Université Libanaise
Faculté des Sciences 1

Programmation Impérative II

Durée: 1 mois

Titre du projet : Dictionnaire

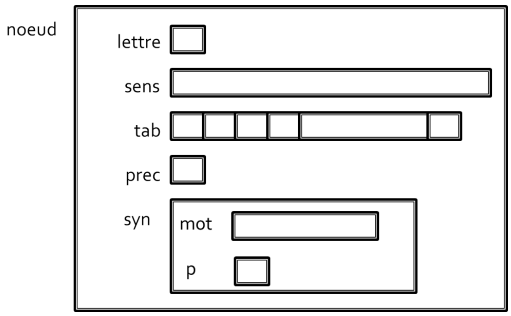
Sujet

Ce mini-projet vise à réaliser un dictionnaire anglais qui inspire largement sa forme d'un arbre n-aires (n=26) où les nœuds représentent les lettres composant les différents mots. Un exemple est fourni en fin de document.

Types utilisés

1. type dictionnaire (dico):
   - tableau de 26 pointeurs vers nœud (car on a 26 lettres dans l'alphabet).
2. type noeud contient:
   - champ lettre.
   - champ prec (pointeur vers le nœud précèdent; les nœuds pointés directement par les pointeurs de t de dico valent NULL).
   - champ tab: tableau de 26 pointeurs vers nœud.
   - champ sens: chaine de caractères (tableau de 200 caractères) ; sens du mot se terminant sur ce nœud, sinon chaine vide "\0".
   - champ syn: variable de type synonyme
3. type synonyme
   - champ mot (20 caractères)
   - champ p pointeur vers le nœud du dictionnaire qui contient le sens de ce synonyme, c.-à-d. le nœud correspondant à la dernière lettre de ce mot).

Ci-dessous une représentation graphique des différents champs constituant un nœud :



Remarque :

D'autres types de données peuvent être définis en vue de leur nécessité.

Fonctions demandées

1. creerDico : crée et retourne un dictionnaire vide dont le tableau de pointeur est initialisé à NULL partout.
2. existe : fonction qui vérifie l'existence d'un mot donné dans un dictionnaire donné. Noter bien qu'il ne suffit pas de trouver tous les nœuds correspondants au lettres successives de ce mot pour dire qu'il existe, mais il faut, en plus, s'assurer que le dernier nœud contienne un sens (chaine non nulle), sinon son existence revient au fait qu'il soit un préfixe d'un autre mot dans le dictionnaire. Si le mot existe dans le dictionnaire (a un sens), cette fonction retourne un pointeur vers le nœud contenant le sens (nœud correspondant à la dernière lettre de ce mot), sinon la fonction retourne NULL.
3. ajouterMot : ajoute un mot donné à un dictionnaire donné (création et lien des différents nœuds correspondants et remplissage du sens donné de ce mot dans le dernier nœud). Retourne 1 en cas de réussite et 0 en cas d'échec (échec : mot existe déjà c.-à-d. il a déjà un sens dans le dictionnaire). Évidemment, cette fonction s'en sert de la fonction existe.
4. changerSens : change le champ sens correspondant à un mot qui existe déjà dans le dictionnaire.
5. rechercher : fonction qui recherche et retourne le sens d'un mot dans le dictionnaire. Évidemment, cette fonction s'en sert de la fonction existe.
6. getMot : fonction qui, à partir d'un pointeur vers un nœud dans le dictionnaire, retourne le mot (chaine de caractères) se terminant par ce nœud. (Utiliser le pointeur prec dans chaque nœud pour faire un parcours en arrière).
7. setSyn : fonction qui prend en paramètre deux mots déjà existants dans le dictionnaire et indique (met à jour la valeur dans le champ syn) que chacun d'eux est le synonyme de l'autre.
8. effacerMot : fonction qui efface un mot du dictionnaire. Dans ce cas, il faut libérer tous les nœuds utilisés auparavant uniquement pour ce mot. (Avant de libérer chaque nœud, vérifier qu'il ne fait pas partie d'autre mots dans le dictionnaire, qui eux doivent continuer à exister. Il faut aussi supprimer le lien de synonymie.
9. imprimer: fonction qui affiche tous les mots du dictionnaire avec leurs sens, et leurs synonymes éventuels. Cet affichage doit se faire dans l'ordre alphabétique (ordre naturel du dictionnaire).
10. liberer : fonction qui libère le dictionnaire entier de la mémoire tas.
11. sauvegarder: qui écrit le dictionnaire entier dans un fichier XML, dans le but de la charger plutard.
12. charger : qui reconstruit un dictionnaire à partir d'un fichier XML.

Astuce

Pour savoir l'indice de la case correspondant à une lettre dans le tableau de pointeur, il suffit de faire indice = lettre - 'a'. Par exemple l'indice de la case correspondante à la lettre 'c' est indice='c'-'a'=2, car les indices commencent par 0.

