TP Programmation C
Perfectionnement

Halim Djerroud (hdd@ai.univ-paris8.fr)
Note : La plus parts de ces exercices sont proposés par
Kais Klai (`https://lipn.univ-paris13.fr/~klai/`)

1 Les arguments

1.1 Exercices 1

Écrire un programme `mydate` en C qui prend en paramètre 3 valeurs (jour mois année) comme suite : `./mydate dd mm aaaa`.

1. Le programme doit afficher la date sous la forme : `mm/jj/aaaa` si utilisateur n'est pas américain (US) sinon afficher la date comme suite : `jj/mm/aaaa`

Astuce : Utilisez les variables d'environnement pour connaître la langue de l'utilisateur courant.

2 Les tableaux

1. Écrire un programme C qui remplit un tableau avec des valeurs entières saisies au clavier, et qui l'affiche après la fin de la saisie.
2. Écrire un programme C qui calcule le minimum d'un tableau d'entiers.
3. Écrire un programme C qui calcule la moyenne d'un tableau.
4. Écrire un programme C qui affiche la position de la première occurrence d'une valeur, si elle existe, dans un tableau.
5. Écrire un programme C qui affiche le nombre d'occurrence d'une valeur dans un tableau ainsi que les positions où elle apparaît dans un tableau.
6. Écrire un programme C qui teste si un tableau est ordonné.
7. Écrire un programme C qui insère une valeur dans un tableau trié.
8. Écrire un programme C qui décale les valeurs dans un tableau d'une position vers la gauche.
   Exemple :
   — Avant : D E C A L A G E
   — Après : E C A L A G E D
9. Écrire un programme qui supprime la première occurrence d'une valeur, si elle existe, dans un tableau.

10. Écrire un programme qui calcule, dans un tableau T3, la fusion de deux tableaux ordonnés T1 et T2 tel que T3 est également ordonné. Pensez à l'efficacité de votre algorithme.

3 Chaînes de caractères

Écrire les programmes suivants en vous servant des fonctions prédéfinies de la bibliothèque `string.h`.

1. Écrire un programme C pour trouver le nombre de caractères dans une chaîne de caractères saisie au clavier.
2. Écrire un programme C pour copier une chaîne de caractères dans une autre.
3. Écrire un programme C pour concaténer deux chaînes de caractères.
4. Écrire un programme C pour comparer deux chaînes de caractères selon l'ordre alphabétique.
5. Écrire un programme C pour convertir les lettres minuscules dans une chaîne de caractères en lettres majuscules.
6. Écrire un programme C pour inverser une chaîne de caractères.
7. Écrire un programme C qui vérifie si une chaîne de caractères s1 est une sous-chaîne d'une chaîne de caractères s2.
8. Écrire un programme C pour vérifier si une chaîne de caractères est un palindrome.
9. Écrire un programme C pour trouver le caractère le plus fréquent dans une chaîne de caractères.
10. Écrire un programme C qui calcule le nombre d'alphabets, le nombre de chiffres et le nombre de caractères spéciaux dans une chaîne de caractères

4 Les fonctions

L'objectif de cette partie est de réaliser un jeu simple en procédant petit à petit par l'écriture de fonctions. Vous allez donc définir plusieurs fonctions.

Il vous restera à réaliser le programme principal qui appellera ces fonctions dans le bon ordre pour réaliser le jeu.

Une combinaison du *Master Mind* est une suite de 5 pions de couleur, sachant qu'il existe 8 couleurs différentes : `rouge, bleu, vert, jaune, marron, blanc, noir, rose` et qu'une combinaison peut comporter plusieurs pions de la même couleur. Le but est de retrouver, en un nombre minimal d'essais, une combinaison choisie au hasard par l'ordinateur.

A chaque essai, vous proposez une combinaison de 5 couleurs, et l'ordinateur vous indique le nombre de pions de la bonne couleur et le nombre de pions bien placés par rapport à la combinaison à découvrir.

Pour simplifier le jeu, on utilisera des chiffres pour remplacer les couleurs : une combinaison sera donc un ensemble de 5 chiffres, chaque chiffre étant compris entre 1 et 8.

On choisit de stocker une combinaison dans un tableau de 5 entiers. Créer deux fichier `master.h` et `master.c` comme précisé ci-dessus. Ajouter dans le fichier `master.h` la déclaration d'une constante `SIZE(define SIZE 5)`qui sera utilisée comme taille physique de vos tableaux (combinaisons).

