Cours d'introduction à l'informatique
Examen de juin 2026
Énoncés et solutions

Énoncés

1. (a) Écrire une fonction prenant en arguments deux entiers strictement positifs, et retournant le nombre de chiffres identiques situés à la même position dans leur écriture décimale, en considérant les positions de la droite vers la gauche. Par exemple, pour les nombres 2026 et 1206, cette fonction doit retourner 1, car seul le chiffre des unités 6 leur est commun. Pour les nombres 1234444 et 274444, la fonction doit retourner 5.

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

2. (a) Écrire une fonction prenant en arguments deux chaînes de caractères `s1` et `s2`, et qui retourne :

   - -1 si `s1` apparaît avant `s2` dans l'ordre du dictionnaire,
   - 0 si les deux chaînes sont identiques,
   - $+1$ si `s1` apparaît après `s2`.

   On suppose que `s1` et `s2` sont non vides et ne contiennent que des lettres minuscules encodées en ASCII.

   Par exemple pour `s1` = `"introduction"` et `s2` = `"informatique"`, la fonction doit retourner $+1$, puisque le premier mot figure après le second dans l'ordre du dictionnaire.

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

3. (a) Décrire le plus simplement possible ce que calcule la fonction C suivante :

```
unsigned f(int t1[], int t2[], unsigned n)
{
  if (n-- && t1[n] == t2[n])
    return f(t1, t2, n) + 1;
```

```
  else
    return 0;
}
```

(b) Quelle est la complexité en espace de la fonction du point (a), en ne tenant pas compte de la mémoire occupée par les données d'entrée ?

(c) Écrire une fonction C réalisant exactement la même opération que la fonction du point (a), mais sans effectuer d'appel récursif.

4. Un vecteur creux est un vecteur contenant beaucoup d'éléments nuls. Afin d'économiser de l'espace, on souhaite représenter un tel vecteur en ne mémorisant que ses éléments non nuls. Plus précisément la représentation d'un vecteur creux $\vec{v}$ est une structure de données composée de

   — la dimension n de $\vec{v}$,
   — le nombre m d'éléments non nuls de $\vec{v}$,
   — un tableau t de taille m contenant ces éléments. Chacun d'entre eux est caractérisé par une paire (i, r) formée par sa position $i \in [0, n-1]$ dans $\vec{v}$ et sa valeur $r \in \mathbb{R}$. Les éléments de $\vec{v}$ apparaissent dans t par ordre strictement croissant de position.

   Par exemple, pour le vecteur $\vec{v} = [0, 0, (3{,}5), 0, 0, 0, (-8{,}3)]$, on a $n = 7$, $m = 2$ et $t = [(2, (3{,}5)), (6, (-8{,}3))]$.

   (a) Écrire un fragment de code définissant un type structuré permettant de représenter un vecteur creux.

   (b) Écrire une fonction prenant en arguments un tableau de réels et sa taille, et retournant une représentation nouvellement allouée de ce tableau sous la forme d'un vecteur creux.

   (c) Écrire une fonction prenant en argument un pointeur vers la représentation d'un vecteur creux, et retournant la somme de tous ses éléments. Par exemple, pour le vecteur précédemment donné en illustration, cette fonction doit retourner $-4{,}8$.

   Note : Il est permis de définir des types ou des fonctions supplémentaires si votre solution le nécessite.

Exemples de solutions

1. (a)
```
unsigned  nb_ch_identiques(unsigned n1, unsigned n2)
{
  unsigned  nb;

  for (nb = 0; n1 && n2; n1 /= 10, n2 /= 10)
    if (n1 % 10 == n2 % 10)
      nb++;

  return nb;
}
```

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

$$\{\texttt{n1} = \texttt{n1}_0 > 0, \texttt{n2} = \texttt{n2}_0 > 0\}$$

```
for (nb = 0; n1 && n2; n1 /= 10, n2 /= 10)
  if (n1 % 10 == n2 % 10)
    nb++;
```

$\{\texttt{nb} =$ nombre de chiffres identiques à la même position dans $\texttt{n1}_0$ et $\texttt{n2}_0\}$

Note : Étant donné que les variables `n1` et `n2` sont modifiées par la boucle, on a introduit deux variables auxiliaires $\texttt{n1}_0$ et $\texttt{n2}_0$ représentant leur valeur initiale, afin de pouvoir y faire référence dans la postcondition.

