Cours d'introduction à l'informatique
Examen de janvier 2021
Énoncés et solutions

Énoncés

1. (a) Écrire en langage C une fonction prenant en argument un nombre entier strictement positif n, et retournant le nombre de diviseurs de n. (Par exemple, cette fonction doit retourner 1 pour $n = 1$, 2 pour $n = 2$, et 8 pour $n = 24$.) On demande que l'implémentation de cette fonction soit raisonnablement efficace.

   (b) Déterminer les complexités en temps et en espace de la fonction obtenue au point (a).

   (c) Par la méthode des invariants, démontrer que la valeur retournée par la fonction obtenue au point (a) est correcte. Démontrer également que cette fonction se termine.

2. Expliquer le plus simplement possible (une phrase suffit) l'opération réalisée par cette fonction C :

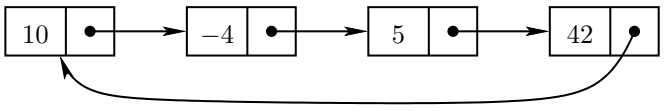
```
int f(char *s)
{
  int r;

  for (r = 0; *s; s++)
    if (*s >= '0' && *s <= '9')
      r = (r + 1) % 2;

  return r;
}
```

3. Un *tampon circulaire* est une structure de données composée d'un certain nombre $k > 0$ d'éléments ordonnés e_1, e_2, $\ldots e_k$, telle que chaque élément e_i contient

   — une valeur (qu'on suppose être un nombre entier par souci de simplicité), et

   — un pointeur vers l'élément e_{i+1} si $i < k$, ou vers e_1 si $i = k$.

Un exemple de tampon circulaire contenant les valeurs 10, −4, 5 et 42 est représenté ci-dessous :



En langage C :

(a) Définir un type structuré permettant de représenter un élément d'un tampon circulaire.

(b) Écrire une fonction prenant en arguments un tableau d'entiers et sa taille (supposée non nulle), et construisant un tampon circulaire contenant ces entiers, dans le même ordre que dans le tableau. Chaque élément de cette structure doit être alloué dynamiquement en mémoire. La fonction doit retourner un pointeur vers le premier élément de la structure de données nouvellement créée.

(c) Écrire une fonction qui calcule la somme de toutes les valeurs contenues dans un tampon circulaire, donné par un pointeur vers son premier élément.

(d) Écrire une fonction qui libère tous les éléments d'un tampon circulaire, donné par un pointeur vers son premier élément.

Exemples de solutions

1. (a) Il suffit d'énumérer tous les diviseurs potentiels d de n dans l'intervalle $[1, \sqrt{n}]$. Pour chaque valeur de d qui divise n, on compte les deux diviseurs d et n/d, ou un seul dans le cas particulier où ces deux valeurs sont égales. On obtient le code suivant.

```
unsigned nb_diviseurs(unsigned n)
{
  unsigned nb, d;

  for (d = 1, nb = 0; d * d <= n; d++)
    {
```

```
      if (n % d)
        continue;

      if (d == n / d)
        nb++;
      else
        nb += 2;
    }

  return nb;
}
```

(b) Pour une valeur donnée de n, le nombre d'itérations de la boucle est borné par $\sqrt{n}$. La complexité en temps de la fonction vaut donc $O(\sqrt{n})$. La quantité de mémoire consommée par la fonction est une constante indépendante de n. Sa complexité en espace vaut donc $O(1)$.

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

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

```
for (d = 1, nb = 0; d * d <= n; d++)
  {
    if (n % d)
      continue;

    if (d == n / d)
      nb++;
    else
      nb += 2;
  }
```

$\{\texttt{nb} = \text{nombre de diviseurs de } \texttt{n}\}$

Ce triplet est équivalent à :

$\{\texttt{n} > 0, \texttt{d} = 1, \texttt{nb} = 0\}$

```
while (d * d <= n)
  {
    if (!(n % d))
      {
        if (d == n / d)
          nb++;
```

```
      else
        nb += 2;
    }
  d++;
}
```
{nb = nombre de diviseurs de n}

Pour trouver un invariant de boucle I, on exprime le traitement effectué par la boucle jusqu'à une itération donnée. Un invariant possible est le suivant :

$$I : \texttt{n} > 0 \text{ et } \texttt{d} \leq \sqrt{\texttt{n}} + 1 \text{ et } \texttt{nb} = \text{nombre de diviseurs } \delta \text{ de } \texttt{n} \text{ tels que } \delta < \texttt{d} \text{ ou } \delta > \frac{\texttt{n}}{\texttt{d}}.$$

Montrons que cet invariant est valide.

