Travaux Dirigés n°1

Programmation C

—L2 MIDO—

Révisions

Révisions, entrées-sorties, types, structures conditionnelles, boucles.

▸ Exercice 1. Comprendre un programme

Commenter ligne par ligne les trois programmes suivants :

```c
int main() {
  printf("Je suis 1 programme C.\n");
  return 0;
}
```

```c
int main() {
  int nombre;
  printf("Voici un nombre : %d\n", 7);
  nombre = 20;
  printf("En voici un autre : %d\n",
    nombre + 22);
  return 0;
}
```

```c
int main() {
  int i = 3, j = 5;
  float x, y, z;
  x = 3.0;
  printf("i=%d, j=%d, somme=%d\n", i,
    j, i+j);
  i = j/2;
  y = x/2;
  z = j/2;
  j = j/2.0;
  printf("i=%d\ny=%f z=%f j=%d\n", i,
    y, z, j);
  return 0;
}
```

▸ Exercice 2. Affichages

Indiquer si les instructions suivantes provoquent des erreurs de compilation. Donner l'affichage de ce programme.

```c
#include <stdio.h>
int main() {
    int i, k;
    float x, y;
    char c, d;
    i = 48;
    printf("%c, %d\n", i, i);
    i = 304%256;
    printf("%c, %d\n", i, i);
    c = 'a';
    printf("%c, %d\n", c, c);
    c++;
    printf("%c, %d\n", c, c);
    d = 'A';
    printf("%c, %d\n", d, d);
    k = 'd';
    printf("%c, %d\n", k, k);
    k = d;
    printf("%c, %d\n", k, k);
    y = 'a';
    x = 65;
    printf("%f, %f\n", y, x);
    return 0;
}
```

▸ **Exercice 3. Lecture au clavier**

Écrire un programme C qui demande à l'utilisateur de saisir un entier au clavier et qui l'affiche ensuite entre deux lignes vides. On se servira de l'instruction `scanf("%d",&i);` qui permet de saisir un entier et de le mettre dans la variable **i**.

▸ **Exercice 4. En moyenne**

Écrire un programme C qui demande à l'utilisateur de saisir une série d'entiers. Lorsque l'utilisateur entre le nombre 0 (qui ne fait pas partie de la série), le programme se termine après avoir affiché le nombre d'entiers saisis, leur moyenne, le plus petit entier entré et le plus grand.

▸ **Exercice 5. Alphabet**

Afficher tous les caractères compris entre `'a'` et `'z'` avec pour chacun leur code ASCII exprimé en décimal. Même chose pour les caractères compris entre `'A'` et `'Z'`. Pour répondre à cet exercice, il n'y a pas besoin de connaître le code ASCII de `'a'`. Utiliser une fois une boucle `while` et une fois une boucle `for`.

▸ **Exercice 6. Décomposition**

Écrire un programme qui lit un entier au clavier et affiche chaque chiffre du nombre sur une ligne en commençant par le chiffre des unités. Ainsi le nombre 2867 est affiché :

```
7
6
8
2
```

Travaux Dirigés n°2

Programmation C

—L2 MIDO—

Fonctions et tableaux

Dans ce TD, les notions de fonctions en C ainsi que les tableaux sont abordés.

▸ Exercice 1. Tableaux

1. Écrire une fonction qui prend en arguments un tableau d'entiers ainsi que sa taille, et qui l'affiche. Pourquoi avons-nous besoin de la taille en argument ?
2. Écrire une fonction qui prend en arguments un tableau d'entiers ainsi que sa taille `n`, et qui demande la saisie de `n` entiers à placer dans le tableau.
3. Écrire une fonction qui prend en arguments un tableau d'entiers et sa taille, et qui renvoie la somme des entiers présents dans le tableau.
4. Écrire une fonction `main()` qui définit un tableau de taille 4 et appelle les 3 fonctions précédentes.
5. Que faut-il faire si on veut maintenant un tableau de taille 10 ? Modifier la taille du tableau à l'aide d'une constante N. Modifier la fonction `main()` pour qu'il demande à l'utilisateur le nombre de valeurs désirées (et redemande tant que ce nombre est plus grand que N).

