Exercice 1 : Un algorithme mystère

Dans cet exercice les tableaux de taille $n \in \mathbb{N}$ sont indicés par des entiers de $[\![0, n-1]\!]$.
On étudie dans cet exercice l'algorithme mystère ci-dessous, qui contient 3 fonctions : une fonction Mystere faisant appel à des fonctions SMystereA et SMystereB.

Algorithme 1 : Un algorithme mystère

```
1  Procedure SMystereA(i, T, n) :
       //i ∈ [0, n−1], T un tableau de taille n
2      si i = n − 1 alors
3          retourner T[i] ;
4      sinon
5          x ← SMystereA(i + 1) ;
6          si T[x] > T[i] alors
7              retourner x ;
8          sinon
9              retourner i ;
10 Procedure SMystereB(i) :
       //i ∈ [0, n−1], T un tableau de taille n
11     si i = n − 1 alors
12         retourner T[i] ;
13     sinon
14         x ← SMystereB(i + 1) ;
15         si T[x] < T[i] alors
16             retourner x ;
17         sinon
18             retourner i ;
19 Procedure Mystere(T, n) :
       //T un tableau d'entiers de taille n
20     pour i = 0 à n − 1 faire
21         si i ≡ 0[2] alors
22             Échanger le contenu de T[i] et T[SMystereA(i)] ;
23         sinon
24             Échanger le contenu de T[i] et T[SMystereB(i)] ;
```

Q. 1 Définir une fonction C, `void affiche_tableau(int tab[], int m)` prenant en paramètres un tableau d'entiers `tab` de taille `m` et affichant le tableau `tab`.

Q. 2 Implémenter une fonction C, `void smysterea(int i, int tab[], int m)` qui se comporte comme la fonction SMystereA de l'algorithme ci-dessus. Au moyen de tests de la fonction `smysterea` décrire son comportement.

Q. 3 Implémenter une fonction C, `void smystereb(int i, int tab[], int m)` qui se comporte comme la fonction SMystereB de l'algorithme ci-dessus.

Q. 4 Implémenter une fonction C, `void mystere(int tab[], int n)` qui se comporte comme la fonction Mystere de l'algorithme ci-dessus. Au moyen de tests de la fonction `mystere` décrire son comportement.

Exercice 2 : Un algorithme mystère

Dans cet exercice les tableaux de taille $n \in \mathbb{N}$ sont indicés par des entiers de $[\![0, n-1]\!]$. On note V le booléen vrai et F le booléen faux.
On étudie dans cet exercice l'algorithme mystère ci-dessous, qui contient 3 fonctions : une fonction Mystere faisant appel à des fonctions SMystereA et SMystereB.

Algorithme 2 : Un algorithme mystère

```
1  Procedure SMystereA(b, T, M) :
       //b est un booléen
       //T est un tableau de taille M
2      si b alors
3          I ← −1 ; D ← M − 1 ; F ← 0 ;
4      sinon
5          I ← 1 ; D ← 0 ; F ← M − 1 ;
6      C ← D ;
7      tant que C ≠ F faire
8          T[C] ← T[C + I] ;
9          C ← C + I ;
10 Procedure SMystereB(T, M) :
       //T est un tableau de taille M
11     si T[0] = T[1] alors
12         T[1] ← T[0] + T[1] ;
13         retourner V
14     sinon
15         retourner F
16 Procedure Mystere(n) :
       //n est un entier
17     Soit T un tableau de taille M (prendre M = 100 pour les tests) ;
18     pour i = 0 à 2^n − 1 faire
19         SMystereA(F) ;
20         T[0] ← 1 ;
21         tant que SMystereB() faire
22             SMystereA(V) ;
23     retourner T[0]
```

Q. 1 Définir une fonction C, `void affiche_tableau(int tab[], int m)` prenant en paramètres un tableau d'entiers `tab` de taille `m` et affichant le tableau `tab`.

Q. 2 Implémenter une fonction C, `void smysterea(bool b, int tab[], int m)` qui se comporte comme la fonction SMystereA de l'algorithme ci-dessus. Au moyen de tests de la fonction `smysterea` décrire son comportement.

Q. 3 Implémenter une fonction C, `bool smystereb(int tab[], int m)` qui se comporte comme la fonction SMystereB de l'algorithme ci-dessus.

Q. 4 Implémenter une fonction C, `int mystere(int n)` qui se comporte comme la fonction Mystere de l'algorithme ci-dessus. Au moyen de tests de la fonction `mystere` décrire son comportement.

Exercice 3 : Un algorithme mystère

Dans cet exercice les tableaux de taille $n \in \mathbb{N}$ sont indicés par des entiers de $[\![0, n-1]\!]$.

