

# Cours d'introduction à l'informatique

## Énoncés et solutions de l'examen de seconde session 2019

### Énoncés

1. (a) Écrire une fonction C prenant en arguments deux tableaux d'entiers  $t_1$  et  $t_2$  ainsi que leur taille (supposée commune), et retournant la longueur de leur plus grand préfixe commun, c'est-à-dire le plus grand entier  $n$  tel que  $t_1[i] = t_2[i]$  pour tout  $i$  tel que  $0 \leq i < n$ .
- (b) Par la méthode des invariants, démontrer que la valeur retournée par cette fonction est correcte. Démontrer également que cette fonction se termine.

2. En langage C :

- (a) Comment déclare-t-on une procédure prenant en argument un pointeur vers un nombre réel ainsi qu'un entier non signé ?
- (b) Quelle est la valeur affichée lors de l'exécution du fragment de code suivant ? (Justifier votre réponse.)

```
int t[2] = { -2, 0 };
int *a, b;

b = *t ? ++*t : --*t;
a = &b;

printf("%d\n", a[*t + 1]);
```

3. (a) Décrire, le plus simplement possible, l'opération effectuée par la fonction C suivante.

```
long f(int t[], int nb)
{
    if (nb <= 0)
        return 0;

    return t[0] + f(t + 2, nb - 2);
}
```

- (b) Quelle est la complexité en temps de cette fonction ?
- (c) Écrire une fonction C réalisant exactement la même opération, mais sans effectuer d'appel récursif.

4. (a) Écrire un fragment de code C définissant un type structuré capable de représenter un intervalle  $[a, b]$  de nombres entiers, avec  $a \in \mathbb{Z} \cup \{-\infty\}$  et  $b \in \mathbb{Z} \cup \{+\infty\}$  tels que  $a \leq b$ .
- (b) Écrire une fonction C prenant en argument des pointeurs vers deux instances du type obtenu à la question précédente, et retournant une valeur booléenne qui indique si le premier intervalle est entièrement inclus dans le second. (Par exemple l'intervalle  $[1, 4]$  est inclus dans  $[0, +\infty]$ , mais pas dans  $[-2, 3]$ .) Il est permis de définir des fonctions auxiliaires de votre choix.

## Exemples de solutions

1. (a) Ce problème peut être résolu en comparant un à un les éléments situés à la même position dans les deux tableaux, tant que ces éléments sont égaux, et en comptant le nombre de positions pour lesquelles cette égalité est constatée. On obtient le code suivant :

```
unsigned prefixe_commun(int t1[], int t2[], taille)
{
    unsigned i;

    for (i = 0; i < taille && t1[i] == t2[i]; i++);

    return i;
}
```

- (b) Montrons d'abord que lorsque la fonction termine son exécution, la valeur qu'elle retourne est correcte. Cela revient à prouver le triplet

$$\{\text{taille} \geq 0\}$$

$$\text{for (i = 0; i < taille \&\& t1[i] == t2[i]; i++);}$$

$$\{i = \text{longueur du plus grand préfixe commun de t1[] et t2[]}\}.$$

En développant la boucle for, cela équivaut à démontrer

$$\{\text{taille} \geq 0, i = 0\}$$

```
while (i < taille && t1[i] == t2[i])
    i++;
```

$\{i = \text{longueur du plus grand préfixe commun de } \mathbf{t1}[] \text{ et } \mathbf{t2}[]\}$ .

Cherchons un invariant de boucle  $I$ . Celui-ci doit satisfaire les trois propriétés suivantes :

- $I$  doit être impliqué par la précondition ( $\mathbf{taille} \geq 0$  et  $i = 0$ ).
- Si  $I$  est vrai avant une itération de la boucle `while`, alors  $I$  doit également être vrai à la fin de cette itération.
- Après la dernière itération de la boucle,  $I$  doit impliquer la postcondition ( $i = \text{longueur du plus grand préfixe commun de } \mathbf{t1}[] \text{ et } \mathbf{t2}[]$ ).

Ces trois propriétés sont vraies pour l'invariant

$$I : \mathbf{taille} \geq 0 \text{ et } i \leq \mathbf{taille} \text{ et } \mathbf{t1}[0 : i - 1] = \mathbf{t2}[0 : i - 1],$$

où  $\mathbf{t}[a : b]$  dénote un tableau contenant les éléments de  $\mathbf{t}$  compris entre les indices  $a$  et  $b$  (inclus).