▸ Exercice 2. Doublons

Le but de cet exercice est d'écrire une fonction qui permet de tester si un tableau d'entiers contient des doublons. Un doublon dans un tableau *tab* est une paire d'indices $\{i, j\}$ telle que $i \neq j$ et $tab[i] = tab[j]$. Par exemple, la paire $\{2, 4\}$ est un doublon dans le tableau 7 1 6 3 6 2 9 8.

1. Écrire une fonction `int NbOccurrencesTab(int tab[], int n, int a)` renvoyant le nombre d'occurrences de `a` dans le tableau `tab` de taille `n`.
2. En déduire une fonction `int ContientDoublons(int tab[], int n)` qui renvoie 1 si `tab` contient un doublon et 0 sinon.

▸ Exercice 3. Second plus grand

Écrire une fonction qui prend en arguments un tableau `tab` et sa taille. Cette fonction retourne l'indice du second plus grand élément du tableau (on suppose que le tableau est de taille au moins 2).

▸ **Exercice 4. Somme à k**

1. Écrire une fonction qui prend en arguments un tableau `tab`, sa taille et un entier k. Cette fonction recherche (et affiche) les paires d'éléments du tableau dont la somme vaut k. La fonction retourne le nombre de paires trouvées.
2. Écrire une fonction `main()` qui appelle la fonction de saisie d'un tableau (déjà écrite) puis cette fonction et qui affiche le nombre de paires trouvées.
3. *[Bonus]* Le nombre d'opérations dans le pire cas de la fonction précédente est de l'ordre de la taille du tableau au carré. On peut faire un peu mieux avec l'astuce suivante : on utilise un tableau auxiliaire `tab2`. On va dans un premier temps mettre toutes les cases de `tab2` à -1. Puis, on va stocker `i` dans `tab2[tab[i]]`, pour chaque case de `tab`. Quelle est la taille nécessaire pour `tab2` ? (pour l'exercice, on suppose que `tab2` est de taille suffisante, et que les entiers dans `tab` sont positifs ou nuls). Comment se servir ensuite de `tab2` pour déterminer s'il existe un complément à k pour un `tab[i]` ? Coder cette fonction. (pour simplifier, on suppose que tous les entiers sont distincts dans le tableau original).

Travaux Dirigés n°3

Programmation C

—L2 MIDO—

Chaînes de caractères et tableaux à deux dimensions

On étudie un cas spécial de tableaux, les chaînes de caractères. On évoque également les tableaux à deux dimensions. On considère qu'il existe une constante `T_MAX` définie par `#define T_MAX 256` qui est utilisée pour la taille des tableaux.

▸ Exercice 1. La taille compte

Écrire la fonction `int LgChaine(char ch1[])` qui retourne la longueur (i.e. le nombre de caractères) de la chaîne `ch1`.

▸ Exercice 2. Comparaison

Écrire une fonction `int CompareChaines(char ch1[], char ch2[])` qui compare des chaînes de caractères contenues dans les tableaux de caractères `ch1` et `ch2`. Le résultat est 0 si les chaînes sont égales, un nombre négatif si `ch1` < `ch2` (dans le dictionnaire), positif dans le cas contraire. On suppose que tous les caractères sont en minuscule et on rappelle que le code ASCII du caractère `'\0'` est 0.

▸ Exercice 3. Majuscules

Écrire une fonction `void Majuscules(char ch[])` qui modifie `ch` pour mettre tous ses caractères en majuscules. Il n'y a pas besoin de connaître les codes ASCII pour écrire cette fonction, il suffit de savoir que les lettres de l'alphabet se suivent dans le code ASCII.

▸ Exercice 4. Palindromes

Le but de cet exercice est d'écrire une fonction qui vérifie si une chaîne de caractères est un palindrome (dont l'ordre des lettres reste le même qu'on le lise de gauche à droite ou de droite à gauche, sans tenir compte des espaces), comme par exemple la phrase : *esope reste ici et se repose*.

