Cours d'introduction à l'informatique
Examen de septembre 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` de nombres réels,
   — le nombre $\texttt{n} \geq 0$ d'éléments de `t`.

   L'opération effectuée par cette procédure consiste à modifier le contenu du tableau `t` de la façon suivante : si t_0, t_1, t_2, ... sont (respectivement) les contenus initiaux des cases `t[0]`, `t[1]`, `t[2]`, ... de `t`, alors après l'exécution de la procédure, la première case de `t` doit contenir t_0, la deuxième $t_0 + t_1$, la troisième $t_0 + t_1 + t_2$, et ainsi de suite.

   (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(unsigned v[], unsigned nb, unsigned m)
{
 int b;

 if (!nb)
   return 0;

 b = (*v % m) ? 0 : 1;

 return f(v + 1, nb - 1, m) + b;
}
```

3. (a) Écrire en C un fragment de code définissant un type structuré représentant un tableau bidimensionnel de chaînes de 100 caractères ou moins. La dimension de ce tableau, c'est-à-dire son nombre de lignes et son nombre de colonnes, doit être retenue dans la structure.

   (b) Écrire en C une fonction prenant en arguments un nombre de lignes n et un nombre de colonnes m, et retournant un pointeur vers une structure nouvellement allouée (de même type que votre réponse à (a)), contenant un tableau de $n \times m$ chaînes initialement vides.

   (c) Écrire en C une fonction prenant en arguments deux pointeurs vers des représentations de tableaux (de même type que votre réponse à (a)), et retournant une valeur booléenne indiquant si ces tableaux sont égaux. Deux tableaux sont égaux s'ils possèdent la même dimension, et si pour chaque position (i, j), les chaînes figurant à cette position dans les deux tableaux ont un contenu identique.

Exemples de solutions

1. (a) On peut résoudre le problème en parcourant les éléments du tableau de gauche à droite, et en maintenant un accumulateur qui est en permanence égal à la somme des éléments déjà visités.

```
void modifier_somme(double t[], unsigned n)
{
  double a = 0.0;
  unsigned i;

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

   (b) Pour une valeur donnée de `n`, la fonction effectue `n` itérations qui s'exécutent chacune en temps constant. La complexité en temps de la fonction est donc $O(\texttt{n})$.

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

$$\{\mathtt{n} \geq 0 \text{ et } \mathtt{a} = 0 \text{ et } \mathtt{t} = [t_0, t_1, \ldots, t_{\mathtt{n}-1}]\}$$

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

$$\{\mathtt{t} = [t_0, t_0 + t_1, t_0 + t_1 + t_2, \ldots, t_0 + t_1 + \cdots + t_{\mathtt{n}-1}]\}$$

En décomposant la boucle `for`, on obtient le triplet équivalent

$$\{\mathtt{n} \geq 0 \text{ et } \mathtt{a} = 0 \text{ et } \mathtt{i} = 0 \text{ et } \mathtt{t} = [t_0, t_1, \ldots, t_{\mathtt{n}-1}]\}$$

```
while (i < n)
  {
    a += t[i];
    t[i] = a;
    i++;
  }
```

$$\{\mathtt{t} = [t_0, t_0 + t_1, t_0 + t_1 + t_2, \ldots, t_0 + t_1 + \cdots + t_{\mathtt{n}-1}]\}$$

Pour trouver un invariant de boucle I, on caractérise le traitement effectué par la boucle jusqu'à une itération donnée. Un invariant possible est

$$\begin{aligned} I : &\mathtt{n} \geq 0 \text{ et } 0 \leq \mathtt{i} \leq \mathtt{n} \text{ et } \mathtt{a} = t_0 + t_1 + t_2 + \cdots + t_{\mathtt{i}-1} \\ &\text{et } \mathtt{t} = [t_0, t_0 + t_1, \ldots, t_0 + t_1 + \cdots + t_{\mathtt{i}-1}, t_{\mathtt{i}}, t_{\mathtt{i}+1}, \ldots, t_{\mathtt{n}-1}]. \end{aligned}$$

Cet invariant exprime qu'au début et à la fin de chaque itération, `a` contient la somme de tous les éléments déjà visités. En outre, les `i` premiers éléments du tableau `t` contiennent les sommes partielles déjà calculées, et les éléments suivants ont gardé leur valeur initiale.

Montrons maintenant que cet invariant est valide.

