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

Énoncés

1. (a) Écrire une fonction capable de calculer la longueur d'une chaîne de caractères reçue en argument, sans tenir compte des caractères blancs (`' '`) éventuels situés au début de cette chaîne. Par exemple, pour les chaînes `"ZZZZ"`, `"  abcd"` et `" 1 2 "`, cette fonction doit retourner 4. Dans le cas d'une chaîne vide ou ne contenant que des caractères blancs, 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 procédure prenant en arguments un pointeur vers un vecteur `v` d'octets non signés, le nombre `n` d'éléments de ce vecteur, et un pointeur vers un vecteur `b` de 256 entiers. L'opération devant être effectuée par cette procédure consiste, pour chaque $i \in [0, 255]$, à placer dans `b[`i`]` une valeur booléenne vraie si le vecteur `v` contient la valeur i (à n'importe quelle position), et fausse sinon. On demande que l'implémentation de cette fonction soit la plus efficace possible.

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

3. (a) Écrire un fragment de code définissant un type structuré représentant une date, composé de trois champs numériques correspondant à un jour, un mois et une année.

   (b) Écrire une fonction recevant en arguments deux pointeurs vers des dates (représentées grâce à la structure définie au point (a)), et retournant une valeur booléenne indiquant si la première de ces dates est antérieure à la seconde. *Note :* On ne demande pas de vérifier que les dates reçues en entrée sont valides.

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

```
int p(char *s1, char *s2)
{
  if (!*s1)
    return 1;

  if (*s1 != *s2)
    return 0;

  return p(++s1, ++s2);
}
```

(b) Écrire une fonction réalisant exactement la même opération, mais sans effectuer d'appel récursif.

5. Écrire une fonction prenant en arguments deux chaînes de caractères, et retournant un pointeur vers une chaîne nouvellement allouée égale à leur concaténation, c'est-à-dire, au contenu de la première chaîne immédiatement suivi par celui de la seconde. Par exemple, pour les chaînes `"infor"` et `"matique"`, le résultat doit contenir `"informatique"`.

   Notes :
   — Vous pouvez programmer des fonctions ou des types de données supplémentaires si votre solution le nécessite.
   — Il est permis d'employer des fonctions de la bibliothèque standard, en particulier la fonction `strlen(str)` qui retourne la longueur de la chaîne de caractères pointée par `str`. Le fichier d'en-tête correspondant est `string.h`.

Exemples de solutions

1. (a) Il suffit d'écrire une boucle qui parcourt tous les caractères qui composent la chaîne, en mettant à jour un compteur initialisé à zéro. Quand ce compteur vaut zéro, il doit être incrémenté pour chaque caractère non blanc rencontré. Quand sa valeur est non nulle, il doit être incrémenté pour chaque caractère. En d'autres termes, à chaque itération de la boucle, le compteur ne doit être incrémenté que s'il est différent de zéro ou si le caractère courant n'est pas blanc. On obtient le code suivant.

```
unsigned longueur_hors_blancs(char *s)
{
  unsigned c;

  for (c = 0; *s; s++)
    if (c || *s != ' ')
      c++;

  return c;
}
```

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

$$\{\mathtt{s} = \mathtt{s}_0\}$$
```
for (c = 0; *s; s++)
  if (c || *s != ' ')
    c++;
```
$$\{\mathtt{c} = \text{longueur de } \mathtt{s}_0 \text{ hors caractères blancs initiaux}\}$$

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

$$\{\mathtt{s} = \mathtt{s}_0,\ \mathtt{c} = 0\}$$
```
while (*s)
  {
    if (c || *s != ' ')
      c++;
    s++;
  }
```
$$\{\mathtt{c} = \text{longueur de } \mathtt{s}_0 \text{ hors caractères blancs initiaux}\}$$

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 : $\exists k \geq 0 : \mathtt{s} = \mathtt{s}_0 + k$,
$\mathtt{c}$ = longueur de $\mathtt{s}_0[0 : k-1]$ hors caractères blancs initiaux
et $\mathtt{s}_0[0 : k-1]$ ne contient aucun caractère terminateur

où $\mathtt{s}_0[0 : k-1]$ désigne le préfixe de longueur k de la chaîne $\mathtt{s}_0$.

Cet invariant exprime qu'avant et après chaque itération de la boucle, la valeur de `c` correspond à la longueur du préfixe de la chaîne initiale $\mathtt{s}_0$ situé à gauche du pointeur `s`, en ne tenant pas compte des caractères blancs situés au début de cette chaîne.

Montrons maintenant que cet invariant est valide.

— Initialement, on a $\mathtt{s}_0 = \mathtt{s}$ et $\mathtt{c} = 0$. L'invariant est donc satisfait, puisque la longueur de la chaîne vide est nulle.

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

$$\{I, *\mathtt{s} \neq 0\}$$
```
if (c || *s != ' ')
  c++;
s++;
```
$$\{I\}$$

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.

