Cours d'introduction à l'informatique
Examen de seconde session 2025
Énoncés et solutions

# Énoncés

1. (a) Écrire une fonction prenant en arguments un tableau d'entiers signés `t` ainsi que sa taille strictement positive `n`, et qui retourne l'écart entre les deux valeurs extrêmes du tableau. Par exemple, si le tableau contient $[2, -2, 10, 5, 10]$, alors la fonction doit retourner 12. Si le tableau contient $[5, 5]$ ou $[3]$, elle doit retourner 0.
   (b) Par la méthode des invariants, démontrer que la valeur calculée par cette fonction est correcte.

2. (a) Écrire une procédure qui inverse l'ordre des caractères dans une chaîne qu'elle reçoit en argument. Par exemple, si on fournit à cette procédure la chaîne de caractères `"Informatique"`, celle-ci doit être modifiée en `"euqitamrofnI"`.

   *Note importante :* Pour résoudre ce problème, il n'est pas permis d'utiliser des fonctions issues de la bibliothèque standard. Vous pouvez en revanche définir des fonctions supplémentaires de votre choix.
   (b) Calculer la complexité en temps de la procédure obtenue au point (a).

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

```
int f(int *a, unsigned b)
{
  if (b <= 1)
    return 1;

  return (a[b - 2] <= a[b - 1]) && f(a, b - 1);
}
```

   (b) Quelle est la complexité en espace de la fonction du point (a) ?
   (c) Écrire une fonction C réalisant exactement la même opération que la fonction du point (a), mais sans effectuer d'appel récursif.

4. Un navigateur WWW enregistre l'historique d'une session en mémorisant les pages Web qui ont été visitées, ainsi que le moment où elles ont été visitées. Chaque page Web est identifiée par une chaîne de caractères appelée URL (*Uniform Resource Locator*), par exemple `"https://www.uliege.be"`. La date et l'heure d'une visite sont représentées par un seul nombre entier de type `unsigned long` correspondant au nombre de secondes écoulées depuis une date de référence fixée au premier janvier 1970 à 0h00. De plus, pour chaque page faisant partie de l'historique, on conserve un moyen d'accès à la page précédente et à la page suivante (par rapport à l'ordre chronologique où elles ont été visitées), de façon à pouvoir naviguer en arrière ou en avant.

   (a) Écrire un fragment de code définissant un type structuré capable de représenter une page Web contenue dans un historique. Ce type structuré doit comprendre
   — un pointeur vers une chaîne de caractères donnant l'URL de la page,
   — la représentation de la date et l'heure de visite de la page,
   — deux pointeurs vers la page précédente et la page suivante dans l'historique. (Si l'une de ces pages n'existe pas, le pointeur concerné est alors vide.)

   (b) Écrire une fonction prenant en arguments un tableau `url` de chaînes de caractères constituant les URLs de pages visitées, dans l'ordre chronologique de leur visite, un tableau `temps` d'entiers contenant les dates et heures de visite de ces pages (en d'autres termes, `temps`$[i]$ contient le moment où la page `url`$[i]$ a été visitée), et la taille `n` commune de ces deux tableaux, supposée non nulle. On demande que la fonction crée un nouvel historique de navigation pour les `n` pages fournies, en allouant dynamiquement la mémoire nécessaire, et retourne un pointeur vers la représentation de la première page faisant partie de cet historique.

   *Note* : Vous pouvez programmer des fonctions ou des types de données supplémentaires si votre solution le nécessite, et utiliser des fonctions issues de la bibliothèque standard.

   (c) Écrire une fonction prenant en argument un pointeur vers n'importe quelle page d'un historique de navigation, et qui retourne le nombre de secondes écoulées entre les moments où la première et la dernière page de cet historique ont été visitées.

# Exemples de solutions

1. (a) L'écart entre les deux valeurs extrêmes du tableau est égal à la différence $v_{max} - v_{min}$ entre la plus grande valeur $v_{max}$ et la plus petite valeur $v_{min}$ qu'il contient. Ces valeurs peuvent être calculées en parcourant une seule fois les éléments du tableau. On obtient le code suivant.

```
int ecart(int t[], unsigned n)
{
  int vmin, vmax;
  unsigned i;

  for (vmin = vmax = t[0], i = 1; i < n; i++)
    {
      if (t[i] < vmin)
        vmin = t[i];

      if (t[i] > vmax)
        vmax = t[i];
    }

  return vmax - vmin;
}
```

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

$$\{\texttt{n} > 0\}$$

```
for (vmin = vmax = t[0], i = 1; i < n; i++)
  {
    if (t[i] < vmin)
      vmin = t[i];

    if (t[i] > vmax)
      vmax = t[i];
  }
```

$$\{\texttt{vmin} = \text{plus petite valeur dans } \texttt{t}[0 : \texttt{n} - 1],\ \texttt{vmax} = \text{plus grande valeur dans } \texttt{t}[0 : \texttt{n} - 1]\},$$

   où la notation $\texttt{t}[a : b]$ dénote le sous-tableau de `t` formé par les éléments dont l'index est compris entre $a$ et $b$ (inclus).

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

$$\{\texttt{n} > 0,\ \texttt{i} = 1,\ \texttt{vmax} = \texttt{vmin} = \texttt{t}[0]\}$$

```
while(i < n)
  {
    if (t[i] < vmin)
      vmin = t[i];

    if (t[i] > vmax)
      vmax = t[i];

    i++;
  }
```

$$\{\texttt{vmin} = \text{plus petite valeur dans } \texttt{t}[0 : \texttt{n} - 1],\ \texttt{vmax} = \text{plus grande valeur dans } \texttt{t}[0 : \texttt{n} - 1]\},$$

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

$$I : 1 \leq \texttt{i} \leq \texttt{n},\ \texttt{vmin} = \text{plus petite valeur dans } \texttt{t}[0 : \texttt{i} - 1],\ \texttt{vmax} = \text{plus grande valeur dans } \texttt{t}[0 : \texttt{i} - 1].$$

Montrons maintenant que cet invariant est valide.