— Initialement, on a $\mathtt{n} \geq 0$ par hypothèse, ainsi que $\mathtt{i} = 0$, $\mathtt{a} = 0$ et $\mathtt{t} = [t_0, t_1, t_2, \ldots, t_{\mathtt{n}-1}]$. L'invariant est donc bien satisfait.

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

$\{I,\ \mathtt{i} < \mathtt{n}\}$

```
a += t[i];
t[i] = a;
i++;
```

$\{I\}$

Montrons que ce triplet est valide, en notant respectivement $\mathtt{x}$ et $\mathtt{x}'$ la valeur d'une variable x avant et après l'itération concernée.

— On a $\mathtt{n}' = \mathtt{n}$ et $\mathtt{i}' = \mathtt{i} + 1$. Des contraintes $\mathtt{i} \geq 0$ et $\mathtt{i} < \mathtt{n}$, on déduit $\mathtt{i}' \geq 0$ (et même $\mathtt{i}' > 0$) et $\mathtt{i}' \leq \mathtt{n}'$.

— On a $\mathtt{a}' = \mathtt{a} + \tau$, où τ est la valeur de l'élément d'indice i du tableau t. Cela entraîne

$$\begin{aligned}\mathtt{a}' &= t_0 + t_1 + t_2 + \cdots + t_{\mathtt{i}}\\ &= t_0 + t_1 + t_2 + \cdots + t_{\mathtt{i}'-1}.\end{aligned}$$

— L'itération laisse les éléments du tableau t inchangés, à l'exception de celui d'indice i qui est remplacé par la valeur de $\mathtt{a}'$. Par conséquent, on a

$$\begin{aligned}\mathtt{t}' &= [t_0, t_0 + t_1, \ldots, t_0 + t_1 + \cdots + t_{\mathtt{i}'-1}, t_{\mathtt{i}+1}, t_{\mathtt{i}+2}, \ldots, t_{\mathtt{n}-1}]\\ &= [t_0, t_0 + t_1, \ldots, t_0 + t_1 + \cdots + t_{\mathtt{i}'-1}, t_{\mathtt{i}'}, t_{\mathtt{i}'+1}, \ldots, t_{\mathtt{n}'-1}]\end{aligned}$$

La postcondition du triplet est donc bien vérifiée.

— En fin de boucle, on a $\{I,\ \mathtt{i} \geq \mathtt{n}\}$, qui implique $\mathtt{i} = \mathtt{n}$. On a bien alors

$$\mathtt{t} = [t_0, t_0 + t_1, t_0 + t_1 + t_2, \ldots, t_0 + t_1 + \cdots + t_{\mathtt{n}-1}].$$

2. Cette fonction compte le nombre d'éléments du tableau n, de taille nb, qui sont divisibles par m.

3.
```
struct chaine
{
  char c[101];
};
```

```c
struct tableau
{
  struct chaine **elements;
  unsigned nb_lignes, nb_colonnes;
};
```

4.
```c
#include <stdlib.h>

struct tableau *creer_tableau(unsigned n, unsigned m)
{
  struct tableau *t;
  unsigned i, j;

  t = malloc(sizeof(struct tableau));
  if (!t)
    return NULL;

  t -> nb_lignes = n;
  t -> nb_colonnes = m;

  t -> elements = malloc(n * sizeof(struct chaine *));
  if (!t -> elements)
    {
      free(t);
      return NULL;
    }

  for (i = 0; i < n; i++)
    {
      t -> elements[i] = malloc(m * sizeof(struct chaine));
      if (!t -> elements[i])
        {
          for (j = 0; j < i; j++)
            free(t -> elements[j]);

          free(t -> elements);
          free(t);

          return NULL;
        }
```

```
    for (j = 0; j < m; j++)
      t -> elements[i][j].c[0] = '\0';
  }

  return t;
}
```

5.
```
int tableaux_egaux(struct tableau *t1, struct tableau *t2)
{
  unsigned i, j, k;

  if (t1 -> nb_lignes != t2 -> nb_lignes ||
      t1 -> nb_colonnes != t2 -> nb_colonnes)
    return 0;

  for (i = 0; i < t1 -> nb_lignes; i++)
    for (j = 0; j < t1 -> nb_colonnes; j++)
      for (k = 0; k < 101; k++)
        {
          if (!t1 -> elements[i][j].c[k] &&
              !t2 -> elements[i][j].c[k])
            break;

          if (t1 -> elements[i][j].c[k] !=
              t2 -> elements[i][j].c[k])
            return 0;
        }

  return 1;
}
```