Algorithme 3 : Un algorithme mystère

Entrée : Un tableau T de taille n

```
1 i ← 0 ;
2 tant que i < n faire
3     si i = 0 alors
4         i ← i + 1 ;
5     sinon si T[i − 1] ⩽ T[i] alors
6         i ← i + 1 ;
7     sinon
8         échanger T[i − 1] et T[i] ;
9         i ← i − 1 ;
```

Q. 1 Définir une fonction C, `void affiche_tableau(int tab[], int m)` prenant en paramètres un tableau d'entiers `tab` de taille `m` et affichant le tableau `tab`.

Q. 2 Implémenter une fonction C, `int mystere(int tab[], int n)` qui se comporte comme la fonction Mystere de l'algorithme ci-dessus. Au moyen de tests de la fonction `mystere` décrire son comportement.

Exercice 4 : Un algorithme mystère

Dans cet exercice les tableaux de taille $n \in \mathbb{N}$ sont indicés par des entiers de $[\![0, n-1]\!]$. $a//b$ désigne le quotient de la division euclidienne de a par b.
On étudie dans cet exercice l'algorithme mystère ci-dessous.

Algorithme 4 : Un algorithme mystère

Entrée : $p \in \mathbb{N}^\star$, T un tableau de taille $n \in \mathbb{N}^\star$

```
1 δ ← 1 ;
2 pour i = 0 à p faire
3     pour k = 0 à (n//(2δ)) − 1 faire
4         si T[2kδ] < T[(2k + 1)δ] alors
5             Échanger le contenu de T[2kδ] et T[(2k + 1)δ] ;
6     δ ← 2δ ;
7 retourner T[0]
```

Q. 1 Définir une fonction C, `void affiche_tableau(int tab[], int m)` prenant en paramètres un tableau d'entiers `tab` de taille `m` et affichant le tableau `tab`.

Q. 2 Implémenter une fonction C, `void smysterea(int p, int tab[], int m)` qui se comporte comme la fonction SMystereA de l'algorithme ci-dessus. Au moyen de tests de la fonction `smysterea` sur différentes valeurs de p décrire son comportement.

Q. 3 En déduire, une fonction C, `int max(int tab[], int m)` retournant le maximum du tableau `tab` de taille `m`.

Exercice 5 : Tableaux 1

Q. 1 Donner une fonction C prenant en argument un tableau d'entiers et le modifiant pour qu'il contienne la somme cumulée (la i-ème case contient la somme des i premières cases) du tableau initial.

Exercice 6 : Tableaux 2

Q. 1 Donner une fonction C prenant en argument un tableau d'entiers et le modifiant de la manière suivante : la première case est inchangée, les cases i suivantes reçoivent la différence initiale entre la case i et la case $i-1$. Par exemple `{1, 2, 5, 1}` est transformé en `{1, 1, 3, -4}`.

Exercice 7 : Tableaux 3

Q. 1 Donner une fonction C prenant en argument un tableau d'entiers et le lissant de la manière suivante : chaque case extrême est non modifiée, les cases internes reçoivent la moyenne des deux cases voisines.

Exercice 8 : Tableaux 4

Q. 1 Définir une fonction C, prenant en paramètres un tableau d'entiers T et un entier k et effectuant une rotation du contenu des cases du tableau, de k cases vers la gauche. Le tableau de taille n devra être vu comme "torique" au sens où la case à gauche de la case d'indice 0 est la case d'indice $n-1$.

Exercice 9 : Tableaux 5

Q. 1 Définir une fonction C, prenant en paramètre un tableau d'entiers T et permutant les éléments de T de sorte qu'à la fin de l'exécution de la fonction :

- la case d'indice $T[0]$ contienne le plus grand élément du tableau initial ;
- la case d'indice $T[1]$ contienne le plus petit élément du tableau initial ;
- la case d'indice $T[2]$ contienne le deuxième plus grand élément du tableau initial ;
- la case d'indice $T[3]$ contienne le deuxième plus petit élément du tableau initial ;
- la case d'indice $T[4]$ contienne le troisième plus grand élément du tableau initial ;
- la case d'indice $T[5]$ contienne le troisième plus petit élément du tableau initial ;
- ...

Exercice 10 : Tableaux 6

Q. 1 Définir une fonction C prenant en paramètres un tableau d'entiers T de taille n et un tableau S de taille n tel que chaque entier de $[\![0, n-1]\!]$ apparaît une et une seule fois dans S. Cette fonction devra permuter les éléments du tableau T de sorte que l'élément se trouvant dans la case d'indice i avant l'exécution, se trouve dans la case d'indice $S[i]$ après exécution.

