

Cours de programmation séquentielle - TP noté 1

Annuaire rudimentaire

08.11.2022, rendu le 22.11.2022 (voir section modalités)

Buts

Dans ce travail vous verrez les concepts suivants:

- Manipulation de chaînes de caractères.
- Implémentation d'une interface en ligne de commande.
- Manipulation de tableaux statiques
- Implémentation d'algorithmes simples (recherche dichotomique et tri par sélection)

Énoncé

Le but est d'écrire un programme permettant de manipuler un annuaire rudimentaire. La structure de données considérée est (presque) la même que celle présentée dans l'exercice 7. Vous pouvez utiliser le code de cet exercice comme base pour ce TP.

Par rapport à l'exercice 7, vous devrez essentiellement ajouter des fonctions de tri et de recherche dans l'annuaire, ainsi qu'un programme principal doté d'une interface en ligne de commande.

Fonctionnement du programme

Lors de son lancement, le programme allouera et initialisera un annuaire avec un ensemble de données. Ensuite, un message invitant l'utilisateur à saisir une commande apparaîtra. L'utilisateur pourra alors choisir quelle action effectuer, et le programme réagira en fonction de cela.

Les commandes à implémenter sont :

- **add** : permet d'ajouter une entrée à l'annuaire. Lorsque l'utilisateur choisit cette option, le programme lui demande le nom, prénom, date de naissance et numéro de téléphone de la nouvelle entrée et ajoute ce nouvel élément à l'annuaire.
- **del** : permet de supprimer une entrée de l'annuaire. Lorsque l'utilisateur choisit cette option, le programme lui demande l'indice de l'élément à supprimer et le supprime de l'annuaire.
- **show** : permet d'afficher une ou toutes les entrées de l'annuaire. Lorsque l'utilisateur choisit cette option, le programme lui demande quelle entrée afficher. Si l'utilisateur entre un nombre, l'entrée à cette indice est affichée. Si l'utilisateur entre **all**, tout l'annuaire est affiché.
- **sort** : permet de trier l'annuaire selon le nom de famille ou le numéro de téléphone. Lorsque l'utilisateur choisit cette option, le programme lui demande quel type de tri appliquer (sur les noms ou les numéros de téléphone). L'annuaire n'est pas automatiquement réaffiché après (il faut utiliser la commande **show** pour voir le nouvel état de l'annuaire). Utiliser le tri par sélection vu en cours pour cette fonctionnalité.
- **search** : permet de rechercher une entrée dans l'annuaire selon le nom de famille ou le numéro de téléphone. Lorsque l'utilisateur choisit cette option, le programme lui demande quel type de recherche effectuer, puis le nom ou le numéro de téléphone à chercher. Pour faire la recherche, triez l'annuaire selon le nom de famille ou le numéro de téléphone et effectuez une recherche dichotomique. Affichez ensuite l'entrée trouvée, ou un message indiquant qu'aucune correspondance n'a été trouvée. Vous pouvez ignorer les cas où des doublons existent (n'affichez qu'une entrée valide).

— quit : quitte le programme.

Exemple d'utilisation

```
$ ./annuaire
available commands : add, del, show, sort, search, quit
> show
enter the index to show or all if you want to show entire directory
all
[0] Jean Paul, 18-12-1967, 12346890
[1] Arthur Rimbaud, 1-1-2002, 73645
[2] Paul Albuquerque, 23-6-1902, 28736474
> del
enter the index to remove : 2
> show
enter the index to show or all if you want to show entire directory
all
[0] Jean Paul, 18-12-1967, 12346890
[1] Arthur Rimbaud, 1-1-2002, 73645
> add
enter the name : Bob
enter the surname : Bib
enter the day of birth : 1
enter the month of birth : 2
enter the year of birth : 3
enter the phone number : 6754
> sort
which type of sort ? (surname or tel) : surname
> show
enter the index to show or all if you want to show entire directory
all
[0] Bob Bib, 1-2-3, 6754
[1] Jean Paul, 18-12-1967, 12346890
[2] Arthur Rimbaud, 1-1-2002, 73645
> search
which type of search ? (surname or tel) : tel
enter the phone number to search : 73645
Arthur Rimbaud, 1-1-2002, 73645
> quit
$
```

De plus, si une commande inconnue est entrée, le programme réaffichera le message indiquant les commandes disponibles ainsi que l'invite de commande. Si une erreur se produit dans le traitement d'une commande (par exemple ajout d'un élément dans un annuaire plein, ou option invalide choisie pendant l'exécution d'une commande), le message d'erreur générique "error while processing the command" sera affiché.

