AWK

José João Almeida
José Bernardo Barros
Pedro Rangel Henriques

91/02/13

Contents

1 AWK

AWK = Aho, Weinberger and Kernighan

1.1 Genericamente

AWK é uma linguagem de programação vocacionada para o processamento de texto (como é habitual, AWK é, também, o nome do interpretador para essa linguagem, i. e., do programa que transforma os nossos programas escritos em AWK em programas executáveis que se comportam como "filtros" de texto). Utiliza-se normalmente em:

- Transformação de formatos
- Teste de consistência
- Procura de registos (linhas) que verifiquem certa propriedade; operações sobre de bases de dados (pessoais)
- Cálculo de estatísticas
- Problemas envolvendo procura e substituição de padrões

1.2 Estrutura geral dum programa AWK

Cada programa AWK vai conter uma sequência de pares condição – acção.

Ao ser executado, um programa "awk", vai ler um texto, registo a registo (por defeito um registo é uma linha) e vai efectuar sobre essa linha todas as acções correspondentes às condições verdadeiras.

Cada registo é visto como uma sequência de campos (por defeito, cada campo é separado por um espaço ou tab). Na leitura, os campos são automaticamente divididos e directamente referenciáveis nas condições e acções.

Devido a ter:

- Declaração implícita de variáveis
- Inicialização automática,
- Leitura de registos e separação em campos automática
- Arrays associativos
- Pattern Matching e pattern substitution predefinidos
- etc,

os programas AWK são normalmente bastante mais pequenos do que os que realizariam a mesma tarefa escritos em C ou PASCAL.

Como se disse, um programa "awk" consiste numa sequência de pares:

- condição , envolvendo possibilidade de testar:
  - se o registo lido (ou cada um dos seus campos) inclui ou não certo pattern
  - expressões envolvendo quantidades, comprimentos de campos, nmeros de registos, etc
  - condição por defeito: Verdadeira
- acção , por exemplo:
  - cálculo de estatísticas simples
  - escrita de conclusões
  - substituição de texto por texto
  - atribuições, ciclos, condicionais, etc
  - acção por defeito: copiar registo de entrada para a saída

Um programa AWK pode ainda conter definição de funções.

1.3 Algoritmo geral interno associado

```
PARA cada Registo pertencente ao texto a processar FAZER
 |
 | PARA cada linhaAvk do programa avk FAZER
 |  |
 |  | SE linhaAvk.condicao se verifica em registo ENTAO
 |  |   |   aplicar linhaAvk.accao a registo
```

1.4 Exemplos simples

Uma das grandes utilizações do AWK é em casos simples

1.4.1 Utilização do comando AWK

```
avk '..programaAvk..' (file | inicializacao)+
avk -f avkProgFilename (file | inicializacao)+
```

Quando não são indicados nomes de ficheiros a processar, é usada a entrada standard (stdin = teclado por defeito). Deste modo é possível utilizar este comando em pipes.

Note-se que o programa AWK (as instruções de processamento dos textos) pode estar contido noutro ficheiro (usar o switch -f seguido do nome do ficheiro) ou podem ser escrito directamente entre apóstrofes (ver 1.a forma).

É possível a inicialização de variáveis internas na própria linha de comando, através de uma instrução awk (tipicamente uma atribuição).

1.4.2 Exemplos

- Imprimir as linhas do ficheiro "f1" com mais de 72 caracteres:

  ```
  avk 'length > 72' f1
  ```

  Obs: a acção vazia equivale copiar registo lido para a saída. (length = comprimento da linha de entrada).

- Imprimir pela ordem oposta os dois primeiros campos de cada registo lido

  ```
  avk '{ print $2, $1 }' f1
  ```

  Obs: a condição vazia equivale a Verdade.

- Calcular o somatório do primeiro campo de todos os registos do ficheiro "f1" e imprimi-lo assim como a média

  ```
  avk -f soma f1
  ```

  sendo "soma":

```
        { a += $1 }
END     { print "soma = ", a, " media = ", a/NR }
```

Obs: A primeira acção é aplicada a todos os registos lidos, implementando o cálculo do somatório dos primeiros campos `a = a + $1`.
`$1` – primeiro campo do registo lido.
No final (condição "END") são escritos os resultados.