1. Écrire une fonction `void Copie(char ch1[],char ch2[])` qui copie le contenu de la chaîne `ch1` dans la chaîne `ch2` ; on considère qu'une place mémoire suffisante a été réservée pour `ch2`.
2. Écrire une fonction `void Miroir(char ch[])` qui inverse les caractères de la chaîne `ch`, sans utiliser de tableau temporaire. Par exemple, la fonction transformera *trace* en *ecart*.
3. Écrire une fonction `void SupprOccur(char ch[], char c)` qui supprime les occurrences de `c` dans `ch`, sans utiliser de tableau annexe et sans boucle imbriquée.
4. En déduire une fonction `int EstPalindrome(char ch[])` qui renvoie 1 si `ch` est un palindrome, 0 sinon.

▸ **Exercice 5. Table de multiplication**

1. Écrire une fonction `RemplirTableauMult()` qui prend en argument un tableau d'entiers à 2 dimensions `tab` et remplit ligne par ligne la table de multiplication de 1 à 10.
2. Écrire une fonction qui l'affiche.

Attention, en C, on ne peut pas passer un tableau à 2 dimensions en argument d'une fonction sans connaître au moins le nombre de colonnes (la taille des lignes)...

▸ **Exercice 6. Tableaux de caractères à 2 dimensions**

1. Écrire une fonction `CreerDamier()` qui prend en argument un tableau de caractères à deux dimensions `tab` et un entier n où $n * n$ est la taille effective du tableau. Cette fonction modifie `tab` de sorte à modéliser un damier : les cases blanches sont codées par le caractère `-` et les cases noires par le caractère `*`. La case d'indice (0, 0) est blanche. Pour 5×5 on obtiendrait :

```
-*-*-
*-*-*
-*-*-
*-*-*
-*-*-
```

2. Écrire une fonction `CreerCroix()` qui prend en argument un tableau de caractères à deux dimensions, un entier n où $n*n$ est la taille effective du tableau et deux entiers i et j. Cette fonction modifie `tab` de sorte que l'élément d'indice (i, j) de `tab` soit remplacé par le caractère `*`. De plus, les deux diagonales qui passent par cet élément doivent contenir des `+` et il doit y avoir des `.` partout ailleurs dans le tableau. Exemple pour 7×7 avec $(1, 2)$, on obtient :

```
.+.+...
..*....
.+.+...
+...+..
.....+.
......+
.......
```

Travaux Dirigés n°4

Programmation C

—L2 MIDO—

Un pointeur est une variable contenant l'adresse mémoire d'une autre variable.

▸ Exercice 1. Échange

Qu'affiche le programme suivant et pourquoi ?

```c
void ech1(int a, int b) {
    int tmp = a;
    a = b;
    b = tmp;
}

void ech2(int *a, int *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

void ech3(int a, int *b) {
    int tmp = a;
    a = *b;
    *b = tmp;
}
```

```c
int main() {
    int a = 1, b = 2;
    ech1(a,b);
    printf("a=%d b=%d\n", a, b);
    ech2(&a,&b);
    printf("a=%d b=%d\n", a, b);
    ech3(a,&b);
    printf("a=%d b=%d\n", a, b);
    return 0;
}
```

▸ Exercice 2. Quelle heure est-il ?

1. Écrire une fonction `timeCutting()` prenant un nombre de minutes en argument et permettant à la fonction appelant `timeCutting()` d'obtenir le nombre d'heures et de minutes correspondant. Par exemple, si on l'appelle avec 30, on doit avoir 0 heure et 30 minutes, et si on l'appelle avec 90, on doit avoir 1 heure et 30 minutes.

   Pour cela, réfléchir à un moyen pour qu'une fonction puisse transmettre plus d'une valeur à la partie du programme qui l'appelle.
2. Écrire une fonction `main()` qui demande à l'utilisateur de saisir un nombre de minutes puis affiche le nombre d'heures et de minutes correspondant déterminé à l'aide de `timeCutting()`.

