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

Énoncés

1. (a) Écrire une fonction prenant en argument un nombre entier non signé et non nul, et retournant le nombre de zéros situés à la fin de son écriture décimale. Par exemple, pour le nombre 42000, cette fonction doit retourner 3. Pour le nombre 2022, elle doit retourner 0.

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

2. Écrire une procédure prenant en arguments un tableau `t` de chaînes de caractères, un tableau `b` d'octets, et la taille `n` (commune) de ces deux tableaux. L'objectif de cette procédure consiste à écrire dans `b`$[i]$, pour chaque indice i, une valeur booléenne qui est vraie si la chaîne contenue dans `t`$[i]$ est vide, et fausse sinon.

3. (a) Calculer les complexités en temps et en espace de la fonction suivante, en expliquant votre raisonnement.

```
unsigned f(unsigned n)
{
  if (n <= 1)
    return n;

  return f(n / 2) + f(n % 2);
}
```

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

4. Un traitement de textes représente une ligne de texte sous la forme d'une liste simplement liée de mots. Chaque élément de cette liste contient un pointeur vers une chaîne de caractères représentant un mot, ainsi qu'un pointeur vers l'élément suivant (qui est vide dans le cas du dernier élément).

On considère qu'une ligne de texte est formée par la concaténation de tous les mots qui la composent, chaque paire de mots consécutifs étant séparés par un espace.

Par exemple, la ligne de texte composée des mots `"introduction"`, `"a"`, `"l"` et `"informatique"` est `"introduction a l informatique"`.

(a) Écrire un fragment de code définissant un type structuré pour un élément d'une liste liée représentant une ligne de texte.

(b) Écrire une fonction qui prend en argument un pointeur vers le premier élément d'une liste liée représentant une ligne de texte, et qui retourne la longueur de cette ligne, c'est-à-dire son nombre total de caractères.

(c) Écrire une fonction qui prend en argument un pointeur vers le premier élément d'une liste liée représentant une ligne de texte, et qui retourne une chaîne de caractères nouvellement allouée contenant cette ligne de texte, ou un pointeur vide en cas d'erreur.

Notes : Pour résoudre ces problèmes, il est permis d'exploiter les fonctions de la bibliothèque standard vues au cours. Il est évidemment aussi permis d'utiliser au point (c) la fonction obtenue au point (b).

Exemples de solutions

1. (a) Il suffit d'écrire une boucle dans laquelle on divise le nombre par 10 tant que cela est possible, en incrémentant un compteur à chaque itération. On obtient le code suivant.

```
unsigned nombre_zeros(unsigned n)
{
  unsigned nb = 0;

  while (!(n % 10))
    {
      n /= 10;
      nb++;
    }

  return nb;
}
```

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

$$\{\mathtt{n} = \mathtt{n}_0 > 0,\ \mathtt{nb} = 0\}$$

```
while (!(n % 10))
  {
    n /= 10;
    nb++;
  }
```

$$\{\mathtt{nb} = \text{nombre de zéros à la fin de l'écriture de } \mathtt{n}_0\}$$

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

$$I : \mathtt{n}_0 > 0 \text{ et } \mathtt{nb} > 0 \text{ et } \mathtt{n}_0 = \mathtt{n}\, 10^{\mathtt{nb}}$$

Cet invariant exprime qu'avant et après chaque itération de la boucle, la valeur de `nb` correspond au nombre de facteurs 10 qui ont été extraits de la valeur initiale de `n`.

Montrons maintenant que cet invariant est valide.

— Initialement, on a $\mathtt{n} = \mathtt{n}_0 > 0$ et $\mathtt{nb} = 0$, ce qui satisfait l'invariant.

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

$$\{I,\ \mathtt{n} \text{ est divisible par } 10\}$$

```
n /= 10;
nb++;
```

$$\{I\}$$

Montrons que ce triplet est valide, en notant respectivement $\mathtt{x}$ et $\mathtt{x}'$ la valeur d'une variable `x` avant et après l'itération concernée.
On a $\mathtt{n}'_0 = \mathtt{n}_0$, $\mathtt{n}' = \mathtt{n}/10$ et $\mathtt{nb}' = \mathtt{nb} + 1$. L'égalité $\mathtt{n}_0 = \mathtt{n}\, 10^{\mathtt{nb}}$ se réécrit alors

$$\begin{aligned} \mathtt{n}'_0 &= 10\, \mathtt{n}'\, 10^{\mathtt{nb}'-1} \\ &= \mathtt{n}'\, 10^{\mathtt{nb}'} \end{aligned}$$

ce qui prouve la validité du triplet.

— En fin de boucle, on a $\{I$, `n` n'est pas divisible par $10\}$. On a donc $\texttt{n}_0 = \texttt{n}\,10^{\texttt{nb}}$ avec `n` non divisible par 10. Cela signifie que `nb` est la plus grande puissance de 10 qui divise $\texttt{n}_0$, en d'autres termes que `nb` est égal au nombre de zéros situés à la fin de l'écriture de $\texttt{n}_0$.

2. Il suffit de tester, pour chaque chaîne de caractères contenue dans `t`, si son premier caractère est égal ou non au caractère terminateur. On obtient le code suivant :

```
void test_vide(char *t[], unsigned char b[], unsigned n)
{
  unsigned i;

  for (i = 0; i < n; i++)
    b[i] = t[i][0] == '\0';
}
```

3. (a) L'évaluation de `f(n % 2)` appelle la fonction `f` avec un argument égal à 0 ou 1. Cet appel s'exécute en temps constant et retourne une valeur égale au reste de la division de `n` par 2.

   L'évaluation de `f(n / 2)` appelle récursivement la fonction `f`, en divisant l'argument par 2 à chaque appel jusqu'à atteindre une valeur inférieure ou égale à 1. La profondeur de récursion et le nombre d'appels récursifs sont donc tous les deux $O(\log \texttt{n})$.

   En résumé, on a donc des complexités en temps et en espace qui sont toutes les deux $O(\log \texttt{n})$.

   (b)
```
unsigned f(unsigned n)
{
  unsigned r;

  for (r = 0; n; n /= 2)
    r += n % 2;

  return r;
}
```

4. (a)
```
struct element_ligne
{
  char *mot;
  struct element_ligne *suivant;
};
```

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

unsigned longueur_ligne(struct element_ligne *e)
{
  unsigned l;

  for (l = 0; e; e = e -> suivant)
    {
      l += strlen(e -> mot);
      if (e -> suivant)
        l++;
    }

  return l;
}
```

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

char *construire_chaine(struct element_ligne *e)
{
  char     *s, *p;
  unsigned  l_mot;

  s = malloc(longueur_ligne(e) + 1);
  if (!s)
    return NULL;

  for (p = s; e; e = e -> suivant)
    {
      l_mot = strlen(e -> mot);
      memcpy(p, e -> mot, l_mot);
      p[l_mot] = e -> suivant ? ' ' : '\0';
      p += l_mot + 1;
    }

  return s;
}
```