Sistemes Operatius

Informe pràctica 2

En aquesta pràctica se'ns proposa realitzar una aplicació que permeti agafar un fitxer tipus PDF i imprimir el seu contingut (les seves paraules) per la sortida estàndard.

Per fer-ho, utilitzarem la comanda *pdftotext*, una sèrie de normes per l'extracció de paraules i una indexació d'elles en una estructura de dades en forma d'abre binari.

La primera pregunta que se'ns qüestiona és cóm hem aconseguit que *pdftotext* enviï per la sortida estàndard el seu *output*, ja que per defecte crea "o imprimeix" en un fitxer tipus *.txt*. Per esbrinar-ho, he recordar els coneixements de Sistemes Operatius 1 i amb la comanda *man pdftotext* he pogut accedir a la seva documentació. A la definició del seu mètode ens expliquen que amb un guió aconseguim que aquesta sortida sigui "per pantalla", en comptes d'especificar un fitxer *.txt*. Així doncs, la declaració d'aquest mètode que s'adequa al que hem demanen es *pdftotext nom_fitxer_pdf* - .

Em aconseguit fer la canonada de l'execució d'aquesta comanda de la següent forma:

```
//Amb largument(el pdf) pasat per consola creem la comanda a executar a popen
char *comanda = malloc(sizeof(char)*20);
char *comanda2 = malloc(sizeof(char)*20);
strcpy(comanda,"pdftotext ");
comanda2 = argv[1];
strcat(comanda,comanda2);
strcat(comanda," -"); //Perque la sortida de pdftotext es faci per la sortida estandard

FILE *fpout; //Per la canonada
int len,i,par,letra_rara; //Len mantindra la longitud. i la longitud de la paraula, par
letra_rara = 1;
fpout = popen(comanda, "r");
```

Com que ens demanen que passem per paràmetre el nom del fitxer *pdf*, no es pot realitzar un *popen* "tal qual", sinó que s'han de construir un parell d'*strings* amb la comanda *strcat*, la qual junta dos *strings*. Concretament, hem reservat memòria per dos *strings* de màxim 20 *chars*, una grandària adequada, hem pensat. A la primera, li hem la comanda *pdftotext* i un espai amb la comanda *strcopy*. A la segona, li hem assignat el primer paràmetre, que es troba en *argv[1]*. Finalment, hem ajuntat la primera *string* amb la segona (*"pdftotext"* + "*argv[1] = nom_fitxer_pdf*") i per últim li hem inserit "-" perquè l'*output* es faci per la sortida estàndard. D'aquesta manera, el *popen* només queda *fpout = popen(comanda, "r")*.

En segon lloc ens demanen que expliquem com hem adaptat el codi de l'arbre binari perquè funcioni amb *chars*:

- Hem canviat el *define* que especifica el tipus dels paràmetres passats en les funcions de *int* a *char**
- Hem canviat la funció *compLT* perquè compari *chars*. Per fer-ho, hem recordat la primera pràctica i hem utilitzat el mètode *strcmp*, que retorna un número més petit que zero si el primer *string* és més petit que el *segon*.
- Hem canviat la funció *comEQ* perquè compari *chars*. Per fer-ho, hem recordat la primera pràctica i hem utilitzat el mètode *strcmp*, que retorna zero si els dos *strings* passats per paràmetre són iguals.

```
static int compLT(TYPE_RBTREE_KEY key1, TYPE_RBTREE_KEY key2)
{
    int rc = strcmp(key1,key2); //Retorna un enter mes petit que zero si el primer string es mes petit que el segon
    if(rc < 0){
        return 1;
    }
    return 0;
}

/**
 *
 * Compares if key1 is equal to key2. Should return 1 (true) if condition
 * is satisfied, 0 (false) otherwise.
 *
 */

static int compEQ(TYPE_RBTREE_KEY key1, TYPE_RBTREE_KEY key2)
{
    int rc = strcmp(key1,key2); //Retorna zero si els dos parametres (strings) son iguals
    if(rc == 0){
        return 1;
    }
    return 0;
}
```

L'arbre està format per nodes. Cada node té els següents atributs:

- Fill dret i esquerra: cada node ha de mantenir quins son els seus fills. Aquets, per tant, també tenen estructura de node.
- Pare: cada node ha de mantenir el seu pare. El primer, el *root*, és null
- Color: pot ser vermell o negre. No sé què és ja que no ho hem fet servir
- Data: la informació que manté el node. En aquest cas, la paraula i el número de vegades que es repeteix. És de tipus RBData, la qual és a la vegada un altre *struct* que manté aquestes dos informacions.

Per comprovar que el nostre codi funcionava hem realitzat les següents proves:

- La primera va ser executar el codi "tal qual" amb el *pdf* de prova que et donen per provar la comanda *pdftotext*. No hi havia cap error de compilació ni execució, així que vam procedir a comprovar que tot es realitzava bé.
- Per fer-ho, vam realitzar un *pdf* amb:
    - Una paraula amb accent
    - Una paraula del tipus 123Blai
    - Una paraula repetida (text)
    - Paraules amb signes de puntuació
- Vam realitzar un petit codi abans d'eliminar l'arbre i abans del acabament de l'aplicació que buscava el Node de la paraula "text" i imprimia la seva variable *num_vegades*, la qual havia de ser 2 si tot anava bé. Com que no funcionava bé, vam fer...
- Imprimir en la funció "extra" la paraula que se l'hi passava i si actualitzava el Node en cas de tenir-la repetida o si en creava un de nou en cas contrari. Així, vam veure que només guardava correctament la primera paraula, per culpa d'un parell d'errors.
- Un cop solucionats i comprovat el seu correcte funcionament, vam tornar a aquell petit codi que buscava per la paraula *text* i imprimia el número de vegades que sortia en el *pdf*. Allà, vam observar finalment imprimia "2" i que per tant tot funcionava com ho havia de fer. Aquest codi l'hem deixat comentat en l'entrega.

```
//Funcio per comprovar el funcionament del codi. B
RBData *treeData;
treeData = findNode(tree,"text");
if (treeData != NULL) {
    int f = (int) treeData->num_vegades;
    printf("%d\n",f);
    } else {
    printf("Element no trobat\n");
}
```