Cours d'introduction à l'informatique
Examen de juin 2021
Énoncés et solutions

Note : En raison des mesures sanitaires qui étaient en vigueur en 2021, cet examen était plus court qu'habituellement.

Énoncés

1. (a) Écrire en langage C une procédure prenant en arguments

   — un tableau `t` d'octets non signés,
   — le nombre `n` ≥ 0 d'octets contenu dans `t`,
   — un tableau `h` de 256 entiers de type `int`.

   L'opération effectuée par cette procédure consiste à modifier le contenu du tableau `h` de la façon suivante : après l'exécution de la procédure, pour tout $i \in [0, 255]$, `h[`i`]` doit contenir le nombre d'octets de `t` égaux à i. Par exemple, si le tableau `t` contient $[7, 0, 42, 7, 0, 7]$, alors on doit avoir `h`$[0] = 2$, `h`$[7] = 3$, `h`$[42] = 1$, et `h[`i`]` $= 0$ pour les autres valeurs de i.

   (b) Déterminer la complexité en temps de la procédure obtenue au point (a).

   (c) Par la méthode des invariants, démontrer que l'opération effectuée par la procédure obtenue au point (a) est correcte.

2. Expliquer le plus simplement possible (une phrase suffit) l'opération réalisée par cette fonction C :

```
unsigned f(char *s, char *t)
{
  unsigned n;

  for (n = 0; *s; s++, t++, n++)
    if (*t != *s)
      break;

  return n;
}
```

3. (a) Écrire en C un fragment de code définissant un type structuré permettant de représenter une sphère dans un espace à trois dimensions. Une sphère est caractérisée par les coordonnées (x, y, z) de son centre et par son rayon r. Les valeurs de x, y, z et r sont réelles.

   (b) Écrire en C une fonction prenant en arguments quatre réels x, y, z et r, et retournant un pointeur vers une représentation de la sphère correspondante, nouvellement allouée. Si la valeur fournie pour le rayon r est négative ou nulle, alors la fonction doit retourner un code d'erreur, sous la forme d'un pointeur nul.

   (c) Écrire en C une fonction prenant en arguments un tableau `s` de pointeurs vers des représentations de sphères et le nombre `nb` d'éléments de ce tableau, supposé non nul. Cette fonction doit retourner un pointeur vers la plus petite sphère pointée par les éléments de `s`. (Dans le cas où plusieurs sphères sont de plus petite taille, la fonction peut choisir arbitrairement l'une d'entre elles.)

Exemples de solutions

1. (a) On peut commencer par initialiser tous les éléments du tableau `h` à 0. Ensuite, on énumère tous les éléments du tableau `t` et on incrémente pour chacun d'entre eux la case correspondante de `h`. (En d'autres termes, si la valeur de l'élément considéré est α, alors on incrémente la case de `h` possédant l'indice α.) On obtient le code suivant.

```
void compter(unsigned char t[], unsigned n, int h[])
{
  unsigned i;

  for (i = 0; i < 256; i++)
    h[i] = 0;

  for (i = 0; i < n; i++)
    h[t[i]]++;
}
```

   (b) On détermine l'évolution du temps de calcul de la procédure en fonction de la taille `n` du tableau d'entrée `t`.

   La première boucle effectue 256 itérations quelle que soit la valeur de `n`. La seconde boucle exécute quant à elle `n` itérations.

   La complexité en temps de la procédure vaut donc $O(256 + \mathtt{n}) = O(\mathtt{n})$.

(c) On souhaite établir la validité du triplet suivant :

$$\{\texttt{n} \geq 0\}$$

```
for (i = 0; i < 256; i++)
  h[i] = 0;

for (i = 0; i < n; i++)
  h[t[i]]++;
```

$$\{\forall k \in [0, 255] : \texttt{h}[k] = \text{nombre de valeurs de } \texttt{t}[0 : \texttt{n} - 1] \text{ égales à } k\},$$

où la notation $\texttt{t}[a : b]$ dénote le sous-tableau de `t` commençant à l'indice a et se terminant à l'indice b.

Première boucle

La première boucle est chargée d'initialiser les éléments de `h`. Pour montrer que cette opération est réalisée correctement, on peut démontrer le triplet

$$\{\texttt{n} \geq 0\}$$

```
for (i = 0; i < 256; i++)
  h[i] = 0;
```

$$\{\forall k \in [0, 255] : \texttt{h}[k] = 0\}$$

Ce triplet est équivalent à :

$$\{\texttt{n} \geq 0,\ \texttt{i} = 0\}$$

```
while (i < 256)
  {
    h[i] = 0;
    i++;
  }
```

$$\{\forall k \in [0, 255] : \texttt{h}[k] = 0\}$$

Pour trouver un invariant de boucle I_1, on exprime le traitement effectué par la boucle jusqu'à une itération donnée, ce qui donne :

$$I_1 : 0 \leq \texttt{i} \leq 256 \text{ et } \forall k \in [0, \texttt{i} - 1] : \texttt{h}[k] = 0$$

Montrons que cet invariant est valide.

