CHOIX BASE DONNEE CSV

Nous avons choisi avec mon camarade de réaliser la base de données à l'aide fichier CSV. Les fichiers CSV permettent de représenter des données sous forme de valeurs séparées par des virgules ou des points virgules. Nous avons choisi de créer deux fichiers pour représenter notre base de données : un fichier adhérent et un fichier livre. Ces fichiers, une fois créés manuellement et formatés de façon à ce qu'ils soient compréhensibles, sont ouverts ici de deux façons différentes : une pour la saisie des nouveaux éléments de la base de données et une pour la recherche et la suppression des éléments déjà existant.

Voyons de quelle façon cela est fait dans le programme :

1) fstream fichier_adherent("adherent.csv", ios::in | ios::out | ios::ate);
2) fichier_adherent = fopen("adherent.csv", "a");

Le premier extrait de code est une ouverture propre au C++, elle va ouvrir un flux vers le fichier adherent.csv, dans le but de lire et d'écrire à l'intérieur. Le « ios :: ate » signifie également que l'on place le curseur à la fin du fichier. Un autre moyen d'ouvrir un fichier avec fstream est de remplacer « ios :: ate » par « ios :: trunc » afin de vider le fichier avant d'écrire dedans.

Le second extrait consiste à ouvrir un flux vers le fichier adherent.csv à l'aide d'un fichier de type FILE * (ici, fichier_adherent). Ici, on ouvre le fichier en « a » ce qui correspond à un mode d'écriture à partir de la fin du fichier. Dans notre projet, nous avons utilisé deux autres modes d'ouverture : le « r » qui permet la lecture du fichier mais pas l'écriture, et le « w + » qui permet de lire et écrire le contenu du fichier tout en le vidant préalablement. Pour ce second extrait de code, il faut ajouter un fclose(FILE *) à la fin de notre fonction afin de fermer le flux et d'éviter les fuites de mémoires.

AFFICHAGE HEURE/DATE

Afin d'afficher l'heure et la date dans notre programme, nous avons inclus la bibliothèque <time.h>. Grâce à cette bibliothèque, nous avons pu récupérer l'heure et la date actuelle de l'ordinateur. Puis, nous avons permis l'affichage grâce à une fonction de <time.h> :

strftime(format, 128, "Date : %A %x\n Heure : %X \n Vous etes a : Paris\n\n\n", &date);

Afin que les utilisateurs puissent être au courant de l'heure à tout moment, l'heure est réaffichée à chaque changement de menu.

IDENTIFICATION PAR MOT DE PASSE

Pour éviter que tous les utilisateurs aient accès à la possibilité de modifier les bases de données, nous avons choisi d'installer un mot de passe à l'exécution du programme. Ainsi, lorsqu'un utilisateur lance le programme, il aura à saisir un mot de passe afin de déterminer ce qu'il pourra faire dans la suite du programme. Le mot de passe utilisateur est donné. Le mot de passe bibliothécaire ne doit être connu que des personnes devant avoir accès aux possibilités de modifier les bases de données.

Pour vérifier si la saisie de l'utilisateur correspond à l'un des mots de passes, le programme réalise une saisie d'une chaine de caractère puis la compare aux deux mots de passes connus par le programme à l'aide d'une fonction contenue dans la bibliothèque <string.h>, la fonction strcmp. Cette fonction prend deux chaînes de caractères en paramètres puis retourne un entier. Cet entier vaut 0 si les deux chaînes sont identiques. Ainsi, voilà ce que donne la boucle permettant de sécuriser l'accès au programme par un mot de passe :

```
do
  {
     scanf("%s", mdp);
     verif1 = strcmp(mdp, mdp_biblio);
     verif2 = strcmp(mdp, mdp_user);


  }
  while (verif1 != 0 && verif2 != 0);
```

Une fois cette vérification faite, le programme va envoyer l'utilisateur sur le menu correspondant à ce qu'il a l'autorisation de faire.

CREATION D'UN MENU

Afin d'avoir un programme clair et simple à utiliser, y compris pour un utilisateur novice, nous avons choisi de réaliser un menu où les choix seront réalisés à l'aide de saisie de chiffres.

Ainsi, après avoir rentré un mot de passe, l'utilisateur va avoir accès à l'un des deux menus que nous avons défini. Le menu utilisateur ne comprend que deux possibilités : rechercher un livre ou quitter le programme :

```
printf("Que puis-je fais pour vous ?\n");
printf("Rechercher un livre ? (1) \n");
printf("Quitter le programme ? (2) \n");
```

En fonction du choix de l'utilisateur, le programme va lancer la fonction de recherche de livres ou quitter le programme.