▸ Exercice 3. MinMax

Écrire une fonction qui passe à la partie du programme qui l'appelle la valeur minimum et la valeur maximum d'une suite de n entiers saisis par l'utilisateur (on demande alors dans un premier temps de saisir ce n).

▸ Exercice 4. Déréférencement

Que fait et affiche le code suivant ?

```
1     int var = 10;
2     int var2 = var;
3     int *p = &var;
4     *p=20;
5     printf("%d %d %d %d\n", var, var2, p, *p);
6     int *p2;
7     *p2=15;
8     printf("%d\n", *p2);
```

▸ Exercice 5. Pointeurs et tableau

Soient le tableau `t` suivant et le pointeur `p` pointant sur `t` :

```
int t[] = {3, 8, 12, 78, 43, 90, 177, 64};
int* p = t;
```

A quoi correspondent les valeurs suivantes : `*p+2`, `*(p+2)`, `t`, `p`, `&p`, `t+3`

▸ Exercice 6. Pointeurs et tableau II

Expliquer ce qu'il se passe ligne par ligne.

```
1     int tab[3];
2     int* p;
3     p = tab;
4     p = tab + 2;
5     tab[1] = 5;
6     p[-1] = 6;
7     tab = p;
```

▸ Exercice 7.

Donner les valeurs de `a`, `b`, `c`, `p1` et `p2` après exécution de chacune des instructions suivantes :

```
1     int a=1, b=2, c=3;
2     int* p1 = &a;
3     int* p2 = &c;
4     *p1 = (*p2)++;
5     p1 = p2;
6     p2 = &b;
7     *p1 -= *p2;
8     ++*p2;
9     *p1 *= *p2;
10    a = ++*p2**p1;
```

Travaux Dirigés n°5

Programmation C

—L2 MIDO—

Structures

Ce TD aborde la notion de structures en C.

▸ Exercice 1. Rationnels

On souhaite représenter un nombre rationnel par son numérateur, son dénominateur, et son signe (numérateur et dénominateur sont donnés positifs).

1. Définir une structure `Rationnel` permettant de représenter un nombre rationnel.
2. Écrire une fonction `RemplirRationnel()` qui demande à l'utilisateur la saisie d'un nombre rationnel. La fonction prend en paramètre l'adresse d'un rationnel et le modifie.
3. Écrire une fonction `AfficheRationnel()` qui affiche le rationnel donné en argument. On rappelle qu'un passage par valeur copie entièrement la structure, contrairement à un passage par adresse.
4. Écrire une fonction effectuant la multiplication entre deux rationnels. Choisir une signature de fonction adaptée. On ignorera la simplification.
5. Écrire une fonction `main()` testant ces fonctions.

▸ Exercice 2. Classe d'étudiants

Un étudiant est modélisé par un nom, un numéro d'étudiant et une série de notes.

1. Écrire une structure symbolisant un étudiant.
2. Écrire une fonction permettant la saisie d'un étudiant à l'utilisateur.
3. Écrire une fonction affichant un étudiant.
4. Écrire une fonction renvoyant la moyenne d'un étudiant.
5. Écrire une fonction permettant l'ajout d'une note à un étudiant. Cette fonction renvoie 1 si l'ajout a été possible, 0 sinon.
6. Écrire une fonction `main()` testant ces fonctions.
7. Écrire une structure symbolisant une classe d'étudiants.
8. Écrire une fonction permettant l'affichage d'une classe.
9. Écrire une fonction permettant l'ajout d'un étudiant dans une classe.
10. Écrire une fonction prenant en argument un nom d'étudiant et une classe et renvoyant la moyenne de cet étudiant, ou -1 s'il n'existe pas dans la classe.
11. Écrire une fonction renvoyant l'étudiant ayant la meilleure moyenne de la classe.
12. Écrire une fonction `main()` testant ces fonctions.