- Imprimir os campos pela ordem inversa

```
avk -f invert f1
```

sendo "invert":

```
{ for (i = NF; i > 0; --i) print $i }
```

Obs: `--i` equivale a `i = i -1`

- Imprimir todas as linhas incluídas entre o pares start/stop

```
avk '/start/ , /stop/' f1 f2
```

Obs: `/start/` – condição que é verdadeira em todos os registos que contêm a palavra "start".
`/start/ ,/stop/` – condição que é verdadeira em todos os registos contidos entre registos que contêm a palavra "start" outros que contêm a palavra "stop".

- Imprimir todas as linhas em que o primeiro campo é diferente do primeiro campo do registo anterior

```
avk -f anterior f1
```

sendo "anterior":

```
$1 != prev { print; prev = $1 }
```

Obs: `prev` inicialmente é ""; para cada registo em que o primeiro campo é diferente do anterior, é escrito todo o registo e é actualizado o prev.

- Imprimir um ficheiro "input", paginando-o, começando a numerar as páginas em 5

```
pr input | avk -f numera n=5
```

sendo "numera":

```
/Page/ { $NF = n++; }
       { print }
```

Obs: O comando "pr" cria automaticamente um cabeçalho em cada página com o seguinte formato: `Mes Dia Hora Ano Nome_do_ficheiro  Page num`
Exemplo:
`Jan 23 18:21 1991 input Page 1`
O programa AWK "numera" vai modificar O Ultimo campo (`$NF`) da linha de cabeçalho, atribuindo-lhe um novo numero (`n`) que deverá ser incrementado e externamente inicializado a "5".

1.5 Gramática comentada do AWK

Um programa AWK é uma sequência de pares **Condição Acção** (como referido no inicio) e/ou definições de funções:

```
AWK_prog -> (  Condicao ["{" Accao "}"]
            | DefdeFuncao            )+
```

1.5.1 Condições

```
Condicao   -> ExpRegular             (1)
            | Expressao PattMatchOp ExpRegular
            | Expressao
            | "BEGIN"                (2)
            | "END"                  (3)
            | CondExp
```

(1) - Verdade se registo inclui a ExpRegular
(2) - Verdade no inicio do texto
(3) - Verdade no fim do texto

```
CondExp   -> Condicao "||" Condicao          (OU)
           | Condicao "&&" Condicao          (E)
           | "!" Condicao                    (NOT)
           | Condicao "," Condicao           (1)
           | "(" Condicao ")"
```

(1) - Verdade em todo o intervalo que começa no registo onde a primeira condição se verifica e acaba no registo onde a segunda condição é Verdade

```
ExpRegular -> "/" ...  "/"  (1)
```

(1) - Usa uma sintaxe semelhante à do VI, ED, GREP (ver exemplos abaixo)

```
Expressao -> termo
           | termo termo ...             (1)
           | var opatrib expressao       (2)
```

```
(1) - justaposicao - concatenacao dos termos tomados como strings
(2) - opatrib e' um de: - += -= *= /= %=
```

```
termo      -> (termo)
            |  incrvar                (++v --v v-- v++)
            |  termo opbin termo      (+ / + - * in)
            |  opuna termo            (+ -)
            |  TermoAtomico

TermoAtomico -> ConstNumerica
            |  ConstString
            |  var
            |  funcao
```

1.5.2 Elementos lexicos

- Constantes numéricas: notação habitual
- Constantes strings: incluídas entre aspas; podem ter "\n"
- palaVras reservadas e variáveis globais predefinidas:
  - **BEGIN** - Pattern que é verdade no início do ficheiro (usado habitualmente para inicialização de variaveis, etc)
  - **END** - Pattern que é verdade no fim do ficheiro (usado para fazer acções finais)
  - **FILENAME** - Nome do ficheiro actualmente a ser processado
  - **FS** - Variável que armazena o separador de campos do registo de entrada (por defeito sequência de " " ou "\t")
  - **RS** - Separador de registos na entrada (por defeito "\n")
  - **OFS** - separador de campos na saída (por defeito " ")
  - **ORS** - separador de registos na saída (por defeito "\n")
  - **NF** - nmero de campos do registo actual
  - **NR** - nmero do registo actual
  - **OFMT** - Formato de impressão de reais
  - **ARGC** - nmero de argumentos da linha de comandos
  - **ARGV** - array de parametros da linha de comandos
  - **RLENGTH** , **RSTART** - ver função match (1.7b)

Algumas considerações acerca dos separadores:

- **FS** - expressão regular separadora de campos (por defeito qualquer sequência de espacos ou tab); pode ser alterado por FS= "ExpReg"
- **RS** - caracter separador de registo (por defeito "\n"); pode ser alterado através de RS="caracter" . No caso particular de RS="" , o separador de campos passa a ser uma linha vazia!!
- **OFS** - separador de campos na saída; pode ser mudado para qualquer string
- **ORS** - separador de registos na saída; pode ser alterado para qualquer string.