Exercice 11 : Tableaux 7

Q. 1 Définir une fonction C prenant en paramètres un tableau d'entiers T de taille impaire dont les éléments sont deux à deux distincts et retournant la médiane de ce tableau.

Exercice 12 : Structures 1

Q. 1 Définir un type structuré `date` permettant la représentation d'une date contenant les grandeurs année, mois, jour.

Q. 2 Définir une fonction `inf_date` prenant deux dates en arguments et testant si la première précède la seconde au sens large.

Q. 3 Définir un type structuré `personne` permettant la représentation d'une personne ayant : un nom, une date de naissance et une date de décès.

Q. 4 Définir une fonction `en_vie` prenant en arguments une date et une personne et testant si cette personne est en vie à cette date.

Q. 5 Définir une fonction `rencontre_possible` prenant en arguments deux personnes et testant s'il est possible que ces deux personnes se soient rencontrées.

Q. 6 Définir une fonction `nb_en_vie` prenant en arguments un tableau de personnes et une date et retournant le nombre de personnes dans le tableau qui sont en vie à la date indiquée.

Exercice 13 : Structures 2

Q. 1 Définir un type structuré `points` permettant la représentation d'un point du plan.

Q. 2 Définir un type structuré `rectangle_par` permettant la représentation d'un rectangle dont les côtés sont parallèles aux axes du plan.

Q. 3 Définir une fonction `dans_rectangle` prenant en arguments un point et un rectangle et testant si le point est dans le rectangle.

Q. 4 Définir une fonction `rectangle_intersecte` prenant en arguments deux rectangles et testant s'ils s'intersectent.

Q. 5 Définir une fonction `rectangle_intersection` prenant en arguments deux rectangles qui s'intersectent et calculant l'intersection (un rectangle).

Q. 6 Définir une fonction `translate` prenant en arguments un rectangle et un point P et retournant le rectangle obtenu par translation du rectangle d'entrée du vecteur $\overrightarrow{OP}$.

Exercice 14 : Structures 3

Q. 1 Définir un type structuré `intervalle` permettant la représentation d'un intervalle $[\![a;b]\!]$ de $\mathbb{Z}$ avec $a, b \in \mathbb{Z}^2$.

Q. 2 Définir une fonction `intervalle_intersecte` permettant de tester si deux intervalles s'intersectent.

Q. 3 Définir une fonction `intervalle_intersection` permettant le calcul de l'intersection de deux intervalles.

Q. 4 Définir une fonction `intervalle_union` permettant le calcul de l'union de deux intervalles d'intersection non nulle.

Q. 5 Définir un type structuré `sous_ensembles_z` permettant la représentation d'un ensemble d'entiers. Votre structure contiendra un tableau T de 100 intervalles et un entier x. Ce type structuré représente alors l'ensemble $\bigcup_{i=0}^{x-1}$ `T[i]`.

Q. 6 Définir une constante `vide` représentant l'ensemble vide.

Q. 7 Définir une fonction `ajout` permettant l'ajout d'un intervalle à un ensemble d'entiers. Si l'ensemble d'entiers contient déjà un intervalle qui s'intersecte avec l'intervalle à ajouter, on fera l'union de ces deux intervalles. Sinon on en ajoutera un nouveau.

Exercice 15 : Structures 4

Q. 1 Définir un type structuré `personne` contenant deux champs : une chaîne de caractères qui est le nom de la personne et un identifiant entier unique.

Q. 2 Définir un type structuré `relation` contenant deux champs : `id_a` et `id_b` qui sont des identifiants entiers.

Dans la suite de l'exercice on manipule des tableaux `personne personnes[]` et des tableaux de `relation relations[]` tels que les identifiants dans `relation` sont des identifiants de personnes dans `personnes`. Si le tableau `relations` contient une structure `s` cela indique que les personnes d'identifiants `s.id_a` et `s.id_b` se connaissent.

Q. 3 Définir une fonction prenant en paramètres une chaîne de caractères `str` et un tableau de personnes dont un des noms est `str` et retournant l'identifiant associé dans le tableau de personnes.

Q. 4 Définir une fonction prenant en paramètres deux chaînes de caractères qui sont les noms de deux personnes, un tableau de personnes et un tableau de relations et retournant si oui ou non ces deux personnes se connaissent.

Q. 5 Définir une fonction prenant en paramètres deux chaînes de caractères qui sont les noms de deux personnes, un tableau de personnes et un tableau de relations et retournant si oui ou non ces deux personnes connaissent une même troisième personne.