Cours d'introduction à l'informatique
Examen d'août 2023
Énoncés et solutions

Énoncés

1. (a) Écrire une fonction prenant en arguments un pointeur s vers une chaîne de caractères, et un caractère c, et retournant le nombre de copies de c situées au début de la chaîne. Par exemple, pour la chaîne `"aaabbaba"` et le caractère `'a'`, la fonction doit retourner 3. Pour cette même chaîne et le caractère `'b'` ou `'c'`, la fonction doit retourner 0.

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

2. (a) Écrire une fonction calculant le plus petit diviseur strictement supérieur à 1 d'un nombre $n > 1$ donné. Par exemple, pour $n = 51$ et $n = 11$, la fonction doit respectivement retourner 3 et 11. On demande que l'implémentation de cette fonction soit raisonnablement efficace, c'est-à-dire que sa complexité en temps soit meilleure que $O(n)$.

   (b) Calculer la complexité en temps de la fonction obtenue au point (a).

3. On décide de représenter une séquence de chaînes de caractères par une structure de données composée d'un tableau de pointeurs vers ces chaînes et du nombre de chaînes contenues dans la séquence. La taille du tableau est fixée et le nombre de chaînes peut varier d'une séquence à une autre. L'ordre des pointeurs dans le tableau correspond à l'ordre des chaînes dans la séquence représentée.

   (a) Écrire un fragment de code définissant un type structuré correspondant à cette structure de données. La taille du tableau peut être librement choisie.

   (b) Écrire une fonction recevant en arguments deux pointeurs vers des séquences de chaînes de caractères, représentées grâce à la structure définie au point (a), et retournant une valeur booléenne indiquant si ces séquences sont identiques. Deux séquences sont considérées comme étant identiques si elles sont composées du même nombre de chaînes, et si ces chaînes sont égales deux à deux (c'est-à-dire, si les premières chaînes

de chaque séquence sont égales, si les deuxièmes sont égales, et ainsi de suite).
Note : Pour comparer les chaînes de caractères, il est permis d'utiliser la fonction `strcmp` de la bibliothèque standard : `strcmp(s1, s2)` retourne 0 si les chaînes pointées par `s1` and `s2` sont égales, et une valeur non nulle sinon. Le fichier d'en-tête correspondant est `string.h`.

4. (a) Expliquer, le plus simplement possible, ce que calcule la fonction suivante :

```
double f(double x, unsigned n)
{
  double a;

  if (!n)
    return x;

  a = f(x, n - 1);

  return a * a;
}
```

   (b) Écrire une fonction réalisant exactement la même opération, mais sans effectuer d'appel récursif. *Note :* Il n'est pas permis d'utiliser des fonctions issues de la bibliothèque standard.

5. Écrire une fonction prenant en arguments deux tableaux d'entiers `t1` et `t2` ainsi que la taille (commune) `n`, supposée non nulle, de ces tableaux. Cette fonction doit retourner un pointeur vers un tableau nouvellement alloué de taille 2`n`, contenant l'entrelacement des éléments de `t1` et `t2` : si `t1` contient les éléments `t1[0]`, `t1[1]`, ..., `t1[n − 1]` et `t2` les éléments `t2[0]`, `t2[1]`, ..., `t2[n − 1]`, alors le tableau construit par la fonction doit contenir les éléments `t1[0]`, `t2[0]`, `t1[1]`, `t2[1]`, ..., `t1[n − 1]`, `t2[n − 1]`, dans cet ordre. En cas d'erreur, la fonction doit retourner le pointeur vide.

Exemples de solutions

1. (a) Le nombre de copies de c situées au début de la chaîne peut être compté grâce à une simple boucle parcourant la chaîne s, qui se termine lorsqu'on rencontre un caractère différent de c, ou le terminateur qui correspond à un cas particulier de cette condition. On obtient le code suivant.

```
unsigned nb_copies(char *s, char c)
{
  unsigned i;

  for (i = 0; s[i] == c; i++);

  return i;
}
```

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

$$\{\texttt{c} \neq \texttt{'\0'}\}$$
$$\texttt{for (i = 0; s[i] == c; i++);}$$
$$\{\texttt{i} = \text{nombres de copies de } \texttt{c} \text{ situées au début de } \texttt{s}\}$$

