TRAVAUX PRATIQUES II

Tableaux en C

Le but de ce TP est d'apprendre à manipuler des tableaux en C. Voici un raccourci vers le Nuage (je vous invite à marquer la page Nuage en marque-page firefox) :

`https://link.infini.fr/tp-info-cg`

1. Télécharger les fichiers du TP [1].

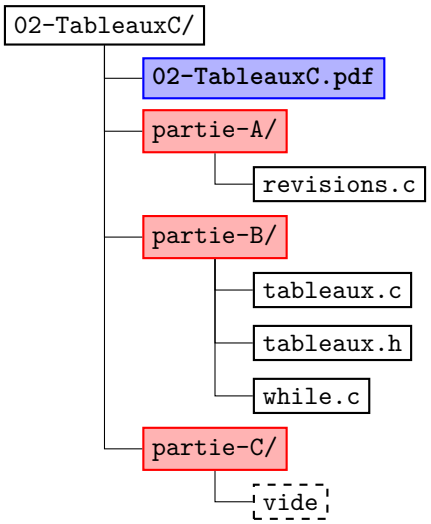

FIGURE II.1 – Organisation du dossier du TP

A Révisions du TP précédent

1. Ouvrez le fichier `revisions.c` . Remplissez la fonction `somme_impairs` [2] de sorte à ce qu'à `n` elle associe la somme des entiers positifs impairs inférieurs ou égaux à `n`.
On pourra s'inspirer de la fonction qui somme les entiers (peu importe leur parité) vue en cours/TD.
Pour vous aider à déboguer, voici les 20 premières valeurs de cette somme :

n	Somme impairs	n	Somme impairs	n	Somme impairs	n	Somme impairs
0	0	5	9	10	25	15	64
1	1	6	9	11	36	16	64
2	1	7	16	12	36	17	81
3	4	8	16	13	49	18	81
4	4	9	25	14	49	19	100

1. Évitez de télécharger l'entiereté du dossier de MP2I de Nuage, puisqu'il contient aussi les cours, les TD, et les autres TP. Il vaut mieux ne télécharger que le dossier `02-TableauxC` .
2. Rappel utile : `a % b` est le reste de madivision euclidienne de `a` par `b` . Vous pouvez l'utiliser pour tester si un entier est pair ou non.

B Tableaux

Pour cette partie, j'ai coupé le code C en deux fichiers : `partie-B/tableaux.c` et `partie-B/main.c`. Vous allez coder dans le premier fichier. Le second fichier contient la fonction `main` ainsi que des tests de vos fonctions : vous pouvez aller le regarder et l'éditer pour modifier les tests si vous le souhaitez.

Le fichier `partie-B/tableaux.h` est une description des fonctions de `partie-B/tableaux.c`, utilisable par le compilateur. Ne vous prenez pas la tête avec, nous verrons son fonctionnement dans quelques mois.

Si vous avez envie de créer des fonctions intermédiaires[3] dans `partie-B/tableaux.c`, n'hésitez pas.

Le prototype de toutes les fonctions de `partie-B/tableaux.c` est correct, vous n'aurez pas besoin de les changer. Vous devrez par contre modifier le corps des fonctions : actuellement, ce sont juste des `return -1;` (ou `return false;`)...

B.1 Rappels brefs du cours

Un tableau `T` de longueur `len` est un « meuble » qui contient `len` variables, nommées `T[0]`, `T[1]`, ..., `T[len-1]` :

Tableau T :	T[0]	T[1]	T[2]	.....	T[len-2]	T[len-1]
Indices :	0	1	2	...	len-2	len-1

FIGURE II.2 – Schéma d'un tableau

On peut traiter les cases d'un tableau les unes après les autres ainsi :

```
1 i ⟵0
2 tant que i < len faire
3 |   traiter T[i]
4 |   i ⟵i+1
```

Pseudo-code

```c
int i = 0;
while (i < len) {
  <traiter T[i]>;
  i = i+1;
}
```

Traduction en C.

Remarque.

- La longueur d'un tableau (son nombre de cases) est définie à la création du tableau et n'est pas modifiable ensuite.
- En anglais, « longueur » se dit « length » et on utilise donc souvent `len` pour nommer la longueur d'un tableau.

B.2 Manipulation de tableaux en C

Définition 1 (Manipulation des tableaux en C).
Si `T` est un tableau et `i` l'indice d'une case de `T`, alors `T[i]` est la case d'indice `i` de `T`. On peut la manipuler comme n'importe quelle variable.

2. Complétez la fonction `premiere_case` (de `partie-B/tableaux.c`). Elle prend en argument un tableau `T`, et vous devez faire en sorte qu'elle renvoie le contenu de la première case de `T`.
3. Complétez la fonction `somme_tableau`. Elle prend en argument un tableau `U`, sa longueur `len` et vous devez faire en sorte qu'elle renvoie la somme des `len` cases de `U`.

