TD/TP 4 — corrigés partiels

1 Listing d'un tableau

Écrire une fonction qui prend comme paramètres un pointeur sur un tableau d'entiers et sa longueur et qui affiche chaque élément sur une ligne, précédé par son indice dans le tableau.

Deux principes généraux :
- Une fonction qui doit agir sur un tableau prend deux paramètres : un pointeur sur le premier élément du tableau, et la longueur du tableau.
- Pour exécuter du code sur tous les éléments d'un tableau, on utilise une boucle `for` qui compte de 0 à $n-1$ où n est la longueur du tableau.

```
void listing_tableau(int *t, int taille) {
    int i;
    for (i = 0; i < taille; i++)
        printf("%d: %d\n", i, t[i]);
}
```

2 Chaînes de caractères

(a) Écrire une fonction qui calcule la longueur d'une chaîne de caractères. Par exemple : `longueur("bon jour")` $\Longrightarrow$ `8`.

Bien qu'une chaîne de caractères soit *in fine* représentée par un tableau de caractères, les chaînes ont leurs méthodes particulières. La boucle standard pour parcourir sur une chaîne continue tant qu'un caractère nul n'est pas atteint. De plus, la présence d'une marque de fin fait une fonction qui agit sur une chaîne n'a pas besoin de connaître sa longueur. Voici deux manières d'écrire la fonction.

```
int longueur_chaine(char *s) {
    int i = 0;
    while (s[i] != 0)
        i++;
    return i;
}
```

```
int longueur_chaine(char *s) {
    int i;
    for (i = 0; s[i]; i++)
        ;
    return i;
}
```

(b) Écrire une fonction qui renvoie l'indice de la première occurrence d'un caractère dans une chaîne. La fonction renverra -1 *si le caractère n'est pas présent dans la chaîne.*

Ici, il y a deux conditions d'arrêt : soit on est arrivé au caractère recherché, soit on est arrivé à la fin de la chaîne. Voici deux manières d'écrire la fonction.

```
int premiere_occurrence(char *s, char c) {
    int i = 0;
    while (s[i] != 0 && s[i] != c)
        i++;
    if (s[i] == 0) return -1;
    else return i;
}
```

```
int premiere_occurrence(char *s, char c) {
    int i = 0;
    while (s[i] != 0) {
        if (s[i] == c) return i;
        i++;
    }
    return -1;
}
```

(c) Même question, mais en cherchant la dernière occurrence plutôt que la première.

Il y a deux méthodes différentes : on peut partir de la fin de la chaîne, et rechercher le caractère vers l'arrière ; ou on peut aller vers la droite mais noter les occurrences du caractère au fur et à mesure et retourner la dernière. Voici un exemple de chaque.

```
int derniere_occurrence_arriere(char *s, char c) {
    int i = longueur_chaine(s);
    while (i >= 0) {
        if (s[i] == c) return i;
        i--;
    }
    return -1;
}
```

```
int derniere_occurrence_avant(char *s, char c) {
    int i = 0;
    int n = -1;
    while (s[i] == 0) {
        if (s[i] == c) n = i;
        i++;
    }
    return n;
}
```

3 Écriture décimale

(a) Écrire une fonction qui prend comme paramètre une chaîne de caractères représentant un nombre entier sous forme décimale et qui renvoie le nombre entier correspondant. On suppose que la chaîne ne contient que des chiffres. Par exemple : `decimal("42")` $\Longrightarrow$ `42`.

(a') Même question, en supposant que la chaîne peut optionnellement commencer par un signe `+` *ou* `-`*.*

Il y a plusieurs façons de procéder, en parcourant la suite de chiffres vers l'avant ou vers l'arrière ; je n'en donne qu'une, qui a le double avantage de traiter les caractères dans l'ordre et de nécessiter le minimum de calculs. Je suppose que la chaîne est bien formée (un signe optionnel et des chiffres).

```
int lecture_decimale (char *s) {
    int a = 0, signe = 1, i = 0;
    if (s[0] == '-') { i = 1; signe = -1; }
    else if (s[0] == '+') i = 1;
    while (s[i] != 0) {
        a = a * 10 + (s[i] - '0')
        i++;
    }
    return a * signe;
}
```

(b) Écrire une fonction qui prend comme paramètre un nombre entier et qui écrit son écriture décimale dans une chaîne allouée globalement.

```
char tampon[50];
void ecriture_decimale(unsigned int x) { ... }
```

La difficulté est que les chiffres sont produits naturellement de droite à gauche. La méthode que j'utilise est de remplir le tableau `tampon` de droite à gauche ; il faut ensuite décaler les chiffres vers le début du tableau. Une autre méthode est de placer au début les chiffres dans l'ordre naturel puis de renverser la chaîne. Quelle que soit la méthode utilisée, il ne faut pas oublier de placer le caractère nul à la fin de la chaîne obtenue, et il faut faire attention au nombre 0 qui est représenté par la chaîne `"0"` et non par la chaîne `""`.

```
void ecriture_decimale(unsigned int x) {
    int i = 50, j = 0;
    if (x == 0) {
        tampon[0] = '0';
        tampon[1] = 0;
        return;
    }
    while (x != 0) {
        i--;
        tampon[i] = '0' + x % 10;
        x = x / 10;
    }
    while (i < 50) {
```

```
        tampon[j] = tampon[i];
        i++; j++;
    }
    tampon[j] = 0;
}
```

(c) Par quoi faut-il remplacer le nombre 50 pour être sûr d'avoir un tableau de caractères de longueur suffisante ? On utilisera les faits suivants :