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

$$\{\texttt{c} \neq \texttt{'\0'},\ \texttt{i} = 0\}$$
$$\texttt{while (s[i] == c)}$$
$$\texttt{i++;}$$
$$\{\texttt{i} = \text{nombres de copies de } \texttt{c} \text{ situées au début de } \texttt{s}\}$$

Pour trouver un invariant de boucle I, on caractérise les opérations effectuées par la boucle jusqu'à une itération donnée. Un invariant possible est

$$I : \texttt{c} \neq \texttt{'\0'} \text{ et } \texttt{i} \geq 0 \text{ et } \forall k \in [0, \texttt{i} - 1] : \texttt{s[}k\texttt{]} = \texttt{c}$$

Cet invariant exprime qu'avant et après chaque itération, tous les caractères de `s` qui ont été lus par les itérations déjà effectuées sont égaux au caractère `c`.

Montrons que cet invariant est valide.

— Initialement, on a $\texttt{c} \neq \texttt{'\0'}$ et $\texttt{i} = 0$ grâce à la précondition, ce qui satisfait I.

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

$$\{I,\ \texttt{s[i]} = \texttt{c}\}$$
$$\texttt{i++;}$$
$$\{I\}$$

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

On a $\texttt{i}' = \texttt{i}+1$. Étant donné que l'invariant implique $\texttt{i} \geq 0$, on a donc bien $\texttt{i}' \geq 0$. De plus, l'invariant implique $\forall k \in [0, \texttt{i} - 1] : \texttt{s}[k] = \texttt{c}$. En combinant cela avec la précondition, on obtient $\forall k \in [0, \texttt{i}' - 1] : \texttt{s}[k] = \texttt{c}$, ce qui entraîne que l'invariant est satisfait à l'issue de l'itération.

— En fin de boucle, on a $\{I, \texttt{s[i]} \neq \texttt{c}\}$. On a donc $\forall k \in [0, \texttt{i} - 1] : \texttt{s}[k] = \texttt{c}$, et $\texttt{s[i]} \neq \texttt{c}$, ce qui signifie que `i` est égal au nombre de copies de `c` situées au début de `s`.

Il reste à démontrer que la boucle se termine. Il suffit pour cela de considérer le variant de boucle $\ell - \texttt{i}$, où ℓ est la longueur de la chaîne `s`. Étant donné que l'invariant I implique que $\texttt{s}[k]$ est différent du caractère terminateur pour tout k tel que $0 \leq k < \texttt{i}$, ce variant possède toujours une valeur entière non négative. De plus, chaque itération de la boucle incrémente `i`, et diminue donc la valeur du variant.

2. (a) On sait que le plus petit diviseur en question est soit égal à n, soit inférieur ou égal à $\sqrt{n}$. Il suffit alors d'énumérer les candidats diviseurs du plus petit au plus grand et de tester s'ils divisent n. On obtient le code suivant.

```
unsigned plus_petit_diviseur(unsigned n)
{
  unsigned d;

  for (d = 2; d * d <= n; d++)
    if (!(n % d))
      return d;

  return n;
}
```

(b) La boucle effectue au plus $\sqrt{n} - 1$ itérations. La complexité en temps de la fonction est donc $O(\sqrt{n})$.

3. (a)
```
#define TAILLE 100

struct sequence
{
  char *chaines[TAILLE];
  unsigned nb_chaines;
};
```

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

int sequences_identiques(struct sequence *s1, struct sequence *s2)
{
  unsigned i;

  if (s1 -> nb_chaines != s2 -> nb_chaines))
    return 0;

  for (i = 0; i < s1 -> nb_chaines; i++)
    if (strcmp(s1 -> chaines[i], s2 -> chaines[i])
      return 0;

  return 1;
}
```

4. (a) Cette fonction calcule x^{2^n}.

(b)
```
double f(double x, unsigned n)
{
  unsigned i;

  for (i = 0; i < n; i++)
    x *= x;

  return x;
}
```

5.
```
#include <stdlib.h>

int *entrelacer(int t1[], int t2[], unsigned n)
{
  int *t;
```

```
  unsigned i;

  t = malloc(2 * n * sizeof(int));
  if (!t)
    return NULL;

  for (i = 0; i < n; i++)
    {
      t[2 * i] = t1[i];
      t[2 * i + 1] = t2[i];
    }

  return t;
}
```