Le bibliothécaire, lui, a plus de choix. En effet, il pourra choisir d'accéder à la base de données des adhérents, la base de données des livres ou de quitter le programme :

```
  printf("Que puis-je fais pour vous ?\n");
  printf("Acceder a la base de gestion des adherents ? (1) \n");
  printf("Acceder a la base de gestion des livres ? (2) \n");
  printf("Ou quitter le programme ? (3) \n");
```

Cependant, là où le choix de la recherche de livre par un utilisateur exécute directement la fonction, ici, si le bibliothécaire choisit d'accéder à l'une des deux bases de données, un sous-menu s'ouvre. Ces deux sous-menus sont présentés de la même façon à l'utilisateur :

```
printf("Ajouter/Supprimer un adherent ? (1) \n");

printf("Rechercher un adherent ? (2) \n");

printf("Ou retourner à l'accueil ? (3) \n");
```

La différence entre les deux sous-menus résidera dans ce que l'on trouve après le « un » dans les deux premières lignes : ce sera « adhérent » si le bibliothécaire a choisi d'accéder à la base de données des adhérents et « livre » s'il a choisi d'accéder à celle des livres. Ici, si le bibliothécaire choisit l'option de retourner à l'accueil, le menu d'accueil se réaffichera (sans la demande de mot de passe, celui-ci ayant déjà confirmé l'identité de l'utilisateur). Sinon, nous verrons dans une autre partie ce qui sera entraîné par le choix de l'utilisateur.

UTILISATION DE STRUCTURES

Dans ce programme, afin de simplifier le code et d'éviter la multiplication de variable, nous avons choisi d'utiliser des structures. Deux d'entre elles sont intéressantes puisqu'elles sont composées de champ ayant des types structurés. Ainsi, pour représenter un adhérent, nous utilisons la structure suivante :

```
structure t_adherent

{

   t_adresse_postale adresse_adherent;

   char nom[50];

   char prenom[50];

   char adresse_mail[50];

   char profession[50];

};
```

Dans cette structure, on peut remarquer que l'on utilise des tableaux de caractères statiques pour stocker les informations pour que l'on ne multiplie pas les lignes dans le code, laissant ainsi le programme plus lisible et digeste.

L'adresse est quant à elle de type structurée afin que notre structure adhérent ne soit pas trop longue. Voilà la structure adresse :

```
structure t_adresse_postale

{

   char rue[50];

   int numero_rue;
```

```
   int code_postal;
   char ville[50];
   char pays[50];
};
```

On peut noter encore une fois que nous utilisons principalement des tableaux de caractères statiques (pour la même raison). Cependant, ici, un seul champ récupère l'intégralité des informations sur la rue. Pour que ce champ soit récupéré correctement, il faut que les espaces dans l'adresse soient remplacés par des « _ ».

La seconde structure intéressante de programme est la structure permettant de définir un livre :

```
structure t_livre
{
   char titre[50];
   t_auteur auteur;
   t_code_livre code_livre;
   int nb_total;
   int nb_disponible;
};
```

On remarque ici que deux des champs sont de type structuré. Le type t_auteur est une structure composé de deux tableaux de caractères statiques permettant de stocker le nom et le prénom de l'auteur. Le type t_code_livre, lui est composé d'un tableau de 3 caractères et d'un entier qui va permettre de stocker le code d'un livre comme demandé dans le sujet, sous la forme : XXX (3 lettres représentant le type du livre : ROM, BDE etc …) – YYY (3 chiffres qui représente le numéro de l'ouvrage relativement à son thème). Maintenant, voyons comment ces structures sont utilisées pour réaliser un ajout dans l'une des bases de données.

AJOUT/SUPPRESSION LIVRE/ADHERENT

Ici, nous allons voir ce qui se passe lorsque que le bibliothécaire a choisi de réaliser un ajout ou une suppression dans l'une des bases de données.

Tout d'abord, un sous-menu se présente à lui, lui donnant le choix entre l'ajout et la suppression :

```
printf("Voulez-vous faire un ajout ou une suppresion ? \n");
printf("Ajout = 1 \n");
printf("Suppression = 2 \n");
```

Voyons tout d'abord ce qui se passe si l'utilisateur choisit de réaliser un ajout. Plaçons-nous dans le cas où il désire réaliser l'ajout d'un livre.

Le programme va tout d'abord demander à l'utilisateur la quantité de livres qu'il tient à rentrer. Puis, une fois cette quantité rentrée, le programme va demander à l'utilisateur de rentrer les données de chacun des livres qu'il désire ajouter.