Remarque. Comme nous le verrons plus tard, quand une fonction prend en argument un tableau et `len`, il est impossible de savoir si `len` est la « vraie longueur » du tableau (le nombre de cases qu'il avait à la création) ou si c'est un « nombre de cases » (donc inférieur ou égal à la vraie longueur). Cela provient du fait que, comme nous le verrons plus tard, il n'y a aucun moyen de détecter si une case est la dernière case du tableau ou non. On ne peut que faire confiance à la valeur de `len` donnée en argument.

Exemple. Dans 3 derniers tests de `somme_tableau` (dans `partie-B/main.c`), c'est le *même* tableau qui est donné à tester à chaque fois, mais avec une `len` indiquée différente.

3. Une fonction intermédiaire est une fonction qui n'est pas explicitement demandée, mais qui simplifie l'écriture de la fonction demandée.

4. **a.** Testez le pseudo-code ci-dessous sur des tableaux (par exemple, des tests affichés par le programme). Que fait-il ?

Algorithme 2 : Mystère

Entrées : `T` un tableau, `len` sa longueur

```
1 i ←0
2 maxi ←T[0]
3 tant que i < len faire
4 |   si maxi < T[i] alors
5 |   |   maxi ←T[i]
6 |   i ←i+1
7 renvoyer T[i]
```

b. Codez cet algorithme en C.

c. Parfois, on a envie d'écrire cet algorithme comme suit. Prouvez que cela ne fonctionne pas :

Algorithme 3 : Mystère qui marche pas

Entrées : `T` un tableau, `len` sa longueur

```
1 i ←0
2 maxi ←0
3 tant que i < len faire
4 |   si maxi < T[i] alors
5 |   |   maxi ←T[i]
6 |   i ←i+1
7 renvoyer T[i]
```

5. Complétez la fonction `est_present`. Elle prend en argument un entier `x`, un tableau `T` et `len` un nombre de cases, et vous devez faire en sorte qu'elle renvoie `true` si et seulement si `x` se trouve dans une des `len` premières cases de `T` (elle renvoie `false` sinon).
Une fois que vous avez une solution, vous pouvez peut-être essayer de l'améliorer pour qu'elle renvoie `true` dès qu'un `x` est trouvé, sans attendre la fin du parcours du tableau.
NSI première : une recherche dichotomique peut-elle s'appliquer ici ? Pourquoi ?

B.3 Création de tableaux

Définition 2 (Création de tableaux).
Pour créer un tableau, on utilise :

```
<type_des_cases> <identifiant>[<longueur>];
```

Syntaxe générale.

```
int tab[500];
```

Exemple.

On dit que l'on **initialise** une case du tableau la toute première fois qu'on lui donne une valeur. On dit qu'on initialise le tableau lorsque l'on initialise toutes ses cases.
⚠ La longueur d'un tableau doit être une **constante littérale**, c'est à dire qu'elle ne doit pas faire apparaitre d'identifiants (de noms de variables).

Remarque.

- La longueur est fixe et n'est pas modifiable une fois le tableau créé.
- Si vous essayez d'indiquer autre chose qu'une constante littérale comme longueur du tableau, cela marchera peut-être : certains compilateurs implémentent cette fonctionnalité, mais c'est un bonus du compilateur et non une fonctionnalité du langage C. De plus, ce que l'on obtient [4] est moins efficace qu'un vrai tableau.
  C'est un piège classique de MP2I, sur lequel on va vous attendre au tournant.
- Si on n'initialise pas les cases du tableau, elles peuvent contenir tout et n'importe quoi. En pratique, elles contiennent le n'importe quoi qui traine dans la mémoire de l'ordinateur à l'endroit où le tableau est créé.
  Il ne faut jamais lire une case d'un tableau qui n'a pas été initialisée.

4. On nomme ces faux tableaux des V.L.A. : Variable Length Array (un tableau dont la longueur est une variable).

- Les fonctions que vous avez écrites jusqu'à présent n'initialisaient pas les tableaux : c'est normal, mes tests les avaient déjà initialisés pour vous.
- Nous verrons plus tard comment simuler proprement la création des tableaux dont la longueur initiale est définie par une variable. En attendant, il y a une astuce classique : si l'on sait que l'on a besoin de par exemple `n` cases avec `n` < 1000, on crée un tableau de 1000 cases qui à n'en utiliser que les `n` premières.

6. La fonction `foo` est supposée créer le tableau `[| 9; 4; 3; 8; 7; 5| ]`, le passer en argument à `est_943875` (qui est déjà codée), et renvoyer la sortie de cette dernière. Complétez `foo` pour qu'elle le fasse.