Notons $\mathtt{s}_1$ le préfixe de la chaîne $\mathtt{s}_0$ situé à gauche du pointeur `s`, et $\mathtt{s}'_1$ le préfixe de la chaîne $\mathtt{s}_0$ situé à gauche du pointeur $\mathtt{s}'$. Étant donné que l'on a $\mathtt{s}' = \mathtt{s} + 1$ et que le caractère $*\mathtt{s}$ n'est pas le caractère terminateur comme l'impose la précondition, la chaîne $\mathtt{s}'_1$ est égale à la concaténation de la chaîne $\mathtt{s}_1$ et du caractère $*\mathtt{s}$.

Il y a trois cas à considérer :

— Si $\mathtt{c} \neq 0$, alors, l'invariant I implique que la chaîne $\mathtt{s}_1$ contient au moins un caractère non blanc, ou un caractère blanc non initial. Cette propriété est donc également vraie pour la chaîne $\mathtt{s}'_1$. On a alors $\mathtt{c}' = \mathtt{c} + 1$, ce qui satisfait la postcondition.

— Si $*\mathtt{s} \neq$ `' '`, alors la longueur de la chaîne $\mathtt{s}'_1$ est supérieure d'une unité à celle de la chaîne $\mathtt{s}_1$, sans tenir compte des caractères blancs initiaux. On a bien dans ce cas $\mathtt{c}' = \mathtt{c} + 1$, ce qui satisfait la postcondition.

— Si aucune des deux conditions précédentes n'est satisfaite, alors la longueur de la chaîne $\mathtt{s}'_1$ est identique à à celle de la chaîne $\mathtt{s}_1$, sans tenir compte des caractères blancs initiaux. On a dans ce cas $\mathtt{c}' = \mathtt{c}$, ce qui satisfait la postcondition.

— En fin de boucle, on a $\{I, *\mathtt{s} = 0\}$, qui implique que `c` est égal à la longueur de la chaîne $\mathtt{s}_0$, sans tenir compte des caractères blancs initiaux.

Il reste à démontrer que la boucle se termine. Il suffit pour cela de considérer le variant de boucle $\ell_0 - k$, où ℓ_0 est la longueur de la chaîne de caractères $\mathtt{s}_0$, et $k \geq 0$ est tel que $\mathtt{s} = \mathtt{s}_0 + k$, comme exprimé par l'invariant de boucle I. Étant donné que l'invariant implique que $\mathtt{s}_0[0 : k-1]$ ne contient pas de caractère terminateur, on a $k \leq \ell_0$, ce qui entraîne que le variant possède toujours une valeur entière non négative. De plus, chaque itération de la boucle incrémente k, et diminue donc la valeur du variant.

2. (a) Il suffit de parcourir une seule fois le vecteur `v`, en mettant à jour pour chaque valeur rencontrée l'élément de `b` qui lui correspond. Initialement, tous les éléments de `b` doivent être faux. On obtient le code suivant.

```
void p(unsigned char *v, unsigned n, int *b)
{
  unsigned i;

  for (i = 0; i < 256; i++)
    b[i] = 0;

  for (i = 0; i < n; i++)
    b[v[i]] = 1;
}
```

   (b) La première boucle effectue toujours 256 itérations et présente donc une complexité en temps égale à $O(1)$. Celle de la seconde boucle est égale à $O(\mathtt{n})$. Au total, la procédure possède donc une complexité en temps qui est $O(\mathtt{n})$.

3. (a)
```
struct date
{
  unsigned char jour, mois;
  int annee;
};
```

   (b)
```
int anterieure(struct date *d1, struct date *d2)
{
  if (d1 -> annee < d2 -> annee)
    return 1;

  if (d1 -> annee > d2 -> annee)
```

```
    return 0;

  if (d1 -> mois < d2 -> mois)
    return 1;

  if (d1 -> mois > d2 -> mois)
    return 0;

  if (d1 -> jour < d2 -> jour)
    return 1;

  return 0;
}
```

4. (a) Cette fonction détermine si la chaîne de caractères pointée par `s1` est un préfixe de celle pointée par `s2`.

   (b)
```
int p(char *s1, char *s2)
{
  for (; *s1; s1++, s2++)
    if (*s1 != *s2)
      return 0;

  return 1;
}
```

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

char *concatenation(char *s1, char *s2)
{
  char *s, *p;
  unsigned len1, len2;

  len1 = strlen(s1);
  len2 = strlen(s2);

  s = malloc(len1 + len2 + 1);
  if (!s)
    return NULL;
```

```
  for (p = s; *s1; s1++)
    *p++ = *s1;

  for (; *s2; s2++)
    *p++ = *s2;

  *p = '\0';

  return s;
}
```

*N*ote : Au lieu d'employer des boucles pour recopier les caractères, on aurait pu utiliser la fonction `memcpy` de la bibliothèque standard.