- Operadores:
  - de atribuição: = += -= *= /= %= ++ --
  - aritmeticos: + - * / % ^
  - relacionais: < > <= >= != >=
  - logicos: && || !
  - Pattern Matching com expressões regulares (PatMatcOp):
    ~ (faz matching)
    !~ (não faz matching)
  - Condicional: cond "?" expressãoVerdade ":" expressãoFalso
  - Relacionados com arrays:
    * in (ex. `if ( a in Arrayid ) )`
    * delete (ex. `delete Arrayid "[" expressao "]" )`
- Delimitadores e seus significados:
  - {...} delimita acção
  - "..." delimita strings
  - /.../ delimita expressões regulares.
- Comentários
  Iniciam-se com #; terminam com "\n"

1.5.3 Acesso a valores de Campos ou registos

- $0 - registo actual inteiro
- $1,$2 - campo1 campo2 do registo actual
- $NF - ltimo campo

1.6 Variaveis

Podem ser simples (inteiras, reais, strings) ou ARRAYS.

Os arrays podem ter índices inteiros ou strings!

Não são declaradas e são automaticamente inicializadas a 0, "" ou array vazio, conforme o respectivo tipo.

Os tipos são automaticamente convertidos entre si

Para percorrer um array, está definido uma estrutura de controlo

```
for(ind in idarray) instruc

Ex.: Populacao["asia"]=20
     Populacao["africa"]=40
     for (p in Populacao)
         print("a populacao de " p "e'" Populacao[p])
```

1.7 Funções predefinidas

1.7.1 Numéricas

- exp, int, log, sqrt, sin, cos: ver C
- srand(semente) (define um elemento que será tomado como base para a geração (pseudo)aleatória de numeros).
- srand() (toma como base para a sequência aleatoria, um numero calculado a partir da hora e data)
- rand() - numero aleatorio entre 0 e 1

1.7.2 Referentes a strings

- **index** (str,substr)
  index("abc","bc")=2; index("abc","cd")=0
- **length** (str)
  length = length($0); length("abc")=3
- **split** (str,array[,sep])
  split("eraumavez",a,"v")=3 e
  a[1]="era";a[2]="uma";a[3]="vez". Quando sep é omitido, sep=FS
- **sprintf**: ver C (ou "man sprintf")
- **substr** (str,pos[,comp])
  substr("abcde",3,2)="cd", substr("abcde",2)="bcde"
- **gsub** (ExpReg,string [,var]) - faz a substituição global de todas as ocorrências de ExpReg por string em var (registo entrado)
- **sub** (ExpReg,string [,var]) - faz a substituição da primeira ocorrência de ExpReg por string em var (registo entrado)
- **match** (string,Expreg) - vê se ExpReg ocorre em string e guarda nas variáveis globais "RLENGTH" e "RSTART" o comprimento e o início da 1.a ocorrência encontrada.

1.8 Expressões Regulares

```
/^A/        - qualquer registo comecado por A

/^[ABC]/    - qualquer registo comecado por A ou B ou C

/Arvore/    - qualquer registo contendo a string Arvore

/a$/        - qualquer registo terminado por a
```

```
/x|y/       - todas os registos que contenham x ou y

/ax+b/      - "a" seguido de uma ou mais repeticoes de "x"
              seguido de "b"

/ax*b/      - "a" seguido de zero ou mais "x" seguido de "b"

/ax?b/      - ab ou axb

/a.b/       - "a" seguido de qualquer caracter menos "\n"
              seguido de "b"

..(...)..   - altera prioridades
```

1.9 Acções

```
Acao -> (instrucao limitador)+

limitador -> ";"| "\n"

instrucao -> atribuicao
           | instrucaoComposta              {...}
           | ifinstrucao            if(cond)instru [else instru]
           | whileinstrucao         while(cond)instru
           | forinstrucao           ver C
           | for(in)instrucao       (1)
           | break                  (ver C)
           | continue               (ver C)
           | next                   (2)
           | exit                   (3)
           | instruEntrada          (4)
           | instruSaida            (4)
```

(1) for(indice in hdarray)instrução (ver variáveis em 1.6)
(2) passa a novo registo (linha)
(3) Salta para a acção associada ao END
(4) Entrada e saída não implícita no algoritmo geral

As variáveis são automaticamente inicializadas a "" o que tem um valor 0.