En décomposant la boucle `for`, ce triplet devient :

$$\{\texttt{n1} = \texttt{n1}_0 > 0, \texttt{n2} = \texttt{n2}_0 > 0, \texttt{nb} = 0\}$$

```
while (n1 && n2)
  {
    if (n1 % 10 == n2 % 10)
      nb++;

    n1 /= 10;
    n2 /= 10;
  }
```

$\{\texttt{nb} =$ nombre de chiffres identiques à la même position dans $\texttt{n1}_0$ et $\texttt{n2}_0\}$

Pour obtenir un invariant de boucle I, on caractérise le travail effectué par la boucle jusqu'à l'itération courante. Un invariant possible est

I : $\texttt{n1}_0 > 0$, $\texttt{n2}_0 > 0$,
$\exists i \geq 0,\ 0 \leq r_1, r_2 < 10^i$:
$\texttt{n1}_0 = \texttt{n1} \,.\, 10^i + r_1$, $\texttt{n2}_0 = \texttt{n2} \,.\, 10^i + r_2$,
$\texttt{nb}$ = nb. ch. identiques aux positions de 0 à $i-1$ dans r_1 et r_2,

en considérant que les positions des chiffres sont numérotées de la droite vers la gauche à partir de 0. Intuitivement les contraintes $\texttt{n1}_0 = \texttt{n1} \,.\, 10^i + r_1$ et $\texttt{n2}_0 = \texttt{n2} \,.\, 10^i + r_2$ décomposent respectivement $\texttt{n1}_0$ et $\texttt{n2}_0$ en leurs i chiffres de poids faible (c'est-à-dire, aux positions de 0 à $i-1$) représentés par r_1 et r_2, et leurs autres chiffres par `n1` et `n2`.

Montrons maintenant que cet invariant est valide.

— Initialement, on a $\texttt{n1}_0 > 0$ et $\texttt{n2}_0 > 0$ par la précondition, ainsi que $\texttt{n1} = \texttt{n1}_0$, $\texttt{n2} = \texttt{n2}_0$ et $\texttt{nb} = 0$. L'invariant est donc satisfait, en choisissant $i = 0$, $r_1 = 0$ et $r_2 = 0$.

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

$$\{I,\ \texttt{n1} \neq 0,\ \texttt{n2} \neq 0\}$$

```
if (n1 % 10 == n2 % 10)
  nb++;

n1 /= 10;
n2 /= 10;
```

$$\{I\}$$

Montrons que ce triplet est valide, en notant respectivement `x` et $\texttt{x}'$ la valeur d'une variable `x` avant et après l'exécution du code. Premièrement, on a $\texttt{n1}'_0 = \texttt{n1}_0$ et $\texttt{n2}'_0 = \texttt{n2}_0$ qui donnent $\texttt{n1}'_0 > 0$ et $\texttt{n2}'_0 > 0$. Ensuite, on a

$$\texttt{n1} = 10 \,.\, \texttt{n1}' + (\texttt{n1 \% 10})$$

et

$$\texttt{n2} = 10 \,.\, \texttt{n2}' + (\texttt{n2 \% 10}),$$

car les instuctions `n1 /= 10` et `n2 /= 10` retirent respectivement de `n1` et `n2` leur chiffre de poids faible `n1 % 10` et `n2 % 10`.

Puisque l'invariant figure dans la précondition, il existe $i, r_1, r_2 \geq 0$ tels que $r_1 < 10^i$, $r_2 < 10^i$, $\texttt{n1}_0 = \texttt{n1} \,.\, 10^i + r_1$ et $\texttt{n2}_0 = \texttt{n2} \,.\, 10^i + r_2$. Les égalités précédentes donnent alors

$$\texttt{n1}'_0 = \texttt{n1}' \,.\, 10^{i+1} + (\texttt{n1 \% 10})\, 10^i + r_1$$

et

$$\texttt{n2}'_0 = \texttt{n2}' \,.\, 10^{i+1} + (\texttt{n2 \% 10})\, 10^i + r_2.$$

En fixant $i' = i+1$, $r'_1 = (\texttt{n1 \% 10})\, 10^i + r_1$ et $r'_2 = (\texttt{n2 \% 10})\, 10^i + r_2$, on obtient donc bien

$$\texttt{n1}'_0 = \texttt{n1}' \,.\, 10^{i'} + r'_1$$

et

$$\texttt{n2}'_0 = \texttt{n2}' \,.\, 10^{i'} + r'_2,$$

ce qui revient à avoir ajouté respectivement les chiffres `n1 % 10` et `n2 % 10` au début de l'écriture décimale de r_1 et r_2. En d'autres termes, ces relations expriment que r'_1 est formé du chiffre `n1 % 10` suivi par les chiffres de r_1, et similairement que r'_2 est formé du chiffre `n2 % 10` suivi par les chiffres de r_2. Il y a deux cas à considérer : Si `n1 % 10` et `n2 % 10` sont égaux, alors r'_1 et r'_2 possèdent un chiffre commun de plus que r_1 et r_2, et l'on a bien $\texttt{nb}' = \texttt{nb}+1$. Dans le cas contraire, le nombre de chiffres communs à r'_1, r'_2 et r_1, r_2 est identique, et l'on a $\texttt{nb}' = \texttt{nb}$. Dans tous les cas, l'invariant I est satisfait.