— Initialement, on a $\{\texttt{n} > 0, \texttt{d} = 1, \texttt{nb} = 0\} \Rightarrow I$.

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

$\{I, \texttt{d} \leq \sqrt{\texttt{n}}\}$
```
if (!(n % d))
  {
    if (d == n / d)
      nb++;
    else
      nb += 2;
  }
d++;
```
$\{I\}$

Montrons que ce triplet est valide, en notant respectivement x et x′ la valeur d'une variable x avant et après l'itération concernée.

— Si d ne divise pas n, alors on a nb′ = nb et d′ = d + 1. Ni d, ni n/d ne sont des diviseurs de n, donc la postcondition est satisfaite.

— Si d divise n et d = n/d, alors on a nb′ = nb + 1 et d′ = d + 1. Le nombre d = n/d est un diviseur de n tel que d < d′, donc la postcondition est satisfaite.

— Si d divise n et d ≠ n/d, alors on a nb′ = nb + 2 et d′ = d + 1. Les nombres d et n/d sont deux diviseurs distincts de n tels que d < d′ et n/d > n/d′, donc la postcondition est satisfaite.

— En fin de boucle, on a $\{I, \mathtt{d} > \sqrt{\mathtt{n}}\} \Rightarrow \{\mathtt{nb} =$ nombre de diviseurs de $\mathtt{n}\}$. En effet, si $\mathtt{d} > \sqrt{\mathtt{n}}$, alors tous les diviseurs δ de $\mathtt{n}$ satisfont l'une des conditions $\delta < \mathtt{d}$ ou $\delta > \mathtt{n}/\mathtt{d}$.

Il reste à montrer que la boucle se termine toujours. Il suffit de considérer le variant

$$\left\lfloor \sqrt{\mathtt{n}} \right\rfloor - \mathtt{d} + 1,$$

où $\lfloor x \rfloor$ dénote le plus grand entier inférieur ou égal à x. Cette expression possède en effet une valeur entière non-négative (comme l'indique l'invariant I), qui décroît à chaque itération de la boucle. La terminaison de cette boucle est donc toujours garantie.

2. Cette fonction retourne une valeur booléenne qui indique si le nombre de chiffres contenus dans la chaîne de caractères pointée par `s` est impair.

3. (a)
```
struct element
{
  int valeur;
  struct element *suivant;
};
```

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

struct element *construire_tampon(int t[], unsigned nb)
{
  unsigned i;
  struct element *e, *e_courant, *e_dernier;

  for (i = 0; i < nb; i++)
    {
      e = malloc(sizeof(struct element));
      if (!e)
        return NULL;

      e -> valeur = t[nb - 1 - i];

      if (i)
        e -> suivant = e_courant;
      else
        e_dernier = e;
```

```
        e_courant= e;
      }

    if (i)
      e_dernier -> suivant = e;

    return e;
  }
```

(c)
```
  int somme_tampon(struct element *e)
  {
    struct element *e_courant;
    int somme;

    for (e_courant = e, somme = 0;;)
      {
        somme += e_courant -> valeur;
        e_courant = e_courant -> suivant;
        if (e_courant == e)
          return somme;
      }
  }
```

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

  void  libere_tampon(struct element *e)
  {
    struct element *e_courant, *e_suivant;

    for (e_courant = e;;)
      {
        e_suivant = e_courant -> suivant;
        free(e_courant);
        if (e_suivant == e)
          return;
        e_courant = e_suivant;
      }
  }
```