Cours d'introduction à l'informatique
Énoncés et solutions de l'examen de janvier 2019

Énoncés

Livres fermés. Durée : 3 h 30. Veuillez répondre aux questions sur des feuilles séparées sur lesquelles figurent votre nom et votre section. Les réponses doivent être justifiées. Soyez bref et concis, mais précis.

1. (a) Écrire une fonction C prenant en arguments deux tableaux d'entiers ainsi que la longueur (supposée commune) de ces tableaux, retournant 1 si ces tableaux possèdent le même contenu, et 0 sinon.

   (b) Par la méthode des invariants, démontrer que la valeur retournée par cette fonction est correcte.

2. En langage C :

   (a) Comment déclare-t-on un tableau à 10 lignes et 20 colonnes, dont chaque élément est un pointeur vers un nombre réel ?

   (b) Quelle est la valeur affichée lors de l'exécution du fragment de code suivant ? (Justifier votre réponse.)

```
int a[] = { 1, 0 };
int *b[2];
int **c;

b[*a] = a + 1;
c = b;

printf("%d\n", **++c);
```

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

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

  if (!s[0])
    return 0;
```

```
  if (s[0] >= '0' && s[0] <= '9')
    n = 1;
  else
    n = 0;

  return n + f(++s);
}
```

(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 les deux types structurés suivants :
   — un type `point` représentant les coordonnées cartésiennes (x, y) d'un point dans le plan, où x et y sont des valeurs réelles.
   — un type `segment` représentant un segment de droite caractérisé par son origine et sa destination (qui sont des points).

   (b) Écrire une fonction C prenant en arguments quatre réels x_1, y_1, x_2 et y_2 définissant les extrémités (x_1, y_1) et (x_2, y_2) d'un segment de droite, et retournant un pointeur vers une représentation de ce segment, nouvellement allouée.

   (c) Écrire une fonction C permettant de libérer une représentation d'un segment créée par la fonction obtenue au point (b).

Exemples de solutions

1. (a) Il suffit de parcourir les deux tableaux élément par élément, et de retourner 0 dès que l'on constate la première différence. Si le parcours se termine sans avoir détecté de différence, alors les tableaux sont égaux et la fonction doit retourner 1. On obtient le code suivant :

```
int compare(int t1[], int t2[], unsigned longueur)
{
  unsigned i;

  for (i = 0; i < longueur; i++)
    if (t1[i] != t2[i])
      return 0;
```

```
  return 1;
}
```

(b) On demande uniquement de démontrer que la valeur retournée par la fonction `compare` est correcte, et non de prouver que cette fonction se termine.

Si `t1[]` et `t2[]` sont deux tableaux de taille `longueur`, cela revient à démontrer le triplet

$$\{\texttt{longueur} \geq 0\}$$

```
for (i = 0; i < longueur; i++)
  if (t1[i] != t2[i])
    return 0;
return 1;
```

$\{$ La valeur retournée est 0 $\Rightarrow$ Les tableaux `t1[]` et `t2[]` sont différents.
La valeur retournée est 1 $\Rightarrow$ Les tableaux `t1[]` et `t2[]` sont égaux. $\}$

Le plus simple consiste à déterminer un invariant de boucle I qui vérifie les trois propriétés suivantes :

— I est vrai avant la première itération de la boucle `for`,

— Si I est vrai avant une itération de cette boucle, alors I est également vrai après celle-ci, et

— Après la dernière itération de la boucle, I implique la propriété souhaitée.

Un invariant possible est le suivant :

$$I : 0 \leq \texttt{i} \leq \texttt{longueur} \text{ et } \forall k \in [0, i-1] : \texttt{t1}[k] = \texttt{t2}[k]$$

En effet :

— Avant la première itération de la boucle, on a bien $\{\texttt{longueur} \geq 0, \texttt{i} = 0\} \Rightarrow I$.

— D'une itération à l'autre de la boucle, on a le triplet

$$\{I, \texttt{i} < \texttt{longueur}\}$$

```
if (t1[i] != t2[i])
  return 0;
i++;
```

$$\{I\},$$

qui est bien valide. En effet :

- si `t1[i]` = `t2[i]`, alors `i` est incrémenté et la condition $\forall k \in [0, \texttt{i} - 1] : \texttt{t1}[k] = \texttt{t2}[k]$ reste vraie.
- si `t1[i]` $\neq$ `t2[i]`, alors ce code retourne 0 sans avoir modifié `i`.

— Lorsqu'on sort de la boucle :

- Soit la condition `t1[i] != t2[i]` est vraie, auquel cas les tableaux `t1[]` et `t2[]` sont différents et la fonction retourne 0,
- Soit le gardien de boucle `i < longueur` est faux ; la fonction retourne alors 1. Dans ce cas, on a

$$\begin{aligned} \{I, \texttt{i} \geq \texttt{longueur}\} &\Rightarrow \{I, \texttt{i} = \texttt{longueur}\} \\ &\Rightarrow \forall k \in [0, \texttt{longueur} - 1] : \texttt{t1}[k] = \texttt{t2}[k], \end{aligned}$$

qui montre que les tableaux `t1[]` et `t2[]` sont égaux.

2. (a) `double *tableau[10][20];`

(b)
- `a + 1` s'évalue en un pointeur vers le deuxième élément du tableau `a`.
- `*a` retourne le premier élément du tableau `a`, qui vaut 1.
- On en déduit que l'instruction `b[*a] = a + 1;` place dans la deuxième case du tableau `b` (celle d'index 1) un pointeur vers le deuxième élément de `a`.
- L'instruction `c = b;` fait pointer `c` vers le premier élément de `b`.
- L'expression `c++` incrémente `c`, qui pointe alors vers le deuxième élément de `b`.
- La première application de l'opérateur `*` retourne donc la valeur du deuxième élément de `b`, qui est un pointeur vers le deuxième élément de `a`.
- La seconde application de `*` retourne alors la valeur du deuxième élément de `a`, qui est égale à 0. Ce fragment de code affiche donc "`0`" (suivi par un retour à la ligne).

3. (a) Cette fonction parcourt un à un les caractères qui composent la chaîne `s`, et calcule la somme d'une valeur qui vaut 1 lorsqu'un caractère est compris entre `'0'` et `'9'`, et 0 sinon. On en déduit que cette fonction compte le nombre de chiffres présents dans la chaîne `s`.

(b) Le nombre d'appels à la fonction est égal à $n+1$, où n est la longueur de la chaîne `s` (c'est-à-dire le nombre de caractères qu'elle contient), et chacun de ces appels s'exécute en temps $O(1)$. Le temps de calcul total est donc $O(n+1) = O(n)$.

(c) Il suffit d'effectuer l'opération de comptage des chiffres dans une boucle plutôt que par récursion :

```
unsigned f(char *s)
{
  unsigned nb;

  for (nb = 0; *s; s++)
    if (*s >= '0' && *s <= '9')
      nb++;

  return nb;
}
```

4. (a)
```
typedef struct
{
  double x, y;
} point;

typedef struct
{
  point origine, destination;
} segment;
```

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

segment *creer_segment(double x1, double y1,
                       double x2, double y2)
{
  segment *p;

  p = malloc(sizeof(segment));
  if (!p)
    return NULL;

  p -> origine.x = x1;
  p -> origine.y = y1;
```

```
  p -> destination.x = x2;
  p -> destination.y = y2;

  return p;
}
```

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

void liberer_segment(segment *p)
{
  free(p);
}
```