Sistemes Operatius II

Pràctica 1

Juan Marin Vega

Iñaki Ubeda Martinez

Lectura de dades d'un fitxer

fgetc

Al utilitzar com a mètode de lectura la funció ***fgetc()*** de vegades surt el caràcter **ASCII 10** que correspon a una nova línia.

Per tal d'utilitzar aquesta funció com a mètode de lectura s'ha de fer la comprovació de final de fitxer després d'haver llegit el caràcter del fitxer obert.

Considerem que aquest es bon un mètode per a extreure paraules d'un fitxer perquè obtenim un control en "temps real" sobre la informació que llegim del fitxer. En cas que la paraula no tingui els paràmetres establerts la podem descartar ràpidament sense haver estat processada prèviament.

fgets

La funció **fgets()** llegeix tota una línia, o fins a la grandària especificada i ho guarda en un buffer.

Al executar el codi exemple, veiem que l'ultima línia no es mostra ja que hi troba un **EOF** i surt del **while**. Podem modificar el codi i posar un **printf()** nomes sortir del **while** per que imprimeixi l'ultima línia.

Si reduïm el *buffer* d'entrada a 10 *bytes*, efectivament nomes llegeix les línies fins a 10 caràcters, i la resta es perden, no es mostren en la següent iteració.

No hem utilitzat aquesta funció per que a priori no es pot saber la longitud màxima de una línia, i encara que podíem suposar que seria de màxim 100 caràcters, no sembla una solució gaire neta.

Per tal d'eliminar el salt de línia de la cadena podríem utilitzar aquest codi:

```
str[strlen(str) - 1] = '\0';
```

Així el que fem es substituir el **'\n'** per un **'\0'**, amb lo qual aconseguim retallar la cadena.

Juan Marin Vega
Iñaki Ubeda Martinez

fscanf

La funció **fscanf**, en aquest codi, esta llegint conjunts de caràcters fins a que troba un caràcter de tipus espai. Això s'aconsegueix fent el *matching* **%s**, que com explica a la documentació, fa un *match* a una seqüència de caràcters que no siguin un espai. La entrada termina quan arriba a un espai en blanc o al final de línia.
A primera vista sembla que l'**scanf()** pot facilitar la lectura de paraules, ja que et retorna cada conjunt de caràcters, però després requereix de recórrer un altre cop les paraules per analitzar el contingut i separar les valides de les no valides. Això augmentaria el cost computacional del nostre algoritme.
Si la paraula te una mida superior a la definida, llavors el programa comença a escriure a posicions del *stack* i pot produir comportaments no esperats.

fread

Aquesta pot ser una bona solució viable sempre i quan la mida del fitxer sigui no massa gran. Per a fitxers molt gran a l'hora de fer el **malloc** aquest pot fallar per no poder proporcionar la quantitat de memòria necessària.

Extracció de paraules

Per tal d'implementar les funcionalitats requerides a la practica hem utilitzat el mètode de lectura de fitxers ***fgetc***.
S'admetrà com a paraula valida tota aquella combinació de caràcters que no contingui números o caràcters invàlids (accents, dièresi…). Per tant tota aquella combinació que contingui els següents caràcters serà netejada i considerada valida:

- Signes de puntuació (interrogants, exclamacions, punts, comes, guions…)
- Operands (+, *, /)

El funcionament de l'algoritme es basa en el fet de comprovar caràcter a caràcter si el contingut guardat al *buffer* fins aquell moment es valid. En cas de que no ho sigui esborrarem el *buffer* i fins el pròxim espai (principal element diferenciador de paraules) no tornarem a considerar els caràcters com a potencial paraula.

Hem optat per trencar les paraules amb signe de puntuació enmig per a poder tenir un algoritme mes simple i fàcil de seguir visualment, creiem que les necessitats d'aquesta practica anaven mes enfocades a fer un bon us de memòria i a entendre les possibilitats que ens oferien les funcions de la llibreria de c.po
Les paraules valides s'emmagatzemen en un *array* estàtic. Es d'un màxim de 1000 punters. No es una solució del tot valida. Una solució correcta seria fer us d'estructures enllaçades en las que emmagatzemar els punters a les paraules i també no fer cas de paraules repetides, que es podria solucionar fent us de tècniques de *hashing* en lloc de comparar paraula per paraula.

Exemple 1

Text d'entrada:

To s*ing a söng thát old+was
sung. Ássuming, man's infirmities, auto-escape
it's ho'la h9k jho5454

Log:

Paraula valida: to
Paraula valida: s
Paraula valida: ing
Paraula valida: a
Paraula valida: old
Paraula valida: was
Paraula valida: sung
Paraula valida: man's
Paraula valida: infirmities
Paraula valida: auto
Paraula valida: escape
Paraula valida: it's
Paraula valida: ho'la
Paraules enregistrades: 13

Exemple 2

Text d'entrada:

GNU Free Documentation License
Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008
Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Log:
Paraula valida: gnu
Paraula valida: free
Paraula valida: documentation
Paraula valida: license
Paraula valida: version
Paraula valida: november
Paraula valida: copyright
Paraula valida: free
Paraula valida: software
Paraula valida: foundation
Paraula valida: inc
Paraula valida: http
Paraula valida: fsf
Paraula valida: org
...
Paraula valida: not
Paraula valida: allowed
Paraules enregistrades: 33

A l'exemple 1 es veu com totes les paraules que contenen els caràcters abans descrits com a invàlids son descartades.

Com es pot veure a l'exemple 2 les urls les dividim, no tractem tota la url com a única paraula, seguint així la metodologia anteriorment esmentada.