Projet 1A : GESTION DE PRÊTS DE LIVRES

IUT CAEN – Département Informatique

November 18, 2011

1 Description et données manipulées

On se propose dans ce projet de réaliser en langage C une mini application permettant de gérer les prêts de livres dans une bibliothèque. Pour le faire, vous manipulerez des structures de différentes natures, dont les champs contiennent à minima les informations données ci-dessous. Vous devrez sans doute ajouter des champs dans ces structures en fonction des choix que vous avez retenu.

1. **Les adhérents**. Un adhérant est caractérisé par :
   - son numéro (`adh_num`) qui identifie de manière unique un adhérent. **Il vous appartient de gérer la notion de numéro unique**.
   - son nom (`adh_nom`),
   - son adresse (`adh_adresse`). Une adresse est constituée par :
     - un code postale (`adr_cp`),
     - un nom de la ville (`adr_ville`),
     - un nom de la rue (`adr_rue`).
   - le nombre de livres empruntés (`adh_nb_emprunts`). Un adhérent peut donc emprunter plusieurs livres.

   Pour représenter les adhérents en C, vous utiliserez les structures suivantes :

```
typedef struct {
  int  adr_cp;
  char adr_ville[50];
  char adr_rue[50];
} adresse_t;

typedef struct {
  int       adh_num;
  char      adh_nom[50];
  adresse_t adh_adresse;
  int       adh_nb_emprunts;
} adherent_t;
```

2. **Les livres**. Un livre est caractérisé par :
   - un numéro (`liv_num`) qui identifie de manière unique un livre. **Il vous appartient de gérer la notion de numéro unique**.
   - un titre (`liv_titre`),
   - un auteur (`liv_auteur`), contenant les champs ci-dessous :
     - un nom (`nom`),
     - un prénom (`prénom`).
   - un numéro contenant la valeur de `adh_num` de l'emprunteur du livre (`liv_emprunteur`).

Pour représenter les livres en C, vous utiliserez la structure suivante :

```
typedef struct {
  int  liv_num;
  char liv_titre[50];
  struct {
    char nom[50];
    char prenom[50];
  }   liv_auteur;
  int liv_emprunteur;
} livre_t;
```

Structures d'implémentation

La fonction `main()` devra contenir les deux structures de travail ci-dessous (ce ne seront pas des variables globales) :

- un tableau de structures `adherent_t` : `adherent_t adherents[MAX_ADHR];`
- un tableau de structures `alivre_t` : `alivre_t livres[MAX_LIVRES];`

Vous devrez enregistrer les données de ces structures dans deux fichiers. Pour cela, vous déclarerez deux variables globales :

- une variable pointeur `f_adhr` ayant pour type `FILE` et dirigée sur le nom du fichier adhérents `fich_adhr`;
- une variable pointeur `f_livres` ayant pour type `FILE` et dirigée sur le nom du fichier livres `fich_livres`;

Les noms des fichiers adhérents et livres devront être saisies par l'utilisateur.

2 Fonctionnalités à implémenter

Votre programme devra proposer à l'utilisateur un menu permettant de naviguer dans les différentes fonctionnalités offertes par votre programme. En particulier, votre menu devra aider à la saisie des informations requises pour la gestion des adhérents, des livres et des emprunts (voir ci-dessous). Pour l'algorithme de tri, vous devez utiliser la fonction `qsort()`.

