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

Énoncés

1. (a) Écrire une fonction prenant en argument un nombre entier non signé, et retournant la somme de tous les chiffres de son écriture décimale. Par exemple, pour le nombre 2022, cette fonction doit retourner 6.

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

2. (a) Écrire une fonction prenant en arguments un tableau `t` d'entiers non signés et le nombre d'éléments `n` de ce tableau, et retournant le plus grand entier présent dans au moins deux cases consécutives de `t`, ou 0 si `t` ne contient pas de valeurs consécutives identiques.Par exemple, pour le tableau $[10, 1, 4, 4, 4, 0, 7, 7, 10]$, cette fonction doit retourner 7. Pour le tableau $[4, 7, 2, 7, 4, 2]$, elle doit retourner 0.

   (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.

```
float f(float t[], unsigned n)
{
  unsigned m;

  if (n == 0)
    return 1.0;

  if (n == 1)
    return *t;

  m = n / 2;

  return f(t, m) * f(t + m, n - m);
}
```

(b) Écrire une fonction réalisant exactement la même opération, mais n'effectuant aucun appel récursif.

4. Pour représenter un vecteur d'entiers dont le nombre d'éléments peut être très variable, on le décompose en blocs d'éléments consécutifs, et on place ces blocs dans une liste liée. En d'autres termes, chaque bloc possède un pointeur vers le bloc suivant, à l'exception du dernier bloc pour lequel ce pointeur est vide. Les blocs apparaissent dans la liste liée dans le même ordre que dans le vecteur. Chaque bloc contient exactement 1000 éléments du vecteur, sauf le dernier qui est plus petit dans le cas où le nombre total d'éléments du vecteur n'est pas un multiple exact de 1000.
   (a) Écrire un fragment de code définissant un type structuré capable de représenter un tel bloc.
   (b) Écrire une fonction prenant en arguments un vecteur d'entiers (fourni sous la forme d'un tableau) et le nombre d'éléments de ce vecteur, supposé non nul, et construisant la représentation de ce vecteur en allouant dynamiquement les blocs qui la composent. La fonction doit retourner un pointeur vers le premier bloc de la représentation obtenue, ou un pointeur vide en cas d'erreur.
   (c) Écrire une fonction prenant en argument un pointeur vers le premier bloc de la représentation d'un vecteur, et retournant la taille de celui-ci (c'est-à-dire le nombre total d'entiers qu'il contient).

Exemples de solutions

1. (a) Il suffit d'écrire une boucle dans laquelle on extrait la valeur du chiffre de droite du nombre, on l'ajoute à un accumulateur, et on retire ce chiffre en divisant le nombre par 10. On répète alors cette opération tant que le nombre obtenu n'est pas nul. On obtient le code suivant.

```
unsigned somme_chiffres(unsigned n)
{
  unsigned s;

  for (s = 0; n; n /= 10)
    s += n % 10;

  return s;
}
```

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

$$\{\texttt{n} = \texttt{n}_0\}$$
```
for (s = 0; n; n /= 10)
  s += n % 10;
```
$$\{\texttt{s} = \text{somme des chiffres de } \texttt{n}_0\}$$

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

$$\{\texttt{n} = \texttt{n}_0,\ \texttt{s} = 0\}$$
```
while (n)
{
  s += n % 10;
  n /= 10;
}
```
$$\{\texttt{s} = \text{somme des chiffres de } \texttt{n}_0\}$$

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

$$I : \texttt{n}_0 \geq 0 \text{ et } (\exists k, r \in \mathbb{N} : \texttt{n}_0 = 10^k \texttt{n} + r \text{ et } 0 \leq r < 10^k \text{ et } \texttt{s} = \text{somme des chiffres de } r)$$

Cet invariant exprime qu'avant et après chaque itération de la boucle, la valeur de `n` correspond au quotient entier de la valeur initiale $\texttt{n}_0$ de cette variable par une certaine puissance de 10. Cela revient à dire que l'écriture décimale de `n` est un préfixe de celle de $\texttt{n}_0$. En outre, `s` contient la somme des chiffres qui figurent dans $\texttt{n}_0$ et pas dans `n`.

Montrons maintenant que cet invariant est valide.

— Initialement, on a $\texttt{n}_0 \geq 0$, $\texttt{n} = \texttt{n}_0$ et $\texttt{s} = 0$. Les valeurs $k = 0$ et $r = 0$ permettent donc à l'invariant d'être satisfait.