Concatenação de strings é indicada por sequência de expressões (a justaposição corresponde ao operador concatenação):

```
{x = "viva"
 x = x " eu " "!!!!"
 print x }

daria

   "viva eu !!!!"
```

Acções de entrada e saída:

```
instruSaida -> "print" expressao ("," expressao)* [redirec]
             |  "printf" "(" formato "," expressoes ")"

instruEntrada ->  "getline"
                |  "getline" variavel
                |  "getline" [variavel] "<" file
                |  comando "|" getline [variavel]

redirec-> ">" file
        | ">>" file
        | "|" comando
```

print – escreve o valor das expressões na saída, separadas por OFS e terminadas por ORS. Pode levar (opcionalmente) "(" ")" a delimitar a sequência dos argumentos.

printf – escreve formatado (ver man printf)

```
print       (sem argumentos) escreve o registo actual na saida
print $0      igual a print
print $i      escreve o campo i
print "valor obtido", $0 > "filename"
print >> "filename"   escreve para um ficheiro. (n.o maximo de
              ficheiros simultaneos = 10 )
print | "mail jj"     saida para uma PIPE unix
print $i | "sort"
printf        ver C
```

getline [var] – le caracteres da entrada até ao caracter fim de registo (RS). Esses caracteres são colocados em $0 [ou var]. $0 é seguidamente dividido em campos (de acordo com o separador "FS") preenchendo $1, ..., $NF, NF.
Valor retornado:

- 0 – se encontrou EOF
- 1 – sucesso na operação
- −1 – erro na leitura

```
getline       le proximo registo na entrada; NR 'e actualizado
getline < file   le proximo registo do ficheiro file; NR nao 'e
              actualizado
getline varid < "file"   le para varid o proximo registo do ficheiro
              "file"; NR nao e' actualizado
"date" | getline        le para $0 o resultado do comando "date"
```

1.10 Outros exemplos

1.10.1 Preprocessador para substituir no texto de entrada as linhas que

tenham a forma: #include "filename", pelo conteúdo de "filename" :

```
$1 ~ "#include" {gsub(/"/, "", $2);       #(1)
                 system("cat " $2);       #(2)
                 next;}                   #(3)
                {print}                   #(4)
```

(1) – Retira as aspas deixando em $2 apenas o nome do ficheiro.
(2) – Executa o comando "cat <nome do ficheiro>", inserindo a sua saída no texto resultado.
(3) – Next - para não imprimir a linha #include ...
(4) – Imprime todas as outras linhas.

1.10.2 Print etc/passwd de modo legível (ver /etc/passwd)

O ficheiro /etc/passwd contem a lista dos utilizadores, passwords, numeros internos, grupos a que pertencem, etc. de todos os utilizadores de um sistema UNIX.

A cada utilizador corresponde uma linha desse ficheiro e os vários campos estão separados por ":", conforme se exemplifica abaixo:

```
% cat /etc/passwd
   pina:GtMhvfyaYtr/A:0:100:ANTONIO PINA:/usr/acct/pina/:
   asq:gSWfC9WjpXu0a:0:100:ARTUR QUINTAS:/usr/acct/asq/:
   root:h01SFZptvvG2E:0:1::/:
   startup:SaXO6rFToMVHE:0:1::/:/etc/startup
   shutdown:FSyOOL/K9jvHE:0:1::/:/etc/shutdown
   uucp:*:6:6:uucp administrative login:/usr/lib/uucp:
   ...
```

Pretende-se fazer um comando para imprimir entradas com formato mais adequado, como se indica a seguir:

```
Username = pina          password = GtMhvfyaYtr/A
Numero = 0000
            Grupo:[ 100 ]
 Nome = ANTONIO PINA

Username = asq          password = gSWfC9WjpXu0a
Numero = 0000
            Grupo:[ 100 ]
 Nome = ARTUR QUINTAS
...
```

Para isso bastaria o programa AWK que se segue:

```
% cat x.awk
   BEGIN  {FS=":"}
          {printf ("Username = %s         password = %s\n",$1,$2);
           printf ("Numero = %s    \n\t\t\t\tGrupo:[ %s ]\n",$3,$4);
           printf (" Nome = %s\n",$5);}
```

Uma possível utilização do programa escrito seria:

```
% grep pina /etc/passwd | awk -f z.awk

  Username = pina         password = GtMhvfyaYtr/A
  Numero = 0000
             Grupo:[ 100 ]
   Nome = ANTONIO PINA
```

Note-se que apenas se mandou formatar a linha do ficheiro correspondente ao utilizador "pina".