— En sortie de boucle, on a $\{I, (\texttt{n1} = 0 \text{ ou } \texttt{n2} = 0)\}$. Considérons d'abord le cas $\texttt{n1} = 0$. L'invariant I entraîne alors $\texttt{n1}_0 = r_1$ avec $r_1 < 10^i$ pour un certain $i \geq 0$, et $\texttt{n2}_0 = \texttt{n2} \,.\, 10^i + r_2$ avec $r_2 < 10^i$. Cela implique que la position des chiffres communs à $\texttt{n1}_0$ et $\texttt{n2}_0$ est nécessairement inférieure à i. Ces chiffres communs sont donc exactement ceux de r_1 et r_2, et on déduit donc de l'invariant que la valeur de `nb` donne bien le nombre de chiffres identiques situés à la même position dans $\texttt{n1}_0$ et $\texttt{n2}_0$. Le cas $\texttt{n2} = 0$ se traite de façon symétrique, en permutant $\texttt{n1}_0$ avec $\texttt{n2}_0$ ainsi que `n1` avec `n2` et r_1 avec r_2.

2. (a) Il suffit de parcourir les caractères qui composent `s1` et `s2` de la gauche vers la droite, en s'arrêtant à la première différence ou à la fin des chaînes. On obtient le code suivant.

```
int  strcmp(char *s1, char *s2)
{
  for (;; s1++, s2++)
    if (*s1 < *s2)
      return -1;
    else
      if (*s1 > *s2)
        return 1;
      else
        if (!*s1)
          return 0;
}
```

(b) Dans le pire des cas, cette fonction effectue un nombre d'itérations linéaire en la longueur de la plus petite chaîne. Sa complexité en temps vaut donc $O(\min(|\texttt{s1}|, |\texttt{s2}|))$, où $|\texttt{s}|$ designe la longueur d'une chaîne `s`.

3. (a) Cette fonction retourne la longueur du plus grand suffixe commun aux tableaux `t1` et `t2`, tous deux de taille `n`.

(b) Le pire cas est atteint lorsque les tableaux sont égaux, ce qui conduit à une profondeur de récursion égale à `n`. Étant donné que les autres données manipulées par la fonction, à l'exception des données d'entrée, sont de taille bornée, cela conduit à une complexité en espace égale à $O(\texttt{n})$.

(c)
```
unsigned f(int t1[], int t2[], unsigned n)
{
  unsigned i;

  for (i = 0; i < n; i++)
    if (t1[n - 1 - i] != t2[n - 1 - i])
      break;

  return i;
}
```

4. (a)
```
typedef struct
{
  unsigned  position;
  double    valeur;
} el_vecteur_creux;

typedef struct
{
  unsigned  dimension, nb_elements;
  el_vecteur_creux *elements;
} vecteur_creux;
```

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

vecteur_creux *creer_vecteur_creux(double *t, unsigned n)
{
  vecteur_creux *vc;
  unsigned i, j, m;

  vc = malloc(sizeof(vecteur_creux));
  if (!vc)
    return NULL;

  for (i = 0, m = 0; i < n; i++)
    if (t[i] != 0.0)
      m++;

  vc -> dimension = n;
  vc -> nb_elements = m;

  if (m)
    {
      vc -> elements = malloc(m * sizeof(el_vecteur_creux));
      if (!vc -> elements)
        {
          free(vc);
          return NULL;
        }
    }
  else
    vc -> elements = NULL;
```

```
  for (i = 0, j = 0; i < n; i++)
    if (t[i] != 0.0)
      {
        vc -> elements[j].position = i;
        vc -> elements[j].valeur = t[i];
        j++;
      }

  return vc;
}
```

(c)
```
double  somme_elements(vecteur_creux *vc)
{
  unsigned  i;
  double  somme;

  for (i = 0, somme = 0.0; i < vc -> nb_elements; i++)
    somme += vc -> elements[i].valeur;

  return somme;
}
```