(1) Gestion des adhérents

- Ajouter, modifier ou supprimer un adhérent
- afficher la liste des adhérents par ordre alphabétique (`adh_num`)
- rechercher un adhérent par son nom et affiche l'enregistrement correspondant
- retour menu principal

(2) Gestion des livres

- Ajouter, modifier ou supprimer un livre
- afficher la liste des livres par ordre alphabétique ((`liv_titre`)
- rechercher le titre d'un livre et affiche l'enregistrement correspondant
- retour menu principal

(3) Gestion des emprunts

- Emprunter un livre
- Afficher la liste des livres empruntés
- Rendre un livre
- Afficher la liste des emprunteurs de livres
- retour menu principal

(4) Quitter le programme

3 Notes de mise en œuvre

Ci-dessous une proposition d'implémentaion du code de la fonction `main()` :

```
int main (void)
{
  livre_t    livres[MAX_LIVRES];
  adherent_t adherents[MAX_ADHR];
  int n_livres = 0;
  int n_adhr   = 0;
  char choix;
  char fich_livres[30];
  char fich_adhr[30];

  do {
    printf("Entrez le nom du fichier de livres  charger: -> ");
    ...
    printf("Entrez le nom du fichier d'adhrents  charger: -> ");
    ...
  } while (!ouvrir_fichiers(fich_livres, fich_adhr));

  n_adhr   = charger_adherents(adherents);
  n_livres = charger_livres(livres);

  while (choix != '4') {
    choix = lire_menu();
    switch (choix) {
    case '1':
      gestion_adhr(adherents, &n_adhr, livres, &n_livres);
      break;
    case '2':
      gestion_livres(livres, &n_livres);
      break;
    case '3':
      gestion_emprunts(adherents, n_adhr, livres, n_livres);
      break;
    case '4':
      fin_prog(adherents, n_adhr, livres, n_livres);
      break;
    default:
      break;
    }
  }

  return 0;
}
```

Ci-dessous une déscription de quelques fonctions permettant d'implémenter les principales fonctionnalités demandées :

1. Une fonction `char lire_menu(void)` qui permet de lire le choix de l'utilisateur. Dès que celui-ci appuie sur un autre choix autre que celui défini par le menu, la fonction doit produire un signal sonore.

2. Une fonction `int ouvrir_fichiers(char* livres, char* adhr)` qui permet d'ouvrir les fichiers des adhérents et livres passés en paramètre. La fonction devra retourner 1 si les fichiers ont été ouverts correctement, 0 sinon.

3. Une fonction `int charger_adherents(adherent_t* adherents)` qui charge le contenu du fichier associé aux adhérents.
4. Une fonction `int int charger_livres(livre_t* livres)` qui charge le contenu du fichier associé aux livres.
5. Une fonction `void affiche_livre(livre_t* livre)` qui permet de visualiser dans un format agréable les informations contenus dans l'enregistrement `livre`.
6. Une fonction `void affiche_adherent(adherent_t* adhr)` qui permet de visualiser dans un format agréable les informations contenus dans l'enregistrement `adhr`.
7. Une fonction `void gestion_adhr(adherent_t* adhr, int* n_adhr, livre_t* livres, int* n_livres)` qui gère les fonctionnalités liées aux adhérents :
   (a) `void ajout_adherent(adherent_t* adhr, int* n);`
   (b) `void modif_adherent(adherent_t* adhr, int n);`
   (c) `void supprime_adherent(adherent_t* adhr, int* n, livre_t* livres, int* n_livres);`
   (d) `void liste_adherents(adherent_t* adhr, int n));`
   (e) `void recherche_adherent(adherent_t* adhr, int n));`
8. Une fonction `void gestion_livres(livre_t* livres, int* n_livres)` qui gère les fonctionnalités liées aux lives :
   (a) `void ajout_livre(livre_t* livres, int* n);`
   (b) `void modif_livre(livre_t* livres, int n);`
   (c) `void supprime_livre(livre_t* livres, int* n);`
   (d) `void liste_livres(livre_t* livres, int n);`
   (e) `void recherche_livre(livre_t* livres, int n);`
9. Une fonction `void gestion_emprunts(adherent_t* adhr, int n_adhr, livre_t* livres, int n_livres)` qui gère les fonctionnalités liées aux emprunts :
   (a) `void emprunter(adherent_t* adhr, int n_adhr, livre_t* livres, int n_liv);`
   (b) `void liste_emprunts(const livre_t* livres, int n);`
   (c) `void rendre_livre(livre_t* livres, int n, adherent_t* adhr, int n_adhr);`
   (d) `void liste_emprunteurs(adherent_t* adhr, int n_adhr);`
10. Une fonction `void fin_prog(adherent_t* adhr, int n_adhr, livre_t* livres, int n_livres)` qui sauvegarde le contenu des tableaux `adhr` et `livres` dans les fichiers associés. Cette fonction devra refermer tous les fichier ouverts avant de quitter le programme.