1.10.3 Generalização do caso anterior

O exemplo seguinte "prtel.awk", generaliza esta ideia supondo casos de tabelas em que a primeira linha tem identificadores dos campos e as linhas seguintes tem a informação til. Considere, por exemplo, que se pretende formatar o ficheiro de telefones seguinte:

```
% cat tel

nome:telefone
Jose Joao:975867
UM paco:27007
emergencia:115
```

O programa AWK abaixo

```
% cat prtel.awk:

BEGIN     { FS=":"}
NR==1     { for (i=1;i<=NF;i++) id[i]=$i;}
NR>1      { print("-----------------------------");
            for(i=1;i<=NF;i++)
               print("    " id[i] " = [" $i "]");
            print("-----------------------------"); }
```

produziria o seguinte resultado:

```
% awk -f prtel.awk tel

-----------------------------
   nome = [Jose Joao]
   telefone = [675866]
-----------------------------
-----------------------------
   nome = [UM paco]
   telefone = [27007]
-----------------------------
-----------------------------
   nome = [emergencia]
   telefone = [115]
-----------------------------
```

1.10.4 Sistema para arquivo de pequenos textos

No exemplo que a seguir se apresenta construiremos três comandos – três "scripts" que usam programas awk – para efectuar a inserção e consulta de um arquivo de textos:

- arput *tit arquivo texto* - armazena o ficheiro "texto" no arquivo "arquivo". O texto fica associado ao título "tit".
- arget *tit arquivo* - manda para a saída o texto de "arquivo" associado ao título "tit".
- artit *arquivo* - mostra todos os títulos existentes em arquivo

```
% cat arput
  #!/bin/csh
  # ARPUT titulo Arquivo [ficheiro]
  if($#argv >= 2 ) then                          #parametros : 2 ou mais
    set x='artit $2 | grep "^$1\."|wc -l'        # ve se titulo existe
    if ($x != 0) then                            # se j'a existe ...
      echo "ERRO: $1 ja existente"
    else                                         #titulo ainda nao existe
      awk ' \
        BEGIN { print ":(+" tit "+)." >> arq}  #copia um cabecalho   \
              { print >> arq }                 #copia as linhas todas\
        END   { print ".(+" tit "+)." >> arq}' #copia um terminador  \
            tit=$1 arq=$2 $3
    endif
  else
    echo "ERRO: "
    echo "utilizacao: $0 titulo arquivo [ficheiro]"
  endif
```

Mais uma vez, este conjunto de scripts pode ser usado em interligação com outros comandos UNIX (em pipes, ate associado ao VI) de modo a permitir guardar e obter bocados de programas, cartas, etc.

Exemplo de utilização:

```
% arput carta1 jj.ar carta
% ls -la | arput dir jj.ar
% cat jj.ar
  :(+carta1+).
  @remetente                              @data
                                          @destinatario
  ----------------------------------------------------
  sua referencia                          nossa referencia
  @saref                                   @nref

  Caro senhor
     E' com o maior prazer que ...
                                 Sou quem sabes
                                          Maria Alice
  .(+carta1+).
```

```
:(+dir+).
drwxrwxrwx  9 jj          512 Jan 15 18:25 .
drwxr-xr-x 25 jj          848 Jan 15 17:14 ..
drwxrwxrwx  2 jj          288 Jan 15 18:21 SCCS
-rw-r--r--  1 jj        19042 Jan 16 16:29 awk
-rw-r--r--  1 jj        14783 Apr  6  1990 lex
-r--r--r--  1 jj        15851 Jan  9 17:50 make
-r--r--r--  1 jj         7353 Apr 12  1989 yacc
.(+dir+).
```

```
% cat artit
   #!/bin/csh
   # ARTIT Arquivo
   if ($#argv == 1) then
       # eliminar os ":(+ +)" das linhas contendo titulos
       awk '/^\:\(\+/ {gsub(/[:(+)]/,"");print}'  $1
   else
       echo "ERRO:"
       echo "utilizacao: $0 arquivo"
   endif
```

```
% artit jj.ar
   cartai.
   dir.
```

Utilização (continuação do exemplo anterior):

```
% cat arget
   #!/bin/csh
   # ARGET titulo arquivo
   if ($#argv == 2) then
       awk -f arget.awk tit=$1 $2
   else
       echo "ERRO"
       echo "utilizacao: $0 titulo arquivo"
   endif
```

```
% cat arget.awk
   BEGIN              {inicio=":(+" tit "+)." ;
                           fim=".(+" tit "+)." ;}
   ($0 == fim )       {interessa = ""; exit}
   (interessa =="a") {print }
   ($0 == inicio)     {interessa = "a"}
```

Note-se que a ordem dos pares "condição – acção" não é arbitrária; se por exemplo escrevermos a linha ($0 == fim) ... depois da linha seguinte, o título final apareceria na saída.