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

$$\{I,\ \texttt{n} \neq 0\}$$
```
s += n % 10;
n /= 10;
```
$$\{I\}$$

Montrons que ce triplet est valide, en notant respectivement $\mathtt{x}$ et $\mathtt{x}'$ la valeur d'une variable $\mathtt{x}$ avant et après l'itération concernée.
On a $\mathtt{n}'_0 = \mathtt{n}_0$ et $\mathtt{s}' = \mathtt{s} + d$, où d est le reste de la division de $\mathtt{n}$ par 10, c'est-à-dire le dernier chiffre de l'écriture décimale de $\mathtt{n}$. On a également

$$\mathtt{n}' = \frac{\mathtt{n} - d}{10},$$

c'est-à-dire $\mathtt{n} = 10\mathtt{n}' + d$.
Avant l'itération, l'invariant implique

$$\mathtt{n}_0 = 10^k\mathtt{n} + r$$

pour certains $k, r \in \mathbb{N}$ tels que $0 \leq r < 10^k$. En remplaçant la valeur de $\mathtt{n}$, on obtient

$$\begin{aligned} \mathtt{n}_0 &= 10^k(10\mathtt{n}' + d) + r \\ &= 10^{k+1}\mathtt{n}' + (10^k d + r). \end{aligned}$$

En posant $k' = k + 1$ et $r' = 10^k d + r$, on a donc

$$\mathtt{n}'_0 = 10^{k'}\mathtt{n}' + r'.$$

En outre, on a bien $0 \leq r' < 10^{k'}$, et la somme des chiffres de r' est égale à celle de r plus d. L'invariant est donc satisfait à l'issue de l'itération, et le triplet est valide.

— En fin de boucle, on a $\{I, \mathtt{n} = 0\}$, ce qui implique que la valeur de $\mathtt{s}$ est égale à la somme des chiffres de $\mathtt{n}_0$. En effet, si $\mathtt{n} = 0$, alors $\mathtt{n}_0 = 10^k\mathtt{n} + r$ entraîne $r = \mathtt{n}_0$.

2. (a)
```
unsigned plus_grand_consecutifs(unsigned t[], unsigned n)
{
  unsigned g, i;

  if (n < 2)
    return 0;

  g = 0;

  for (i = 1; i < n; i++)
    if (t[i] > g && t[i] == t[i-1])
      g = t[i];

  return g;
}
```

(b) Cette fonction se contente de parcourir une seule fois le tableau `n` ; sa complexité en temps est donc $O(\texttt{n})$. Elle utilise un espace mémoire constitué du tableau `t`, de taille `n`, et d'un nombre borné de variables, donc sa complexité en espace est également $O(\texttt{n})$.

3. (a) Cette fonction calcule le produit des éléments du tableau `t`, contenant `n` nombres réels.

   (b)
```
float f(float t[], unsigned n)
{
  float p;
  unsigned i;

  for (i = 0, p = 1.0; i < n; i++)
    p *= t[i];

  return p;
}
```

4. (a)
```
struct bloc
{
  int elements[1000];
  unsigned short nb;    // Nombre d'éléments dans le bloc
  struct bloc *suivant;
};
```

   (b) Le plus simple consiste à allouer les éléments de la liste liée dans l'ordre inverse de leur position, c'est-à-dire en commençant par le dernier et en continuant jusqu'au début de la liste.

   La seule difficulté est alors de calculer la taille du dernier bloc composant la liste. Si le vecteur comprend n éléments, alors cette taille doit être égale à 1000 si n est divisible par 1000, et au reste de la division de n par 1000 sinon.

   On obtient alors le code suivant. Notons qu'en cas d'erreur d'allocation de mémoire, la partie de la liste liée déjà construite doit être libérée. Étant donné que cette opération de libération peut être utile dans d'autres contextes, on la place dans une procédure séparée.

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

void liberer_vecteur(struct bloc *b)
{
  struct bloc *b_suivant;

  while (b)
    {
      b_suivant = b -> suivant;
      free(b);
      b = b_suivant;
    }
}

struct bloc *allouer_vecteur(int v[], unsigned long n)
{
  struct bloc *b, *liste;

  for (liste = NULL; n;)
    {
      b = malloc(sizeof(struct bloc));
      if (!b)
        {
          liberer_vecteur(liste);
          return NULL;
        }

      b -> nb = (n - 1) % 1000 + 1;
      memcpy(b -> elements, v + n - b -> nb,
             b -> nb * sizeof(int));
      b -> suivant = liste;
      liste = b;
      n -= b -> nb;
    }

  return liste;
}
```

(c)
```
unsigned long taille_vecteur(struct bloc *b)
{
  unsigned long taille;

  for (taille = 0; b; b = b -> suivant)
    taille += b -> nb;

  return taille;
}
```