Structures de données

L'annuaire stocke pour chaque personne :

- Le prénom sous forme de chaîne de caractères;
- Le nom sous forme de chaîne de caractères;
- La date de naissance sous forme d'un type structuré;
- Le numéro de téléphone sous forme d'un nombre entier ¹.

Ainsi, une date est représentée par le type

1. Ce n'est évidemment pas la représentation idéale pour ce type d'information puisqu'il n'est pas possible de représenter les 0 au début du numéro. Mais cela conviendra pour l'exercice.

```
typedef struct {
    int day, month, year;
} date_t;
```

et une personne par le type²

```
typedef struct {
    char name[40];
    char surname[40];
    date_t birthday;
    int phone_number;
} person_t;
```

L'annuaire est représenté sous forme d'un tableau statique de personnes. Comme on travaille avec un tableau statique, sa capacité est fixe. La capacité de l'annuaire est définie à l'aide d'une macro DIR_CAPACITY (par exemple #define DIR_CAPACITY 1000). Le nombre d'éléments stockés dans l'annuaire peut cependant évoluer au fil de l'exécution du programme, ainsi, la taille réelle est stockée dans une variable liée au tableau. Un annuaire est représenté par le type :

```
typedef struct {
    int size;
    person_t content[DIR_CAPACITY];
} directory_t;
```

Fonctions à implémenter

Implémentez les fonctions de traitement d'annuaire suivantes. Attention ! La fonction `dir_delete_entry` retourne maintenant un code d'erreur, contrairement à l'énoncé de l'exercice 7.

```
// Prints a person record with the format "Name Surname, day-month-year, phone number"
// Parameter : p - the person
void person_print(person_t p);

// Prints the content of a directory each line with the format "[n] person"
// with n being the position in the directory
// Parameters : dir - the directory
void dir_print(directory_t *dir);

// Initialize a directory by setting its size to 0 and thus actually erasing its content
// Parameter : dir - the directory
void dir_init(directory_t *dir);

// Add a person to the directory if its capacity is not reached and increase its size
// Parameters : dir - the directory, p - the person
// Returns : 0 if success, 1 otherwise
int dir_add_entry(directory_t *dir, person_t p);

// Fill a directory with content, erasing any previous content
// Parameter : dir - the directory
// Returns : 0 if success, 1 otherwise
int dir_fill(directory_t *dir);

// Delete the entry at position n of the directory if n is smaller than
// the size of the directory, does nothing otherwise
// Parameters : dir - the directory, n - the position
// Returns : 0 if success, 1 otherwise
int dir_delete_entry(directory_t *dir, int n);

// Sort the directory based on surnames increasing lexicographical order
```

2. La limite de cette représentation est que chaque personne peut avoir un nom et un prénom d'au plus 40 caractères.

```

// Uses the selection sort algorithm
// Parameter : dir - the directory
void dir_sort_surname(directory_t *dir)

// Sort the directory based on phone numbers increasing order
// Uses the selection sort algorithm
// Parameter : dir - the directory
void dir_sort_tel(directory_t *dir)

// Searches for an entry with a given phone number
// This function starts by sorting the algorithm according to phone numbers and perform a dichotomic search
// The state of the directory is thus actually modified by the function
// Parameters : dir - the directory, phone_number - the phone number to search
// Returns : the index of a matching entry, a negative number if no match is found
int dir_search_tel(directory_t* dir, int phone_number)

// Searches for an entry with a given surname
// This function starts by sorting the algorithm according to surnames and perform a dichotomic search
// The state of the directory is thus actually modified by the function
// Parameters : dir - the directory, surname - the surname to search
// Returns : the index of a matching entry, a negative number if no match is found
int dir_search_surname(directory_t* dir, char* surname)

```

Vous devez respecter la signature de ces fonctions. Chaque fonction possède un commentaire de documentation définissant ce que doit faire la fonction. Le code d'erreur retourné par certaines fonctions est 0 en cas de succès et 1 en cas d'échec (par exemple si la capacité de l'annuaire est atteinte). Remarquez que l'annuaire est systématiquement passé par référence.