Définition 3 (Initialisateur de tableau).
On peut créer *et* initialiser un tableau `[|x0; x1; ...; xk|]` simultanément comme suit :

```
<type_des_cases> <identifiant>[] = {<x0>, <x1>, <...>, <xk>};
```

Par exemple, pour créer le tableau `[|5; 16; 8; 4; 2; 1|]`

```
int tableau[] = {5, 16, 8, 4, 2, 1};
```

On appelle `{<x0>, <x1>, <...>, <xk>}` un **initialisateur**. La longueur du tableau est calculée automatiquement à partir du nombre d'éléments de l'initialisateur.

7. Refaites la question précédente avec un initialisateur.

Remarque. On peut indiquer une longueur explicite du tableau même avec un initialisateur. Dans ce cas, toutes les cases non-initialisées (car l'initialisateur contient moins d'éléments que la longueur indiquée) sont initialisées à 0.

Ainsi, `int T[5000] = {};` crée un tableau de 5000 cases toutes initialisées à 0.

B.4 Tableaux et fonctions

Remarque. Les explications des deux blocs ci-dessous seront plus claires une fois que nous aurons eu le cours sur la mémoire. En attendant, ne retenez que les deux phrases en gras.

Définition 4 (Affaiblissement en pointeur).
Lorsqu'on passe un tableau en argument à une fonction, son contenu est passé par référence.
En effet, passer le tableau par valeur impliquerait de recopier tout son contenu ce qui serait (potentiellement) trop long. Au lieu de recevoir une copie du tableau d'origine, la fonction reçoit l'adresse du tableau d'origine (où le trouver en mémoire) : elle peut donc lire toutes les valeurs des cases, mais si elle les modifie elle modifie les originaux. On dit que le tableau est **affaibli en pointeur**.

Corollaire 5 (LES TABLEAUX NE SE RENVOIENT PAS).
⚠ **On ne peut pas renvoyer un tableau. ON NE PEUT PAS.**
En effet, pour les mêmes raisons, renvoyer un tableau provoque un affaiblissement en pointeur. Une fonction qui créerait un tableau et le renverrait renverrait donc en fait l'adresse du tableau qu'elle a créé elle-même. Sauf qu'en C, à la fin de la fonction, sa mémoire est nettoyée automatiquement et toutes les variables de la fonction sont ainsi supprimées : on renverrait donc l'adresse d'un tableau qui n'existe plus.

Remarque. Nous verrons plus tard comment créer de la mémoire qui n'est pas nettoyée automatiquement, et qui permet donc de contourner cette difficulté.

8. La fonction `double_tableau` doit doubler le contenu d'un tableau. Complétez-la.
9. La fonction `copie` doit copier le contenu du tableau `src` (de longueur `len`) dans le tableau `dest` (idem). Complétez-la.
10. La fonction `map2_min` prend deux tableaux en entrées (`T` de longueur `len` et `U` également de longueur `len`), et doit remplir le tableau `V` (lui aussi passé en argument) de sorte à ce que pour tout indice `i` des tableaux, on ait `V[i] = min(U[i], T[i])`. Complétez-la.

C Un peu de pratique

Bravo, nous avons les fondamentaux ! Nous ajouterons dans (environ) trois semaines les pointeurs, mais d'ici là l'objectif des TP sera de consolider les bases.

Voici des exercices pratiques : vous devez coder les fonctions demandées dans `partie-C/` (qui est volontairement vide : vous devez en plus réussir à écrire un fichier compilable, et non uniquement compléter un code à trous).

Vous devrez également coder des tests pour ces fonctions. Si vous souhaitez rajouter des tests interactifs, vous pouvez copier/coller les fonctions `print_tableau` et `demande_remplir_tableau` depuis `partie-B/main.c` .

11. Écrire une fonction qui prend en argument `T` un tableau, `len` sa longueur, `U` un autre tableau de même longueur, et qui renvoie `true` si et seulement si toutes les cases de `T` et `U` sont égales.

12. Écrire une fonction qui prend en argument un tableau et sa longueur, et teste si les valeurs de ce tableau peuvent être le début d'une suite arithmétique.

13. Quand on compare des mots pour savoir lequel vient avant l'autre dans le dictionnaire, on procède de la manière suivante :

- On lit les mots lettre à lettre simultanément de gauche à droite.
- Si lors de cette lecture la lettre d'un mot est inférieur (« avant dans l'alplhabet) à la lettre de l'autre, le mot ayant la lettre inférieure est déclaré inférieur à l'autre (et sera donc rangé dans le dictionnaire).
- Si on atteint la fin d'un mot mais pas de l'autre, le mot terminé est inférieur.
- Si on atteint la fin des deux mots en même temps… les mots étaient les mêmes.

On appelle cette comparaison la **comparaison lexicographique**. Implémenter une fonction qui compare lexicographiquement deux tableaux. Vous renverrez `true` si le premier tableau est strictement inférieur au second, `false` sinon.

Par exemple :

- `[|0; 5; 2; 330|]` $<$ `[|0; 4; -60|]`
- `[|0; 5|]` $<$ `[|0; 5; -100|]`