En effet :

- Initialement, on a ( $\mathbf{taille} \geq 0$  et  $i = 0$ )  $\implies I$ , car deux tableaux de longueur nulle sont forcément égaux.
- D'une itération à l'autre de la boucle, on a le triplet

$$\begin{aligned} &\{I, i < \mathbf{taille}, \mathbf{t1}[i] = \mathbf{t2}[i]\} \\ &\quad i++; \\ &\{I\}, \end{aligned}$$

qui est bien valide. En effet, si  $i'$  dénote la valeur de  $i$  après l'exécution du fragment de code, on a

$$i' = i + 1,$$

donc

$$(I \text{ et } i < \mathbf{taille} \text{ et } \mathbf{t1}[i] = \mathbf{t2}[i])$$

implique

$$(i' \leq \mathbf{taille} \text{ et } \mathbf{t1}[0 : i' - 1] = \mathbf{t2}[0 : i' - 1]).$$

- En sortie de boucle, on a

$$(I \text{ et } (i \geq \mathbf{taille} \text{ ou } \mathbf{t1}[i] \neq \mathbf{t2}[i]))$$

qui implique bien

$$(i = \text{longueur du plus grand préfixe commun de } \mathbf{t1}[] \text{ et } \mathbf{t2}[]).$$

Pour prouver cette dernière proposition, il y a deux cas à considérer :

— Si ( $I$  et  $i \geq \text{taille}$ ) est vrai, alors on a

$$i = \text{taille} \text{ et } t1[0 : \text{taille} - 1] = t2[0 : \text{taille} - 1],$$

et le plus grand préfixe commun de  $t1[]$  et  $t2[]$  est bien de longueur  $i = \text{taille}$ .

— Si ( $I$  et  $t1[i] \neq t2[i]$ ) est vrai, alors le plus grand préfixe commun de  $t1[]$  et  $t2[]$  est bien de longueur  $i$ , car on a alors

$$t1[0 : i - 1] = t2[0 : i - 1] \text{ et } t1[i] \neq t2[i].$$

Il reste à démontrer que l'exécution de la fonction se termine toujours. Cela peut se faire en considérant le variant de boucle

$$v = \text{taille} - i,$$

qui est clairement un entier non négatif (car l'invariant de boucle  $I$  implique  $i \leq \text{taille}$ ), dont la valeur décroît strictement à chaque itération de la boucle.

2. (a) `void procedure(double *r, unsigned e);`

(b) — L'évaluation de `*t` retourne la valeur contenue dans la première case du tableau pointé par `t`, c'est-à-dire  $-2$ .

— L'expression conditionnelle `*t ? ++*t : --*t` évalue donc son premier composant `++*t`, dont l'effet est d'incrémenter la valeur pointée par `t`. En d'autres termes, la première case du tableau pointé par `t` devient égale à  $-1$ .

— Cette valeur  $-1$  est écrite dans `b`.

— On écrit dans la variable `a` un pointeur vers la variable `b`.

— L'expression `a[*t + 1]` lit la deuxième case du tableau pointé par `t`, qui vaut  $0$ , et lit ensuite la case d'index correspondant dans le tableau pointé par `a`, ce qui revient à lire le contenu de la variable `b`. Ce code affiche donc  $-1$  (suivi d'un retour à la ligne).

3. (a) Cette fonction retourne la somme des éléments d'index pair du tableau `t`, supposé de taille `nb`.

(b) L'évaluation de `f(t, nb)` provoque

$$\left\lfloor \frac{\text{nb} + 3}{2} \right\rfloor$$

appels à la fonction `f`, où  $\lfloor x \rfloor$  dénote le plus grand entier inférieur ou égal à  $x$ . Chacun de ces appels nécessite l'exécution d'un nombre borné d'opérations. La complexité en temps de la fonction vaut donc

$$O\left(\left\lfloor \frac{nb+3}{2} \right\rfloor\right) = O\left(\frac{nb+3}{2}\right) = O(nb).$$

(c) `long f(int t[], int nb)`

```
{
    unsigned i;
    long      s;

    for (i = 0, s = 0; i < nb; i += 2)
        s += t[i];

    return s;
}
```

4. (a) Une solution simple consiste à représenter chaque borne de l'intervalle par un booléen indiquant si cette borne est infinie ou non, et par un entier donnant la valeur de cette borne dans le cas fini :

```
typedef struct
{
    unsigned char a_est_infini, b_est_infini;
    int          a, b;
} intervalle;
```

(b) `int est_inclus(intervalle i1, intervalle i2)`

```
{
    if (i1.a_est_infini && !i2.a_est_infini)
        return 0;

    if (!i2.a_est_infini && i1.a < i2.a)
        return 0;

    if (i1.b_est_infini && !i2.b_est_infini)
        return 0;

    return i2.b_est_infini || i1.b <= i2.b;
}
```