On présente ici une proposition d'implémentation pour la fonction `dir_fill` :

```

// Fill a directory with content, erasing any previous content
// Parameter : dir - the directory
// Returns : 0 if success, 1 otherwise
int dir_fill(directory_t *dir){
    dir_init(dir);
    if(dir_add_entry(dir, (person_t){ "Jean", "Paul", (date_t){18, 12, 1967}, 12346890})) return 1;
    if(dir_add_entry(dir, (person_t){ "Arthur", "Rimbaud", (date_t){1, 1, 2002}, 73645})) return 1;
    if(dir_add_entry(dir, (person_t){ "Paul", "Albuquerque", (date_t){23, 6, 1902}, 28736474})) return 1;
    if(dir_add_entry(dir, (person_t){ "Christophe", "Charpilloz", (date_t){11, 4, 2008}, 76253})) return 1;
    if(dir_add_entry(dir, (person_t){ "Emily", "Dickinson", (date_t){19, 8, 1950}, 636363})) return 1;
    if(dir_add_entry(dir, (person_t){ "Ada", "Lovelace", (date_t){12, 12, 2022}, 12345})) return 1;
    if(dir_add_entry(dir, (person_t){ "Qwertz", "Uiop", (date_t){1, 1, 562}, 67879})) return 1;
    return 0;
}

```

Après l'appel à cette fonction, l'appel à `dir_print` devra produire :

```

[0] Jean Paul, 18-12-1967, 12346890
[1] Arthur Rimbaud, 1-1-2002, 73645
[2] Paul Albuquerque, 23-6-1902, 28736474
[3] Christophe Charpilloz, 11-4-2008, 76253
[4] Emily Dickinson, 19-8-1950, 636363
[5] Ada Lovelace, 12-12-2022, 12345
[6] Qwertz Uiop, 1-1-562, 67879

```

Modalités

Ce TP est évalué et devra être rendu au plus tard le 22.11.2022. Un dépôt GIT à votre nom sous la forme https://githopia.hesge.ch/prog_soir_22-23_tp_1/VOTRE.NOM sera créé. La visibilité de ce dépôt doit rester privée et accessible uniquement à vous et aux enseignants. Si vous n'avez pas accès à

ce dépôt, faites le savoir à l'enseignant. C'est le code présent dans ce dépôt au moment du rendu qui sera pris en compte pour l'évaluation.

L'intégralité de votre code doit se trouver dans un dossier nommé `src` à la racine de votre dépôt et ses sous dossiers. Le programme principal doit se trouver dans un fichier nommé `directory.c` et le programme généré doit s'appeler `directory`. Vous êtes libres d'ajouter d'autres fichiers.

Le respect du cahier des charges est important, veillez à ce que les fonctionnalités demandées fonctionnent comme attendu et respectez les signatures imposées. Vous êtes libres d'ajouter d'autres fonctions.

Si vous voulez ajouter des informations complémentaires à destination de l'utilisateur (exemple d'utilisation, bug connu et comment l'éviter, ...) vous pouvez le faire dans le fichier `README.md` à la racine du projet.

La structure, la modularisation et la lisibilité de votre code sont prises en compte pour l'évaluation. Préférez des noms de fonctions et de variables explicites, des fonctions les plus courtes possibles et dont le rôle est aisé à comprendre. Evitez au maximum la logique dans le programme principal. Restez constant dans le style de nommage (pas de mélange anglais / français ou de snake / camel case).

Vous devez faire figurer un commentaire de documentation pour chaque fonction (un commentaire expliquant ce que fait la fonction, quelles sont ses entrées et sorties). Ajoutez des commentaires explicatifs lorsque cela est nécessaire.