uma sintaxe definida pelas seguintes expressões em notação BNF:

⟨escrita de dados⟩ ::= `print()` |
`print(file =` ⟨nome de ficheiro⟩`)` |
`print(`⟨expressões⟩`)` |
`print(`⟨expressões⟩`, file =` ⟨nome de ficheiro⟩`)`

⟨nome de ficheiro⟩ ::= ⟨nome⟩

Para as duas novas alternativas aqui apresentadas, a semântica desta função é definida do seguinte modo: ao encontrar a invocação da função `print(file =` ⟨nome⟩`)`, o Python escreve uma linha em branco no ficheiro ⟨nome⟩; ao encontrar

a invocação da função `print(`⟨exp$_1$⟩`, ...` ⟨exp$_n$⟩`, file =` ⟨nome⟩`)`, o Python avalia cada uma das expressões ⟨exp$_1$⟩ ... ⟨exp$_n$⟩, escrevendo-as na mesma linha do ficheiro ⟨nome⟩, separadas por um espaço em branco e terminando com um salto de linha.

A seguinte interação mostra a utilização da função `print` utilizando nomes de ficheiros:

```
>>> f1 = open('teste1', 'w')  # 'teste1' é aberto para escrita
>>> f1.write('abc')
3
>>> print('cde', file = f1)
>>> print('fgh', 5, 5 * 5, file = f1)
>>> print('ijk', file = f1)
>>> f1.close()
>>> f1 = open('teste1', 'r')  # 'teste1' é aberto para leitura
>>> cont = f1.read()
>>> cont  # conteúdo de 'teste1'
'abccde\nfgh 5 25\nijk\n'
>>> print(cont)
abccde
fgh 5 25
ijk

>>> f1.close()
>>> f1 = open('teste1', 'a') # 'teste1' é aberto para adição
>>> print(file = f1)
>>> print(file = f1)
>>> f1.write('lmn')
3
>>> f1.close()
>>> f1 = open('teste1', 'r')  # 'teste1' é aberto para leitura
>>> cont = f1.read()
>>> cont    # conteúdo de 'teste1'
'abccde\nfgh 5 25\nijk\n\n\nlmn'
>>> print(cont)
abccde
```

```
fgh 5 25
ijk


lmn
>>> f1.close()
```

11.4 Notas finais

Apresentámos o conceito de ficheiro, considerando apenas ficheiros de texto. Um ficheiro corresponde a uma entidade que existe no computador independentemente da execução de um programa.

Para utilizar um ficheiro é necessário abrir o ficheiro, ou seja, associar uma entidade do nosso programa com um ficheiro físico existente no computador ou na rede a que o computador está ligado e dizer qual o tipo de operações a efetuar no ficheiro, a leitura ou a escrita de informação.

Depois de aberto um ficheiro apenas é possível efetuar operações de leitura ou de escrita, dependendo do modo como este foi aberto.

11.5 Exercícios

1. Escreva uma função em Python que que recebe duas cadeias de carateres, que correspondem a nomes de ficheiros. Recorrendo a listas, a sua função, lê o primeiro ficheiro, linha a linha, e calcula quantas vezes aparece cada uma das vogais. Após a leitura, a sua função escreve no segundo ficheiro, uma linha por vogal, indicando a vogal e o número de vezes que esta apareceu. Apenas conte as vogais que são letras minúsculas. Por exemplo, a execução de

   ```
   >>> conta_vogais('testevogais.txt', 'restestavogais.txt')
   ```

   poderá dar origem ao ficheiro:

   ```
   a 41
   e 45
   i 18
   ```

```
o 26
u 20
```

2. Escreva uma função em Python que que recebe três cadeias de carateres, que correspondem a nomes de ficheiros. Os dois primeiros ficheiros, contêm números, ordenados por ordem crescente, contendo cada linha dos ficheiros apenas um número. O seu programa produz um ficheiro ordenado de números (contendo um número por linha) correspondente à junção dos números existentes nos dois ficheiros. Este ficheiro corresponde ao terceiro argumento da sua função. Para cada um dos ficheiros de entrada, o seu programa só pode ler uma linha de cada vez.

3. Escreva uma função em Python que que recebe uma cadeia de carateres, que contém o nome de um ficheiro, lê esse ficheiro, linha a linha, e calcula quantas vezes aparece cada uma das vogais. Recorrendo a dicionários, a sua função deve devolver um dicionário cujas chaves são as vogais e os valores associados correspondem ao número de vezes que a vogal aparece no ficheiro. Apenas conte as vogais que são letras minúsculas. Por exemplo,

   ```
   >>> conta_vogais('testevogais.txt')
   {'a': 36, 'u': 19, 'e': 45, 'i': 16, 'o': 28}
   ```

4. Escreva uma função em Python que que recebe duas cadeias de carateres, que correspondem a nomes de ficheiros. A sua função, lê o primeiro ficheiro, linha a linha, e calcula quantas vezes aparece cada uma das vogais. Após a leitura, a sua função escreve no segundo ficheiro, uma linha por vogal, indicando a vogal e o número de vezes que esta apareceu. Apenas conte as vogais que são letras minúsculas. Por exemplo, a execução de

   ```
   >>>  conta_vogais('testevogais.txt', 'restestavogais.txt')
   ```

   poderá dar origem ao ficheiro:

   ```
   a 41
   e 45
   i 18
   o 26
   u 20
   ```

5. Escreva um programa em Python para formatar texto. O seu programa deve pedir ao utilizador o nome do ficheiro que contém o texto a ser

formatado (este ficheiro, designado por ficheiro fonte, contém o texto propriamente dito e os comandos para o formatador de texto tal como se descrevem a seguir) e o nome do ficheiro onde o texto formatado será guardado (o ficheiro de destino). Após esta interação deve ser iniciado o processamento do texto, utilizando os seguintes valores de omissão:

- Cada página tem espaço para 66 linhas;
- O número de linhas de texto em cada página é de 58;
- A margem esquerda começa na coluna 9;
- A margem direita acaba na coluna 79;
- As linhas são geradas com espaçamento simples (isto é, não são inseridas linhas em branco entre as linhas de texto);
- As páginas são numeradas automaticamente, sendo o número da página colocado no canto superior direito;
- Os parágrafos (indicados no ficheiro fonte por uma linha que começa com um ou mais espaços em branco) são iniciados 8 espaços mais para a direita e separados da linha anterior por uma linha em branco;
- Cada linha do texto começa na margem da esquerda e nunca ultrapassa a margem da direita.

Estes valores de omissão podem ser alterados através de comandos fornecidos ao programa, comandos esses que são inseridos em qualquer ponto do texto. Cada comando é escrito numa linha individualizada que contém o caráter "`$`" na primeira coluna. Os comandos possíveis são:

`$ nl`

Este comando faz com que o texto comece numa nova linha, ou seja, origina uma nova linha no ficheiro de destino.

`$ es` ⟨n⟩

Este comando faz com que o espaçamento entre as linhas do texto passe a ser de ⟨n⟩ linhas em branco (o símbolo não terminal ⟨n⟩ corresponde a um inteiro positivo).

`$ sp` ⟨n⟩

Este comando insere ⟨n⟩ linhas em branco no texto, tendo em atenção o valor do espaçamento (o símbolo não terminal ⟨n⟩ corresponde a um inteiro positivo).

`$ ce` ⟨texto⟩

Este comando causa um salto de linha e centra, na linha seguinte, a cadeia de carateres representada pelo símbolo não terminal ⟨texto⟩.

`$ al`

Depois da execução deste comando, todas as linhas produzidas pelo programa estão alinhadas, ou seja, as linhas começam na margem esquerda e acabam na margem direita, embora o número de carateres possa variar de linha para linha.

Para conseguir este comportamento, o seu programa insere espaços adicionais entre as palavras da linha de modo a que esta termine na margem direita.

`$ na`

Depois da execução deste comando, as linhas produzidas pelo programa não têm de estar alinhadas, ou seja, as linhas começam na margem da esquerda e podem acabar antes da margem da direita, desde que a palavra seguinte não caiba na presente linha. Neste caso, não se verifica a inserção de espaços adicionais entre palavras.

`$ me` ⟨n⟩

Depois da execução deste comando, a margem da esquerda passa a começar na coluna correspondente ao símbolo não terminal ⟨n⟩. Se o valor de ⟨n⟩ for maior ou igual ao valor da margem da direita, é originado um erro de execução e este comando é ignorado (o símbolo não terminal ⟨n⟩ corresponde a um inteiro positivo).

`$ md` ⟨n⟩

Depois da execução deste comando, a margem da direita passa a acabar na coluna correspondente ao símbolo não terminal ⟨n⟩. Se o valor de ⟨n⟩ for menor ou igual ao valor da margem da esquerda, é originado um erro de execução e este comando é ignorado (o símbolo não terminal ⟨n⟩ corresponde a um inteiro positivo).

`$ pa`

A execução deste comando causa um salto de página, exceto se, quando ele for executado, o formatador de texto se encontrar no início de uma página.

Sugestão: Utilize uma cadeia de carateres para armazenar a linha que está a ser gerada para o ficheiro de destino. O seu programa deve ler palavras

do ficheiro fonte e decidir se estas cabem ou não na linha a ser gerada. Em caso afirmativo, estas são adicionadas à linha a ser gerada, em caso contrário a linha é escrita no ficheiro de destino, eventualmente depois de algum processamento.

6. Adicione ao processador de texto do exercício anterior a capacidade de produzir um índice alfabético. Para isso, algumas das palavras do seu texto devem ser marcadas com um símbolo especial, por exemplo, o símbolo "`@`", o qual indica que a palavra que o segue deve ser colocada no índice alfabético, que contém, para cada palavra, a página em que esta aparece. Por exemplo, se no seu texto aparece `@programação`, então a palavra "programação" deve aparecer no índice alfabético juntamente com a indicação da página em que aparece. As palavras do índice alfabético aparecem ordenadas alfabeticamente. Tenha em atenção os sinais de pontuação que podem estar juntos com as palavras mas não devem aparecer no índice.

   Sugestão: utilize um dicionário em que cada chave contém uma palavra a inserir no índice alfabético, associado à página, ou páginas, em que esta aparece. Mantenha este dicionário ordenado.

7. Escreva um programa que aceite como dado de entrada um ficheiro fonte do seu formatador de texto, e conte quantas palavras e quantos parágrafos o texto contém. Tenha cuidado para não contar os comandos do formatador de texto como palavras.