Travaux Dirigés n°6

Programmation C

—L2 MIDO—

Allocation dynamique

Pendant les TD précédents, nous avons manipulé des tableaux possédant une taille maximum fixée à l'initialisation. Nous allons maintenant voir comment gérer des ensembles de structures dont la taille maximum n'est pas fixée à l'avance. On rappelle l'existence des 4 fonctions suivantes de la bibliothèque `<stdlib.h>` :

```
void *malloc(size_t size);
void free(void * ptr);
void *calloc(size_t nmemb, size_t size);
void *realloc(void * ptr, size_t size);
```

▸ Exercice 1. Ensemble d'entiers

1. Définir une structure permettant de stocker un ensemble d'entiers. On utilisera 3 champs : un tableau pour stocker les entiers, un entier indiquant la taille du tableau et un entier indiquant le cardinal de l'ensemble, i.e. le nombre réel d'entiers dans le tableau au moment courant.
2. Écrire une fonction retournant un pointeur sur un ensemble vide, i.e. sans élément. Elle allouera l'espace pour l'ensemble lui même, et pour le tableau stockant les entiers, avec une taille initiale de 5.
3. Écrire une fonction prenant un pointeur sur un ensemble E et un entier x en arguments, et renvoyant 1 si $x \in E$, 0 sinon.
4. Écrire une fonction prenant un pointeur sur un ensemble et un entier en arguments, et ajoutant l'entier à l'ensemble s'il n'était pas déjà présent. Note : si le tableau utilisé n'est pas de taille suffisante, on double sa taille avec la fonction `realloc()`.
5. Écrire une fonction prenant un pointeur sur un ensemble et un entier en arguments et supprimant cet entier de l'ensemble.
6. Écrire une fonction prenant un pointeur sur un ensemble et désallouant son espace mémoire.
7. Écrire une fonction prenant deux pointeurs vers deux ensembles X et Y, et renvoyant un pointeur vers un troisième ensemble $Z = X \cup Y$.
8. Écrire une fonction prenant deux pointeurs vers deux ensembles X et Y, et renvoyant un pointeur vers un troisième ensemble $Z = X \cap Y$.
9. Réfléchir aux changements qui doivent être faits si on demande que les ensembles soient triés (et que les opérations sur ces ensembles gardent cette propriété), et son influence sur le nombre d'opérations effectuées dans le pire des cas dans les fonctions précédentes.

▸ **Exercice 2. Tableau**

Soit la fonction `main()` suivante dans laquelle un tableau `v` de taille `n` est créé puis affiché :

```c
int main() {
    int *v = NULL; /* An array */
    int n; /* the size of array v */

    generate(&v, &n);
    display(v, n);
    free_mem(&v);

    return 0;
}
```

1. Écrire la fonction `generate()` qui demande à l'utilisateur de saisir la taille d'un tableau d'entiers, et qui le remplit avec le carré de l'indice du tableau.
2. Écrire la fonction `display()` qui affiche les éléments du tableau du dernier au premier.
3. Écrire la fonction `free_mem()` qui libère la mémoire et remet le pointeur du tableau à `NULL`.

Travaux Dirigés n°7

Programmation C

—L2 MIDO—

Listes chaînées

Ce TD aborde les listes chaînées en C.