— Initialement, on a $\texttt{n} \geq 1$, $\texttt{i} = 1$ et $\texttt{vmax} = \texttt{vmin} = \texttt{t}[0]$. Étant donné que le sous-tableau $\texttt{t}[0 : \texttt{i} - 1]$ ne contient que l'élément $\texttt{t}[0]$, l'invariant est satisfait.

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

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

```
if (t[i] < vmin)
  vmin = t[i];
if (t[i] > vmax)
  vmax = t[i];
i++;
```

$$\{I\}$$

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'exécution du code. Premièrement, on a $\mathtt{i}' = \mathtt{i} + 1$, qui combiné avec la précondition $1 \leq \mathtt{i} < \mathtt{n}$ et $\mathtt{n}' = \mathtt{n}$ entraîne $1 \leq \mathtt{i}' \leq \mathtt{n}'$.

Ensuite, si $\mathtt{t}[\mathtt{i}] < \mathtt{vmin}$, alors $\mathtt{vmin}' = \mathtt{t}[\mathtt{i}]$, sinon $\mathtt{vmin}' = \mathtt{vmin}$. Dans les deux cas, $\mathtt{vmin}'$ est bien égale à la plus petite valeur contenue dans $\mathtt{t}[0 : \mathtt{i}' - 1]$. De même, si $\mathtt{t}[\mathtt{i}] > \mathtt{vmax}$, alors $\mathtt{vmax}' = \mathtt{t}[\mathtt{i}]$, sinon $\mathtt{vmax}' = \mathtt{vmax}$. Dans les deux cas, $\mathtt{vmax}'$ est bien égale à la plus grande valeur contenue dans $\mathtt{t}[0 : \mathtt{i}' - 1]$. La postcondition est donc satisfaite.

— En sortie de boucle, on a $\{I, \mathtt{i} \geq \mathtt{n}\}$, qui implique $\mathtt{i} = \mathtt{n}$. Il découle alors de l'invariant que $\mathtt{vmin}$ et $\mathtt{vmax}$ sont respectivement égales à la plus petite et la plus grande valeur contenue dans $\mathtt{t}[0 : \mathtt{n} - 1]$, ce qui correspond bien à la postcondition.

2. (a) Pour résoudre ce problème, on peut commencer par mesurer la longueur $n$ de la chaîne de caractères reçue en argument, et ensuite effectuer une boucle chargée de permuter pour $i = 0, 1, \ldots$ les caractères situés aux positions $i_1 = i$ et $i_2 = (n - 1) - i$, tant que l'on a $i_1 < i_2$. On obtient le code suivant.

```
static unsigned longueur(char s[])
{
  unsigned n;

  for (n = 0; *s; s++)
    n++;

  return n;
}

void inverser(char s[])
{
  unsigned n, i;
  char aux;

  n = longueur(s);
```

```
  for (i = 0; i + i + 1 < n; i++)
    {
      aux = s[i];
      s[i] = s[n - 1 - i];
      s[n - 1 - i] = aux;
    }
}
```

(b) La boucle de la fonction `longueur` effectue un nombre d'itérations égal à la longueur $|\mathtt{s}|$ de la chaîne `s` qu'elle reçoit en argument ; sa complexité en temps vaut donc $O(|\mathtt{s}|)$. Après avoir invoqué une seule fois `longueur`, la fonction `inverser` effectue une boucle dont le nombre d'itérations est inférieur à $|\mathtt{s}|$, chacune d'entre elles nécessitant un temps constant. Au total, la complexité en temps de la fonction `inverser` vaut donc $O(|\mathtt{s}| + |\mathtt{s}|) = O(|\mathtt{s}|)$.

3. (a) Cette fonction retourne une valeur booléenne qui indique si les `b` éléments du tableau d'entiers `a` sont triés par ordre croissant.

(b) La profondeur de récursion, c'est-à-dire le nombre maximum d'appels à `f` simultanément en cours, est égale à $\min(1, \mathtt{b})$. La complexité en espace de la fonction vaut donc $O(\mathtt{b})$.

(c)
```
int f(int *a, unsigned b)
{
  unsigned i;

  for (i = 1; i < b; i++)
    if (a[i - 1] > a[i])
      return 0;

  return 1;
}
```

4. (a)
```
struct page
{
  char *url;
  unsigned long visite;
  struct page *precedent, *suivant;
};
```

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

struct page *nouvel_historique(char *url[],
                 unsigned long temps[], unsigned n)
{
  struct page *t;
  unsigned i;

  t = malloc(n * sizeof(struct page));
  if (!t)
    return NULL;

  for (i = 0; i < n; i++)
    {
      t[i].url = url[i];
      t[i].visite = temps[i];
      t[i].precedent = i ? (t + i - 1) : NULL;
      t[i].suivant = (i + 1 < n) ? (t + i + 1) : NULL;
    }

  return t;
}
```

(c)
```
unsigned long nb_secondes(struct page *p)
{
  struct page *premier, *dernier;

  for (premier = p; premier -> precedent;
       premier = premier -> precedent);

  for (dernier = p; dernier -> suivant;
       dernier = dernier -> suivant);

  return dernier -> visite - premier -> visite;
}
```