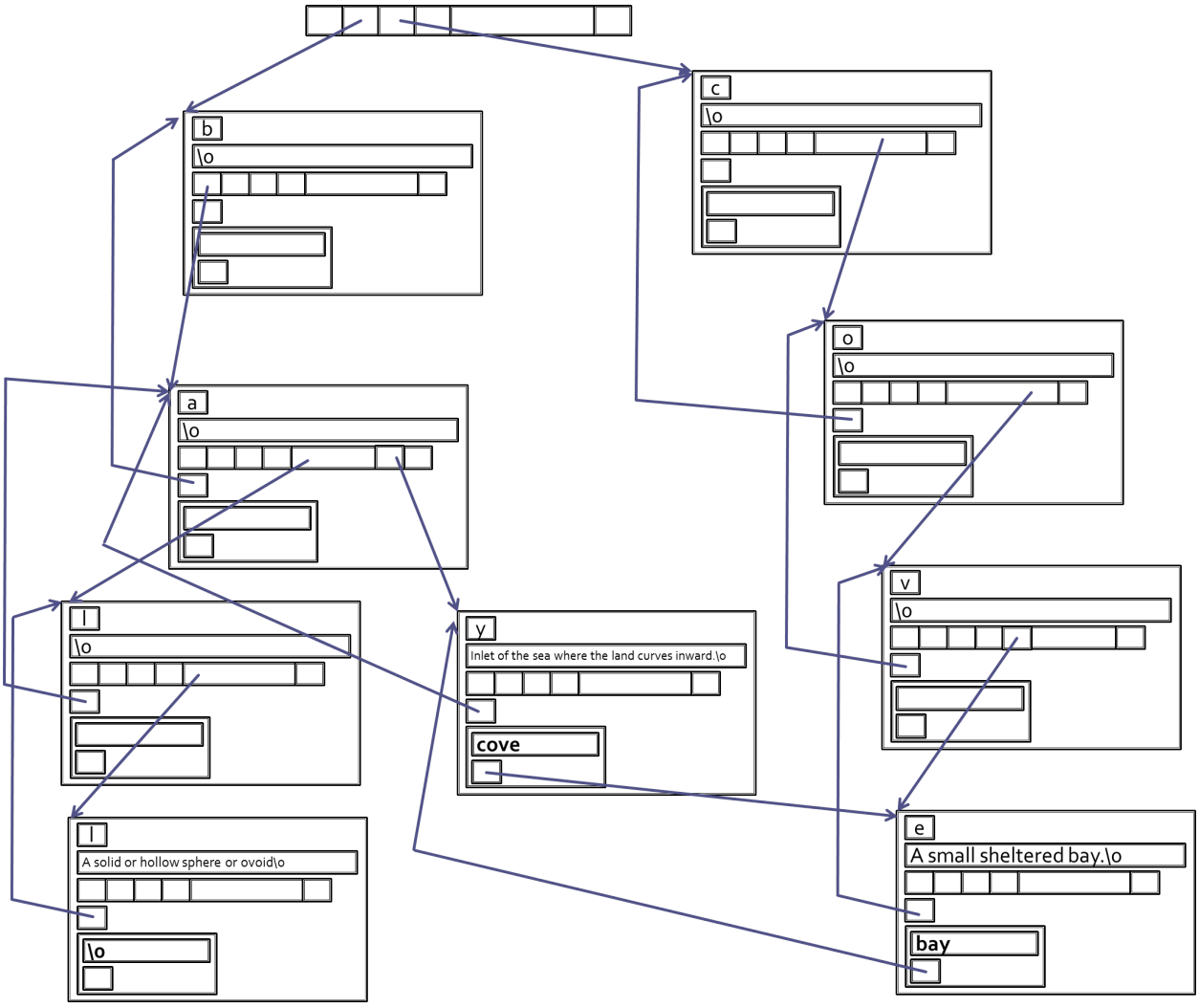
Exemple détaillé

Ci-dessous, une représentation graphique d'un dictionnaire contenant 3 mots :

- ball : A solid or hollow sphere or ovoid.
- bay : Inlet of the sea where the land curves inward. (syn : cove)
- cove : A small sheltered bay.(syn : bay)

On donne ainsi le contenu du fichier XML représentant ce dictionnaire :

<dico> <a> <l> <l> <sens> A solid or hollow sphere or ovoid. </sens> </l> </l> <y> <sens> Inlet of the sea where the land curves inward. </sens> <syn> cove </syn> </y> <c> <o> <v> <e> <sens> A small sheltered bay. </sens> <syn> bay </syn> </e> </v> </o> </c> </dico>



Remise des projets

Le projet doit etre reparti sur plusieurs fichiers

- dico.h, dico.c
- menu.h, meu.c
- test.h, test.c
- etc
- et finalement mainfile.c contenant la fonction main, cette derniere contient un appel unique vers le menu deroulant

Chaque fonction dans dico.c doit

- être précédée d'un commentaire de deux lignes décrivant son travail, ce qu'elle prend en entrée et ce qu'elle retourne en sortie
- avoir une fonction de test dans test.c
- chaque fichier doit avoir un entête obligatoire du style :

```
/*************************************************************************
*   name            :   Siba Haidar                                      *
*   id              :   69000                                            *
*   email           :   siba.haidar@gmail.com                            *
*   tel             :   03-123123                                        *
*   date created    :   june 2009                                        *
*   last modified   :   6/6/2012                                         *
*   purpose         :   simulate bank accounts and transactions          *
*                       with date manipulation                           *
*                       and options to save/load in files                *
*   method and types:   linked list for accounts and                     *
*                       another sorted linked list for transactions      *
*   file types use  :   binary                                           *
*   number of files :   7                                                *
*   filename        :   banking.c1                                       *
*   all file names  :   banking.c, bank.h, bank.c, dates.h, dates.c      *
*                       menu.h, menu.c                                   *
*************************************************************************/

#include "menu.h"

void main(){
    doMenu();
}
```