▸ Exercice 1. Liste chaînée

1. Définir une structure `Maillon` contenant un entier (la valeur que l'on veut stocker) et un pointeur vers un maillon (le maillon suivant).
2. Définir une structure `Liste` contenant la taille de la liste ainsi qu'un pointeur vers un maillon, le premier élément de la liste.
3. Écrire une fonction créant une liste chaînée vide et retournant un pointeur sur celle-ci.
4. Écrire une fonction prenant un pointeur sur une liste chaînée et un entier, ajoutant un maillon contenant cet entier en valeur au début de la liste chaînée. Quel est le nombre d'opérations effectuées dans le pire des cas ?
5. Écrire une fonction prenant un pointeur sur une liste chaînée en argument et l'affichant. Écrire une version itérative.
6. Écrire une fonction prenant un pointeur sur une liste chaînée et un entier, ajoutant un maillon contenant cet entier en valeur à la fin de la liste chaînée. Quel est le nombre d'opérations effectuées dans le pire des cas ? Réfléchir à une version plus rapide en temps.
7. Écrire une fonction prenant un pointeur sur une liste chaînée et un entier, retournant 1 si l'entier est présent dans la liste, 0 sinon.
8. Écrire une fonction d'affichage récursive d'une liste chaînée. Pour cela, vous écrirez une fonction prenant un pointeur vers un maillon en plus de celle prenant une liste.
9. Écrire une fonction retournant un pointeur sur le premier maillon de la liste chaînée prise en argument. La fonction enlève ce maillon de la liste.
10. Écrire une fonction prenant un pointeur sur une liste chaînée en argument détruisant la liste chaînée et libérant son espace mémoire.
11. Écrire une fonction prenant un pointeur sur une liste chaînée et un entier en arguments et supprimant toutes les occurrences de cet entier dans la liste. Attention à la gestion de l'espace mémoire.
12. (A la maison) Écrire une fonction prenant un pointeur sur une liste chaînée et qui en inverse le chaînage, sans créer de nouveaux maillons.

▸ **Exercice 2. File pile**

On rappelle qu'une file se comporte comme une file d'attente *(First In First Out)* alors qu'une pile se comporte comme une pile d'assiettes *(Last In First Out)*.

Créer un type `File` et un type `Pile`, utilisant la structure de liste précédemment écrite.

En utilisant les fonctions définies à l'exercice précédent, créer des fonctions permettant les opérations suivantes :

— `PileInitialiser()` et `FileInitialiser()` renvoyant un pointeur sur une Pile ou un File.

— `PileVide()` et `FileVide()` indiquant si la Pile et la File sont vides.

— `PileHauteur()` et `FileLongueur()` donnant la taille de la Pile et de la File.

— `Enfiler()` qui prend un pointeur sur une file et un entier qui est ajouté à la file.

— `Empiler()` qui prend un pointeur sur une pile et un entier qui est ajouté à la pile.

— `Defiler()` qui prend un pointeur sur une file et retire l'entier devant sortir en premier.

— `Depiler()` qui prend un pointeur sur une pile et retire l'entier devant sortir en premier.

— `PileDetruire()` et `FileDetruire()`.

Un code proprement écrit devrait utiliser des fichiers séparés pour la définition (`.h`) et le code (`.c`) des listes, ainsi que pour les files et piles. Nous verrons ceci plus tard.

Travaux Dirigés n°8

Programmation C

—L2 MIDO—

Liste chaînée triée

Implémentation d'opérations sur des listes chaînées triées d'entiers.

▸ **Exercice 1.**

1. Écrire une fonction `Maillon *RechercheTrie(Liste *L, int valeur)` qui renvoie l'adresse de la première cellule de la liste `L` contenant l'entier `valeur` et qui renvoie `NULL` si une telle cellule n'existe pas.
2. Écrire une fonction `int EgaleTrie(Liste *un, Liste *deux)` qui renvoie 1 si les deux listes triées sont identiques (mêmes éléments avec le même nombre d'occurences) et 0 sinon.
3. Écrire une fonction qui supprime tous les doublons dans une liste triée. Par exemple, la liste `(1 1 1 3 4 4 5 5 5)` sera transformée en `(1 3 4 5)`.
4. Écrire une fonction effectuant l'insertion d'une valeur dans une liste triée (elle doit garder la liste triée).
5. Écrire une fonction permettant d'enlever un élément `x` d'une liste triée (elle retourne le maillon correspondant).
6. On cherche à réaliser la fusion de deux listes triées `Un` et `Deux`.
   (a) Écrire une fonction réalisant cette opération avec création de nouvelles cellules, sans utiliser la fonction d'ajout en fin de liste. La fonction retournera la liste résultat, `NULL` en cas d'échec.
   (b) (À la maison) Écrire une fonction réalisant cette opération sans création de nouvelles cellules, la liste `Un` étant formée des cellules précédemment dans les deux listes.