Cours d'introduction à l'informatique
Examen de janvier 2022
Énoncés et solutions

Énoncés

1. (a) Écrire une fonction prenant en argument une chaîne de caractères, et retournant la somme de tous les chiffres qu'elle contient. Par exemple, pour la chaîne `"Le 24 janvier 2022"`, cette fonction doit retourner 12. Dans le cas particulier d'une chaîne qui ne contient aucun chiffre, la fonction doit retourner 0.

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

2. (a) Écrire une fonction prenant en argument un entier $n > 1$, et retournant le plus grand diviseur d de n tel que $d < n$. Par exemple, cette fonction appliquée à $n = 21$ doit retourner 7. On souhaite que l'implémentation de cette fonction soit raisonnablement efficace.

   (b) Déterminer les complexités en temps et en espace de la fonction obtenue au point (a).

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

```
double f(double t[], double a, unsigned n)
{
  if (a < 0.0)
    return f(t, -a, n);

  if (n)
   return f(t + 1, a, --n);

  return a * t[0];
}
```

   (b) Écrire une fonction réalisant exactement la même opération, mais n'effectuant aucun appel récursif et ne comprenant aucune instruction de boucle.

4. (a) Écrire un fragment de code définissant un type structuré capable de représenter un tableau bidimensionnel d'entiers dont chaque ligne est de taille quelconque. (Cela signifie en particulier que deux lignes distinctes de ce tableau peuvent être de taille différente.) La taille de chaque ligne du tableau doit être retenue dans la structure.

   (b) Écrire une fonction prenant en arguments un vecteur `tailles` d'entiers non signés et un entier non signé `nb`, et retournant un pointeur vers une représentation nouvellement allouée d'un tableau à `nb` lignes, dans lequel la taille de la ligne d'indice i est égale à `tailles[`i`]`, pour $i = 0$, 1, 2, ..., `nb` $- 1$. Tous les éléments de ce tableau doivent être initialisés à 0. En cas d'erreur, la fonction doit retourner un pointeur vide.

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

   (d) Écrire le prototype d'une fonction capable de déterminer si deux tableaux bidimensionnels d'entiers possèdent la même forme, c'est-à-dire le même nombre de lignes et des lignes mutuellement de même taille. Les deux tableaux à comparer sont fournis sous la forme de pointeurs vers leur représentation, du type obtenu au point (a), et la fonction doit retourner une valeur booléenne. On souhaite que la portée de cette fonction soit limitée au fichier source dans lequel elle est définie. On ne demande pas de fournir l'implémentation de cette fonction.

Exemples de solutions

1. (a) Il suffit de parcourir la chaîne de caractères reçue en argument et, pour chaque caractère de celle-ci, tester s'il est un chiffre. Le cas échéant, on calcule la valeur numérique de ce chiffre et on l'ajoute à la valeur courante du total. On obtient le code suivant.

```
unsigned total_chiffres(char *c)
{
  unsigned total;

  for (total = 0; *c; c++)
    if (*c >= '0' && *c <= '9')
      total += *c - '0';

  return total;
}
```

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

$\{\texttt{c} = \texttt{c}_0\}$

```
for (total = 0; *c; c++)
  if (*c >= '0' && *c <= '9')
    total += *c - '0';
```

$\{\texttt{total} = \text{total des chiffres contenus dans la chaîne } \texttt{c}_0\}$

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

$\{\texttt{c} = \texttt{c}_0, \texttt{total} = 0\}$

```
while (*c)
{
  if (*c >= '0' && *c <= '9')
    total += *c - '0';
  c++;
}
```

$\{\texttt{total} = \text{total des chiffres contenus dans la chaîne } \texttt{c}_0\}$

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

$I : \texttt{c} \geq \texttt{c}_0$ et $\texttt{total} =$ total des chiffres contenus dans la chaîne
$*\texttt{c}_0 \ *(\texttt{c}_0 + 1) \ *(\texttt{c}_0 + 2) \ \ldots \ *(\texttt{c} - 1)$
et $\forall d : \ \texttt{c}_0 \leq d < \texttt{c} \ \Rightarrow \ *d \neq 0$

En d'autres termes, cet invariant exprime qu'avant et après chaque itération de la boucle, la valeur de `total` correspond au total des chiffres appartenant à un préfixe de la chaîne fournie à la fonction. Ce préfixe commence à l'endroit désigné par le pointeur $\texttt{c}_0$ passé en argument à la fonction, et se termine immédiatement avant le caractère pointé par la valeur courante de `c`. L'invariant exprime aussi que ce préfixe ne contient aucun caractère de terminaison (c'est-à-dire nul).

Montrons maintenant que cet invariant est valide.