Une fois une information récupérée, le programme l'écrit dans le fichier CSV.

Voilà la façon dont cette récupération et cette écriture se présente dans le programme :

```
printf("Creation du livre %d \n \n", cpt + 1);
printf("Saisie du titre : ");
scanf("%s", livre.titre);
printf("\n");
fprintf(fichier_livre, "%s", livre.titre);
fprintf(fichier_livre, ";");
```

On répète cette opération pour saisir l'intégralité des informations dont on a besoin pour initialiser un livre. Puis on recommence autant de fois que nécessaire pour initialiser le nombre de livres que désire rentrer l'utilisateur. De plus, à chaque nouvel ajout, dans un fichier texte, on tient le compte du nombre d'ouvrage différents que l'on trouve dans la base de données :

```
nb_livre = fopen("livre.txt", "r");
fscanf(nb_livre, "%d", &nb_actuel);
fclose(nb_livre);
nb_livre = fopen("livre.txt", "w+");
fprintf(nb_livre, "%d", cpt + nb_actuel);
fclose(nb_livre);
```

Ici, on récupère le contenu actuel du fichier texte, puis on rouvre le fichier en vidant son contenu afin d'y écrire la somme du nombre de livres ajoutés et du nombre d'ouvrages précédent.

Intéressons-nous maintenant à la suppression d'une entrée.

Tout d'abord, nous allons rechercher l'entrée dans la base de données (à l'aide de la fonction recherche que nous verrons après), on recherche donc directement dans le fichier CSV.

Une fois l'entrée trouvée dans le fichier CSV, on demande au bibliothécaire s'il est sûr de vouloir supprimer l'entrée (ici, un adhérent) :

```
cout << "\n" << "Voulez-vous vraiment supprimer l'adhérent : ";
cout << "(1 = oui)" << "\n" << "(2 = non)" << "\n";
cin >> i;
```

Si sa réponse est non, le programme renvoie à l'accueil du sous-menu de la base de données adhérent. Toutefois, si le bibliothécaire veut supprimer l'entrée, on place le curseur au début de la ligne où se trouvent les informations sur l'adhérent, puis, on remplace chacune des données par un espace :

```
taille = ligne.size() ;
fichier_livre.seekp(- taille, ios::cur);
fichier_livre << « » << ";" << « » << ";" << « » << ";" << « » << ";" << « » << ";";
```

Enfin, au fichier texte contenant le nombre d'adhérent, on retire 1. Cela se fait de la même manière que la mise à jour du nombre d'adhérents (ou d'ouvrage, présenté plus haut).

Ainsi, dans le CSV, la ligne où se trouvait l'adhérent a été intégralement effacée.

Recherche de livres

Cette partie a été réalisée à l'aide du C++, étant donné que nous n'avons pas réussi à la réaliser en langage C, le C++ amenant des facilités non négligeables dans la manipulation et la récupération de données dans un fichier .CSV.

En premier lieu, l'utilisateur va rentrer l'entrée qu'il veut que le programme cherche et on rentre cette entrée dans un getline :

```
cout << "\n" << "\n" << "Entrez le nom du livre, ou son auteur : ";
getline(cin, recherche);
```

Ce getline est une fonction qui permet de lire l'intégralité d'une ligne et donc, de les comparer.

Après cela, on place le curseur au début du fichier .CSV.

Puis, le programme va parcourir l'ensemble du fichier jusqu'à sa fin, ligne par ligne, en vérifiant à chaque ligne si l'entrée de l'utilisateur se trouve dans la ligne actuelle. Pour cela, on utilise une constante : std :: string :: npos. Cette constante, qui est de type unsigned integral, vaut -1 (ce qui correspond à la plus grande valeur possible pour une valeur de type unsigned integral). Si jamais le programme trouve l'entrée dans la ligne, cette constante prendra la même valeur que la position de l'entrée dans la ligne. Ainsi, si, dans une ligne, on trouve l'entrée que l'on recherche, on aura une valeur différente de -1.

Dans ce cas, on indique à l'utilisateur que l'on a trouvé son entrée dans la base de données, puis, on affiche l'intégralité des informations contenues à cette ligne :

```
while(!fichier_livre.eof())
{
getline(fichier_livre, ligne);
if (ligne.find(recherche) != std::string::npos)
{
if (choix == 0)
{
cout << "\n" << "L'ouvrage fait partie de cette bibliotheque" << "\n";
i = 1;
ligne_entiere = ligne;
system("pause");
std::cout <<"Il se trouve a la ligne : " << ligne_entiere << std::endl;
}
}
```