Utilização (continuando ainda com o exemplo anterior):

```
% arget cartai jj.ar
   Remetente                          Data
```

```
                                        @destinatario
-------------------------------------------------------
sua referencia                          nossa referencia
@saref                                   @nref

Caro senhor
   E' com o maior prazer que ...
                                Sou quem sabes
                                         Maria Alice
```

1.10.5 Base de Dados Bibliográfica em formato BibTex

O BibTex é um formato de referências bibliográficas usado no sistema de processamento de texto LaTeX. Com base num texto BibTex, é feita a geração do capítulo "referências" onde são colocadas (em formato conveniente, standardizado e ordenado) as entradas correspondentes às referências que apareceram citadas durante o texto.

Considere o seguinte exemplo de um extrato de um texto bibtex:

```
@techreport{BWS3a,
   author = "Manfred Broy and Martin Wirsing",
   title = "Generalized Heterogeneous Algebras",
   year = 1983,
   month = Feb,
   institution = "Institut fur Informatik, TUM",
   note = "(draft version)",
   annote = "espec algebrica"
   }
@inbook{Val87a,
   author = "José M. Valenca",
   title = "Algoritmos",
   chapter = 1,
   year = 1987,
   month = Oct,
   series = "Sebentas de Ciências da Computa\,cão",
   publisher = gdcc,
   address = um,
   annote = "algoritmos, espec formal"
   }
...
```

Neste formato é possível colocar um campo "annote", onde tipicamente se introduz uma classificação da referência em causa.

Pretende-se um programa que procure e imprima as "entradas" que contêm certo padrão.

Note-se que o caracter @ pode ser usado como separador de registos, registos esses que vão ocupar mais que uma linha.

Para tal considere e analise a seguinte "script" escrita com base em dois programas AWK:

```
   #!/usr/ucb/csh
   set bib=/usr/acct/epl/DOC/BIBLIO/prh.bib

   if ($1 == "-c") then                                    #(0)
```

```
awk 'BEGIN {RS="@";FS="\n" ;ORS="\n";OFS="\n"}                    #(1)\
    /'$2'/ {for(i=1;i<=NF;i++)                                    #(2)\
               if($i ~ /(author |title |annote )/) print $i       #(3)\
            print "-------------------------" }'  $bib | more
else
awk 'BEGIN {RS="@";FS="\n" ;ORS="\n";OFS="\n"}    \
    /'$1'/ {for(i=1;i<=NF;i++) print $i           \
            print "-------------------------" }'  $bib | more
endif
```

(0) - se se pretende a versão compacta (i.e, apresentar só os autores, títulos e notas dos registos pretendidos) então,
(1) - Separador de registo = @; cada linha é um campo
(2) - Para todos os campos que existam em cada registo que verifique a o padrão indicado no 2.o parametro da script,
(3) - Se esse campo incluí as palavras "author " ou "title " ou "annote ", então escreve-lo

Exemplo de resultados obtidos com esta script:

```
% bib formal

inbook{Val87a,
   author = "José M. Valença",
   title = "Algor\'itmos",
   chapter = 1,
   year = 1987,
   month = Oct,
   series = "Sebentas de Ciências da Computa\,cão",
   publisher = gdcc,
   address = um,
   annote = "algoritmos, espec formal"
   }
-------------------------
book{Rev85a,
   author = "G. E. Revész",
   title = "Introduction to Formal Languages",
   year = 1985,
   publisher = "McGraw-Hill Book Co.",
   annote = "linguagem formal, gramaticas"
   }
-------------------------

% bib -c formal
-------------------------
   author = "José M. Valença",
   title = "Algor\'itmos",
   annote = "algoritmos, espec formal"
-------------------------
   author = "G. E. Revész",
   title = "Introduction to Formal Languages",
   annote = "linguagem formal, gramaticas"
```