1. Écrire une fonction `display_comb(int comb[])` qui affiche les chiffres composant une combinaison séparés par-. Par exemple, la combinaison54642sera affichée comme suit : 5 - 4 - 6 - 4 - 2
2. Écrire une fonction `int read_comb(int comb[])` qui saisit une combinaison au clavier sous forme d'un nombre entier. La fonction extrait les chiffres composant ce nombre et les stocke dans le tableau d'entiers comb. Si le nombre donné par l'utilisateur ne contient pas5chiffres, ou que l'un de ses chiffres n'est pas compris dans l'interval[1,8], la fonction retourne0. Si la combinaison est valide, la fonction retourne1.
3. Écrire une fonction `void random_comb(int array[])` qui génère une combinaison aléatoire de 5 chiffre.
   Note : Pour générer un nombre aléatoire, utiliser `srand` et `rand` disponibles dans les bibliothèques `stdlib.h` et `math.h`
4. Écrire une fonction `int well_placed(int comb[], int propos[])` qui reçoit une combinaison à deviner (comb) et une combinaison proposée par l'utilisateur (propos), et retourne le nombre de chiffre qui sont bien placés.
   Exemple :
   comb = 5 4 8 7 2
   propos = 8 4 1 2 3
   valeur retournée : 1 (seul le chiffre 4 est bien placé)
5. crire une fonction `int at_wrong_place(int comb[], int propos[])` qui reçoit la combinaison à deviner (comb) et une combinaison proposée parl'utilisateur (propos), et qui retourne le nombre de chiffres qui sont présents,mais à la mauvaise place. On ne compte pas les chiffres bien placés.
   Exemple :
   comb = 5 4 8 7 2
   propos = 8 4 1 2 3
   valeur retournée : 2 (2 et 8 sont présents mais à la mauvaise place)
6. Ecrire une fonction `void result(int x, int y)` qui reçoit le nombre de chiffres présents, et le nombre de chiffres bien placés, et qui les affiche comme suite :
   `x presents - y bien places`
7. Ecrivez maintenant le programme principal qui appelle dans le bon ordre les différentes fonctions écrites. Le programme principal doit afficher le nombre de tentatives faites pour découvrir la combinaison cachée. Le nombre maximum de tentatives est de 20, si l'utilisateur n'a pas trouvé à la20 ème tentative, il a perdu. Dans le programme principal, on créera, pour les combinaisons proposées par l'utilisateur, un tableau à deux dimensions de 20 lignes et 5 colonnes pour stocker toutes les propositions successives du joueur.

5 Pointeurs et allocation dynamique

1. Écrire une fonction `int *create_array(int size)` qui crée un tableau dynamique de size entiers
2. Écrire une fonction `void create_array_bis(int **A, int size)` qui crée un tableau dynamique de size entiers. Vous remarquerez que le prototype est différent de la fonction précédente, quelle conséquence ça a sur la définition de la fonction et sur son appel à partir du `main()` ?
3. Écrire une fonction `void fill_array(int *A, int nb)` qui remplit le tableau A avec `nb` valeurs données par l'utilisateur.
4. Écrire une fonction `void disp_array(int *A, int nb)` qui affiche le tableau A (qui contient `nb` valeurs).
5. Écrier la fonction principale `main()` pour tester les fonctions développées ci-dessus.
6. Reprendre l'exercice 2 (Les tableaux) en utilisant des tableaux dynamiques.
7. Écrire une fonction `int ** create_2darray(int L, int C)` qui crée un tableau dynamique à deux dimensions (ayant L lignes et C colonnes).
8. Écrire une fonction `void create_2darray_bis(int ***A, int L, int C)` qui crée un tableau dynamique à deux dimensions (ayant `L` lignes et `C` colonnes). Vous remarquerez que le prototype est différent de la fonction précédente, quelle conséquence ça a sur la définition de la fonction et sur son appel à partir du `main()` ?
9. Écrire une fonction `void fill_2darray(int **A, int L, int C)` qui remplit le tableau à deux dimensions `A` avec `L*C` valeurs données par l'utilisateur.
10. Écrire une fonction `void disp_2darray(int **A, int L, int C)` qui affiche le tableau à deux dimensions A (contenant L*C valeurs) ligne par ligne.
11. Écrire la fonction principale `main()`pour tester les fonctions développées ci-dessus.

6 Structures

Soit les types `Date_t` et `Etudiant_t` définis ci-dessous et permettant de représenter une date et un étudiant respectivement.

```
/* Definition d'une date dans Date.h */
typedef struct {
    int jour;
    int mois;
    int annee;
}Date_t;

/* Prototypes dans Date.h */
void lireDate(Date_t*);
void afficheDate(Date_t*);
```

```
int compareDates(Date_t*, Date_t*);

/* Definition d'un etudiant dans Etudiant.h*/
typedef struct {
    char nom[30];
    char prenom[30];
    Date_t dateNaissance;
    long numEtud;
    double moyenne;
}Etudiant_t;

/* Prototypes dans Etudiant.h */
void lireEtudiant(Etudiant_t*);
void afficheEtudiant(Etudiant_t*);
```

Écrire et tester les fonctions suivantes :