La *suppression* d'un élément du tableau implique de décaler la "case libre" vers la fin du tableau afin de supprimer les "trous" du tableau. Tous les tableaux doivent être passés par paramètre de la manière suivante :

- Si la fonction ne modifie pas le nombre **nb** d'éléments présents dans le tableau **T** dont le nombre maximal est **taille**, elle devra être déclarée comme suit :
  `<type> fonctionUtiliseTableau (<type> T[ ], int nb, int taille) ;`
- Si la fonction modifie le nombre, pointé par **pnb**, d'éléments présents dans le tableau **T** dont le nombre maximal est **taille**, elle devra être déclarée comme suit :
  `<type> fonctionModifieTableau (<type> T[ ], int *pnb, int taille) ;`
  Au retour de la fonction, ***pnb** donne le nouveau nombre d'éléments présents.

4 Travail à rendre

Le projet est à réaliser en binômes (ou, avec l'accord de votre chargé de TP, en monôme). Le travail à rendre est un projet sous forme d'une archive zip **à envoyer par mail à votre chargé de TP**.

Le nom de l'archive doit avoir la forme suivante : `Nom1Nom2.grTP.zip` ou `Nom1.grTP.zip` où `Nom1` et `Nom2` sont les noms de famille des membres des polynômes et `grTP` est le nom du groupe de TP auquel ils appartiennent (1.1, 1.2, etc.).

Note: Lors des séances de TP, les chargés de TP suivront l'avancement de votre projet et pourront vous aider sur certains points difficiles.

Ce qu'il faut rendre :

- Le code source complet de votre application en C largement commenté.
- Un exécutable testé **sous Linux** et opérationnel avec sa documentation d'installation.

  ATTENTION : si votre programme fait appel à d'autres librairies externes (comme par exemple les librairies du SDL), il est impératif de les inclures pour les besoins de test.
- Un court rapport d'une longueur comprise entre 3 et 10 pages présentant :
  - les fonctionnalités implémentées (très brièvement).
  - organisation du programme : découpage en fonctions, rôle de ces fonctions, explications du programme.
  - l'organisation et la répartition des tâches au sein du binôme durant la durée du projet (brièvement).
  - bilan qualitatif du travail, difficultés rencontrées, points qui vous ont paru intéressants.
  - un mode d'emploi avec quelques illustrations (p. ex. capture d'écrans, scénario d'exécution...), destinées à montrer l'opérationnalité de votre application.
  - Une conclusion sur l'apport (ou non) du projet en termes technique, scientifique, humain.

  Le code source ne doit pas faire partie du rapport d'une dizaine de pages (sinon en annexe).

5 Évaluation du projet et calendrier

Le projet est à rendre le **lundi 02 janvier 2012** (tout retard conduira à des pénalités). L'évaluation sera réalisée en focntion des critères ci-dessous :

- *qualité technique du code* : découpage en fonctions, modularité et réutilisabité, instructions, algorithmes, efficacité, gestion des erreurs lors de d'une saisie, ou lorsque les données fournies en paramètres des fonctions sont incorrectes.
- *lisibilité du code* : présentation du programme (indentation), usage de variables ayant des noms explicites, commentaires pour préciser les points difficiles dans les algorithmes, paramètres des fonctions ...
- *documentation fournie* : organisation du programme et son mode d'emploi, bilan.
- *présentation orale* : démonstration du programme et questions sur le travail réalisé.