— Initialement, on a $\texttt{c} = \texttt{c}_0$, et donc le préfixe considéré est vide. On a alors bien $\texttt{total} = 0$.

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

$$\{I,\ *\mathtt{c} \neq 0\}$$

```
if (*c >= '0' && *c <= '9')
  total += *c - '0';
c++;
```

$$\{I\}$$

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

— Dans tous les cas, on a $\mathtt{c}' = \mathtt{c} + 1$, et la chaîne de caractères $*\mathtt{c}_0\ *(\mathtt{c}_0{+}1)\ *(\mathtt{c}_0{+}2)\ \ldots\ *(\mathtt{c}'{-}1)$ est donc égale à la concaténation de la chaîne $*\mathtt{c}_0\ *(\mathtt{c}_0 + 1)\ *(\mathtt{c}_0 + 2)\ \ldots\ *(\mathtt{c} - 1)$ et du caractère $*\mathtt{c}$. La précondition implique que ce caractère $*\mathtt{c}$ n'est pas nul.

— Si $*\mathtt{c}$ est un chiffre, alors l'instruction située dans le `if` s'exécute, et on a $\mathtt{total}' = \mathtt{total} + \gamma$, où γ est la valeur numérique du chiffre en question. La postcondition est satisfaite.

— Si $*\mathtt{c}$ n'est pas est un chiffre, alors l'instruction située dans le `if` ne s'exécute pas , et on a $\mathtt{total}' = \mathtt{total}$. La postcondition est aussi satisfaite dans ce cas.

— En fin de boucle, on a $\{I,\ *\mathtt{c} = 0\}$, ce qui implique que la valeur de `total` est égale au total des chiffres contenus dans la chaîne de caractères commençant à l'endroit pointé par $\mathtt{c}_0$, et se terminant au premier caractère terminateur (qui est celui pointé par `c`).

2. (a) Pour énumérer les diviseurs de n, on peut utiliser la même stratégie que celle vue au cours pour la recherche de nombres parfaits : il suffit de considérer tous les entiers contenus dans l'intervalle $[2, \sqrt{n}\,]$, et de trouver le plus petit d'entre eux qui divise n. Si ce nombre est noté i, alors le diviseur recherché vaut $d = n/i$. Si aucun diviseur n'est trouvé, alors on a $d = 1$. On obtient alors le code suivant.

```
unsigned plus_grand_diviseur(unsigned n)
{
  unsigned i;

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

  return 1;
}
```

(b) Le nombre d'itérations effectuées est borné par $\sqrt{n}$, donc la complexité en temps de cette fonction est $O(\sqrt{n})$.
La fonction consomme un espace mémoire de taille constante, donc sa complexité en espace est $O(1)$.

3. (a) Cette fonction calcule le produit de l'élément du tableau `t` situé à la position `n` et de la valeur absolue de `a`.

(b)
```
double f(double t[], double a, unsigned n)
{
  if (a < 0.0)
    a = -a;

  return a * t[n];
}
```

4. (a)
```
struct tableau_2d
{
  unsigned nb_lignes;
  unsigned *taille_lignes;
  int **elements;
};
```

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

struct tableau_2d *creer_tableau_2d(unsigned tailles[],
                                    unsigned nb)
{
  struct tableau_2d *t;
  unsigned i, j;
```

```
  t = malloc(sizeof(struct tableau_2d));
  if (!t)
    return NULL;

  t -> nb_lignes = nb;
  t -> taille_lignes = malloc(nb * sizeof(unsigned));
  if (!t -> taille_lignes)
  {
    free(t);
    return NULL;
  }

  t -> elements = malloc(nb * sizeof(int *));
  if (!t -> elements)
  {
    free(t -> taille_lignes);
    free(t);
    return NULL;
  }

  for (i = 0; i < nb; i++)
  {
    t -> taille_lignes[i] = tailles[i];
    t -> elements[i] = malloc(tailles[i] * sizeof(int));
    if (!t -> elements[i])
    {
      for (j = 0; j < i; j++)
        free(t -> elements[j]);

      free(t -> elements);
      free(t -> taille_lignes);
      free(t);
      return NULL;
    }

   for (j = 0; j < tailles[i]; j++)
     t -> elements[i][j] = 0;
  }
  return t;
}
```

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

void liberer_tableau_2d(struct tableau_2d *t)
{
  unsigned i;

  for (i = 0; i < t -> nb_lignes; i++)
    free(t -> elements[i]);

  free(t -> elements);
  free(t -> taille_lignes);
  free(t);
}
```

(d)
```
static int meme_forme(struct tableau_2d *,
                      struct tableau_2d *);
```