1. Dans un fichierDate.c
   (a) Une fonction `void lireDate(Date_t*)` qui permet de saisir une date au clavier sous la forme `jour/mois/annee`
   (b) Une fonction `void afficheDate(Date_t*)` qui permet d'afficher une date sous la forme `jour/mois/annee`.
   (c) Une fonction `int compareDates(Date_t*, Date_t*)` qui compare deux dates : la fonction admet deux paramètres d1 et d2, de `typeDate_t*`, et retourne 0 si la date pointée par d1 est égale à la date pointée par d2, 1 si la date pointée par d1 est ultérieure à la date pointée par d2, et 1 si la date pointée par d1 est antérieure à la date pointée par d2.
2. Dans un fichier `Etudiant.c`
   (a) Une fonction `void lireEtudiant(Etudiant_t*)` qui permet de saisir les membres d'un étudiant au clavier, et de les stocker dans la structure pointée par le paramètre de type `Etudiant_t*`.
   (b) ne fonction `void afficheEtudiant(Etudiant_t *)` qui permet d'afficher à l'écran les membres d'un étudiant.

6.1 Tableau de structures statiques

1. Définissez une classe d'étudiants `classEtud` ayant comme membres `listeEtud` qui est un tableau dynamique de `Etudiant_t`,nbEtudqui est un entier stockant le nombre d'étudiants de la classe, et moyClasse qui est un nombre réel stockant la moyenne générale de la classe.
2. Ecrire et tester (dans un `main()`) les fonctions ci-dessous. Utilisez, comme d'habitude, deux fichiers `classeEtud.h` (pour la définition de la structure `classEtud` et les prototypes des fonctions) et `classeEtud.c` (pour définir les fonctions déclarées dans `classeEtud.h`), en plus du fichier contenant le programme principal (`main()`). N'hésitez pas à vous servir des fonctions que vous avez développées dans la première partie.

(a) Une fonction `void lireTabEtudiants(Etudiant_t * T, int N)` qui remplit un tableau d'*Etudiant_ t*.

(b) Une fonction `void afficheTabEtudiants(Etudiant_t * T, int N)` qui affiche un tableau d'`Etudiant_t`.

(c) Une fonction `double moyTabEtudiants(Etudiant_t * T, int N)` qui retourne la moyenne (générale) des moyennes des étudiants.

(d) Une fonction `int meilleurMoy(Etudiant_t * T, int N)` qui retourne l'indicede l'étudiant ayant la meilleur moyenne.

(e) Une fonction `int plusJeuneEtudiant(Etudiant_t * T, int N)` qui retourne l'indice du plus jeune étudiant. Vous utiliserez la fonction compareDates() pour comparer deux dates.

(f) Ecrire une fonction `void lireClasse(classEtud*)` qui reçoit une structurede type classEtudet qui remplit ses membres de la structure `classEtud` à partir du clavier (sauf la moyenne générale de la classe, moyClasse, qui doit être calculée en utilisant la fonction `moyTabEtudiants()`).

(g) Écrire une fonction `afficheClasse()` qui affiche à l'écran les membres d'une structure de type `classEtud`.

(h) N'oubliez pas de libérer le membre dynamique `listeEtud` de la classe à la fin du programme principal. Pour ce faire, utilisez une fonction `void libClasseEtud(classEtud*)`

3. Tableau de structures
On se propose dans cette partie de manipuler un tableau de pointeurs sur des structures.

(a) Écrire une fonction `Etudiant_t *creerEtudiant()` qui alloue dynamiquement une structure `Etudiant_t` et la retourne.

(b) Changer le membre `listeEtud` de la structure `classEtud` pour que ça soit un tableau de pointeurs sur des structure `Etudiant_t`.

(c) Quelle est la conséquence de ce changement sur les fonctions développées dans la section précédente (Tableau de structures statiques) ? Redéveloppez toutes ces fonctions ainsi que le programme principal `main()`.

4. Fichier texte

Dans le cas d'un fichier texte, on peut supposer que chaque ligne du fichier texte (par exemple `Etudiants.txt`) contient les informations concernant un étudiant (les membres de la structure `Etudiant_t`) séparées par des espaces :

```
Blanc Michel 12/1/1999 123421 12.5
Martin Tom 3/12/2000 123543 13.5
```

En utilisant le type FILE*, et les fonctions `fprintf()` et `fscanf()`, écrivez et testez :

(a) Une fonction qui remplit un fichier d'étudiants à partir de données saisies au clavier.

(b) Une fonction qui lit le contenu d'un fichier d'étudiants et l'affiche à l'écran.

(c) Une fonction qui lit le contenu d'un fichier d'étudiants et le stocke dans une structure `classEtud`.

5. Fichier binaire

   Nous pouvons stocker une liste d'étudiants en manipulant directement la structure Etudiant_t qu'on écrit (ou lit) dans un fichier binaire (par exemple appelé Etudiants). Contrairement à un fichier texte, un fichier binaire n'est pas affichable et ne peut être manipulé qu'à travers un programme.
   En utilisant le type `FILE*`, et les fonctions `fwrite()` et `fread()`, écrivez et testez les trois fonctions ci-dessus (Section 3.1).