— Initialement, on a $\{\texttt{n} \geq 0,\ \texttt{i} = 0\} \Rightarrow I_1$.

— Pour chaque itération de la boucle, on a le triplet

$$\{I_1,\ \texttt{i} < 256\}$$
```
h[i] = 0;
i++;
```
$$\{I_1\}$$

qui est clairement valide.

— En fin de boucle, on a $\{I_1, \texttt{i} \geq 256\} \Rightarrow \{\forall k \in [0, 255] : \texttt{h}[k] = 0\}$. En effet, $\texttt{i} \geq 256$ et $\texttt{i} \leq 256$ impliquent $\texttt{i} = 256$.

Deuxième boucle

En utilisant le triplet que nous venons de démontrer pour la première boucle, il reste à établir le triplet suivant :

$$\{\texttt{n} \geq 0 \text{ et } \forall k \in [0, 255] : \texttt{h}[k] = 0\}$$
```
for (i = 0; i < n; i++)
  h[t[i]]++;
```
$$\{\forall k \in [0, 255] : \texttt{h}[k] = \text{nombre de valeurs de } \texttt{t}[0 : \texttt{n} - 1] \text{ égales à } k\}$$

Ce triplet est équivalent à :

$$\{\texttt{n} \geq 0,\ \texttt{i} = 0 \text{ et } \forall k \in [0, 255] : \texttt{h}[k] = 0\}$$
```
while(i < n)
  {
    h[t[i]]++;
    i++;
  }
```
$$\{\forall k \in [0, 255] : \texttt{h}[k] = \text{nombre de valeurs de } \texttt{t}[0 : \texttt{n} - 1] \text{ égales à } k\}$$

On trouve un candidat invariant de boucle I_2 en caractérisant le travail réalisé par la boucle jusqu'à une itération donnée :

$I_2 : \texttt{n} \geq 0$ et $0 \leq \texttt{i} \leq \texttt{n}$ et
$\forall k \in [0, 255] : \texttt{h}[k] =$ nombre de valeurs de $\texttt{t}[0 : \texttt{i} - 1]$ égales à k.

Montrons que cet invariant est valide.

— Initialement, on a bien

$$\{\texttt{n} \geq 0,\ \texttt{i} = 0 \text{ et } \forall k \in [0, 255] : \texttt{h}[k] = 0\} \Rightarrow I_2,$$

car `i` $= 0$ implique que le nombre d'éléments à considérer dans le tableau `t` est nul.

— Pour chaque itération de la boucle, on a le triplet

$$\{I_2,\ \texttt{i} < \texttt{n}\}$$
```
h[t[i]]++;
i++;
```
$$\{I_2\}$$

qui est valide : le seul élément du tableau `h` qui change de valeur est celui d'indice `t`[`i`].

— En fin de boucle, on a bien

$$\{I_2, \texttt{i} \geq \texttt{n}\} \Rightarrow \forall k \in [0, 255] : \texttt{h}[k] = \text{nombre de valeurs de } \texttt{t}[0 : \texttt{n} - 1] \text{ égales à } k.$$

2. Cette fonction retourne la longueur du plus grand préfixe commun aux chaînes de caractères pointées par `s` et `t` (en d'autres termes, le plus grand nombre de caractères consécutifs situés au début de ces chaînes qui sont égaux deux à deux).

3. (a)
```
struct sphere
{
  double x, y, z, r;
};
```

(b)
```
#include <stdlib.h>

struct sphere *nouvelle_sphere(double x, double y,
                               double z, double r)
{
  struct sphere *s;

  if (r < 0.0)
    return NULL;

  s = malloc(sizeof(struct sphere));
```

```
  if (!s)
    return NULL;

  s -> x = x;
  s -> y = y;
  s -> z = z;
  s -> r = r;

  return s;
}
```

(c)
```
struct sphere *plus_petite_sphere(struct sphere *s[], unsigned nb)
{
  unsigned i, i_min;

  i_min = 0;

  for (i = 1; i < nb; i++)
    if (s[i] -> r < s[i_min] -> r)
      i_min = i;

  return s[i_min];
}
```