- *L'opérateur* `sizeof` *permet de connaître le nombre de* bytes *qu'occupe une variable d'un type donné. Par exemple,* `sizeof(unsigned int)` *est une constante entière, dont la valeur est le nombre de* bytes *occupés par un* `unsigned int`*.*
- *La constante* `CHAR_BIT`*, définie dans l'en-tête* `limits.h`*, indique le nombre de bits (chiffres binaires) dans un* byte*. On pourra utiliser l'approximation suivante : un chiffre décimal suffit pour représenter trois bits.*

Il y a `(CHAR_BIT * sizeof(unsigned int))` bits dans un entier non signé. D'autre part, puisque un chiffre décimal permet de représenter trois bits, il suffit pour représenter `n` bits de `(n + 2) / 3 - 1` chiffres décimaux (c'est $n/3$, mais arrondi au-dessus et non au-dessous). D'où la taille du tampon (en n'oubliant pas le caractère nul final) :

```
(CHAR_BIT * sizeof(unsigned int) + 2) / 3
```

(c') Autre méthode : utiliser la constante `UINT_MAX` définie dans `limits.h`, qui est le plus grand nombre représentable dans une variable de type `unsigned int`.

On calcule le logarithme en base 10 de `UINT_MAX`. Le résultat de `nombre_de_chiffres_decimaux(UINT_MAX)`, plus 1 pour le caractère nul, est la meilleure taille possible pour le tableau `tampon`.

```
unsigned int nombre_de_chiffres_decimaux(unsigned int x) {
    unsigned int n = 0;
    while (x != 0) {
        n++;
        x = x / 10;
    }
    return n;
}
```

4 Histogrammes

On se donne un tableau d'entiers positifs `a` *à* `n` *éléments. Un histogramme correspondant à ce tableau est un graphique qui contient pour chaque entrée du tableau un rectangle dont la longueur est proportionnelle à la valeur de l'entrée du tableau.*

Utiliser la fonction `putchar` *pour afficher un histogramme en mode texte, comme ceci :*

```
******
**

***
```

(pour le tableau `{6, 2, 0, 3}`*).*

On fait deux boucles imbriquées, une à l'extérieur pour parcourir le tableau `a`, et une à l'intérieur pour afficher autant d'étoiles qu'il faut.

```
#include <stdio.h>
void histogramme_horizontal(int *a, int n) {
    int i, k;
    for (i = 0; i < n; i++) {
        for (k = 0; k < a[i]; k++) putchar('*');
        putchar('\n');
    }
}
```

(c) Même question, mais on veut dessiner l'histogramme dans l'autre sens :

```
*
*
*
*  *
** *
** *
```

C'est plus difficile. On commence par déterminer la hauteur `m` de l'histogramme, qui est la plus grande entrée du tableau. Ensuite, on a encore deux boucles imbriquées, avec la boucle extérieure qui correspond aux lignes et la boucles intérieure qui correspond aux colonnes. Cette fois, sur chaque ligne, on affiche une étoile seulement dans les colonnes correspondant aux entrées du tableaux qui sont supérieures au numéro de la ligne.

```
#include <stdio.h>
void histogramme_vertical(int *a, int n) {
    int i, k, m;
    m = 0;
    for (i = 0; i < n; i++)
        if (a[i] > m) m = a[i];
    for (k = m; k > 0; k--) {
        for (i = 0; i < n; i++) {
            if (a[i] >= k) putchar('*');
            else putchar(' ');
        }
        putchar('\n');
    }
}
```

5 Recherche d'un élément dans un tableau trié

On veut écrire une fonction qui recherche un élément x dans un tableau d'entiers triés en ordre croissant. Bien sûr, on peut utiliser une fonction de recherche écrite pour le cas général, mais il y a plus efficace. Au lieu de regarder les éléments du tableau de gauche à droite, on commence par regarder l'élément du milieu a. Si a est égal à x, on a gagné. Si a est plus grand que x, on sait que x ne peut être que avant a : on a éliminé la moitié de l'espace de recherche en une comparaison. De même, si $a < x$, on n'a plus qu'à chercher x dans la deuxième moitié du tableau. Tant qu'on n'a pas trouvé x, on applique cette méthode à un sous-tableau dont la taille diminue de moitié à chaque coup. Si on arrive à un limiter x à un sous-tableau de taille 0, on sait que x est absent.

Cette technique s'appelle une *recherche dichotomique*. On recherche x dans le sous-tableau composé des éléments de `a` d'indice compris entre `debut` et `(fin-1)`. À chaque tour de boucle, la taille de ce sous-tableau diminue de moitié ; c'est ce qui rend cette méthode beaucoup plus efficace que la précédente.

Note : attention aux indices. Comme `fin` est toujours strictement supérieur à `debut` si on rentre dans la boucle, `pivot` ne reçoit jamais la valeur `fin` ; lors du dernier passage dans la boucle, on a `fin == debut + 1` ou `fin == debut + 2` et `pivot` prend la valeur de `fin` moins un. Si vous n'êtes pas convaicus, regardez ce qui se passe pour des tableaux de taille 0, 1, 2, 3, 4, 5.

```
int recherche_dichotomique(int *a, int taille, int x) {
    int debut = 0, fin = taille, pivot;
    while (fin > debut) {
        pivot = (fin + debut) / 2;
        if (x == a[pivot])          return pivot;
        else if (x > a[pivot])      debut = pivot + 1;
        else /* (x < a[pivot]) */   fin = pivot;
    }
    return -1;
}
```