Les sous-programmes: exercices corrigés
Corrigé

Exercice 1 : Sous-programmes sur les chaînes de caractères

L'objectif de cet exercice est de définir les opérations suivantes sur les chaînes de caractères.

- obtenir la longueur d'une chaîne ;
- initialiser une chaîne à partir d'une autre ;
- comparer deux chaînes de caractères. On retournera un nombre négatif si la première est inférieure, nulle si les deux sont égales ou positif si c'est la deuxième qui est la plus grande. On utilisera naturellement l'ordre lexicographique ;
- concaténer deux chaînes de caractères qui consiste à ajouter à la fin d'une première chaîne les caractères d'une autre chaîne ;
- initialiser une chaîne à partir des (au plus) n premiers caractères d'une autre chaîne.

On supposera que les chaînes de caractères fournies en paramètre des opérations auront des capacités suffisantes !

Écrire les sous-programmes qui correspondent aux opérations ci-dessus.

Solution : Traitons les opérations demandées dans l'ordre.

obtenir la longueur d'une chaîne. L'objectif est « obtenir la longueur d'une chaîne de caractères ». Ce sous-programme prend deux paramètres :

- la chaîne dont on veut obtenir la longueur (`in`) ;
- la longueur de la chaîne (`out`).

Un seul paramètre en `out`, d'autres qui sont tous en `in`, on peut donc en faire un fonction.

Voici la spécification de ce sous-programme.

```
1 Fonction longueur(ch: in Chaine): Entier Est
2     -- Longueur de la chaîne de caractères ch.
```

Passons à l'implantation. Le principe est de compter le nombre de caractères qui précède le caractère nul, indicateur de la fin de la chaîne. On en déduit donc l'algorithme suivant.

C

```
1 /* Longueur de la chaîne de caractère ch.
2  * Cette fonction correspond à strlen de <string.h>.
3  */
4 int longueur(const char ch[])
5 {
6     int indice = 0;     /* parcourir les caractères de ch */
7     while (ch[indice] != '\0') {
8             /* Remarque : on est sûr que la chaîne se termine.
9              * Il est donc inutile de tester la capacité.
```

```
10              */
11          indice++;
12      }
13      return indice;
14  }
```

Initialiser une chaîne à partir d'une autre. Il s'agit de faire une copie de chaînes. Ce sous-programme prend deux chaînes en paramètre : la source (`in`) et la destination (`out`), c'est-à-dire celle que l'on initialise.

Les paramètres laissent penser que l'on peut faire une fonction, mais il s'agit d'une initialisation. Dans ce cas, on fait une procédure.

```
1  Procédure initialiser(destination: out Chaine; source: in Chaine) Est
2      -- Initialiser la chaîne destination à partir de la chaîne source.
3      --
4      -- Nécessite :
5      --          capacité(destination) > longueur(source)
```

Le code de cette procédure consiste à copier tous les caractères de source dans destination jusqu'au caractère `'\0'` inclu.

C

```
1  /* Initialiser destination à partie de source.
2   * Cette fonction correspond à strcpy de <string.h>.
3   *
4   * Nécessite
5   *      capacité de destination > longueur(source);
6   */
7  void initialiser(char destination[], const char source[])
8  {
9      int indice = 0;     /* parcourir les caractères de ch */
10     while (source[indice] != '\0') {
11         destination[indice] = source[indice];
12         indice++;
13     }
14     destination[indice] = '\0';
15 }
```

En s'appuyant sur l'arithmétique des pointeurs, on peut écrire une version beaucoup plus consises mais aussi beaucoup moins lisible :

```
1  /* Initialiser destination à partie de source.
2   * Cette fonction correspond à strcpy de <string.h>.
3   *
4   * Nécessite
5   *      capacité de destination > longueur(source);
6   */
7  void initialiser_illisible(char *destination, const char *source)
8  {
9      while ((*destination++ = *source++))
10             /* Les doubles parenthèses sont pour éviter le message
```

```
11              * d'avertissement émis par le compilateur car il voit une
12              * affectation dans une condition.
13              */
14          ;
15  }
```

Concaténer deux chaînes de caractères. L'objectif est déjà énoncé. Les paramètres sont la chaîne à augmenter (`in out`) et la chaîne source (`in`). On fait donc une procédure.

```
1  Procédure concaténer(destination: in out Chaîne; source: in Chaîne) Est
2      -- ajouter la chaîne source à la fin de la chaîne destination
3      --
4      -- Nécessite
5      --  capacité(destination) > longueur(destination) + longueur(source)
```

Le code de sous-programme consiste à chercher la fin de la chaîne destination (en répérant le zéro final) puis à copier les éléments de source à partir de cette position.

C

```
1  /* Ajouter la chaîne source à la fin de la chaîne destination.
2   * Cette fonction correspond à strcat de <string.h>.
3   *
4   * Nécessite
5   *      capacité(destination) > longueur(destination) + longueur(source);
6   */
7  void concatener(char destination[], const char source[])
8  {
9      int idest;          /* parcourir les caractères de destination */
10     int isource;        /* parcourir les caractères de source */
11
12     /* Chercher la fin de destination */
13     idest = 0;
14     while (destination[idest] != '\0') {
15         idest++;
16     }
17
18     /* Copier les caracèters de source */
19     isource = 0;
20     while (source[isource] != '\0') {
21         destination[idest] = source[isource];
22         idest++;
23         isource++;
24     }
25
26     /* Mettre le zéro final */
27     destination[idest] = '\0';
28 }
```

Initialiser une chaîne à partir des (au plus) n premiers caractères d'une autre chaîne. Il s'agit d'une initialisation, donc on fait une procédure. Les paramètres sont la chaîne à initialiser

(out), la chaîne qui sert à initialiser (in), le nombre maximal de caractères à copier (in).

```
Procédure initialiser_max(destination: out Chaine;
                          source: in Chaine;
                          nb_max: in Entier) Est
    -- Initialiser destination avec les (au plus) n premiers caractères de
    -- source.
    --
    -- Nécessite
    --  capacité de destination >= nb_max
```

Le code est le même que pour initialiser sauf qu'il y a une condition d'arrêt supplémentaire portant sur le nombre de caractères copiés.

C

```c
/* Initialiser destination à partie de source dans la limite des (au plus)
 * nb_max premiers caractères de source.
 * Cette fonction correspond à strncpy de <string.h>.
 *
 * Nécessite
 *      capacité de destination >= nb_max
 */
void initialiser_max(char destination[], const char source[], int nb_max)
{
    int indice = 0;     /* parcourir les caractères de ch */
    while (source[indice] != '\0' && nb_max > 0) {
        destination[indice] = source[indice];
        indice++;
        nb_max--;
    }
    if (nb_max > 0) {
            /* Attention : strncat ne met le zéro final que si moins
             * de nb_max caractères ont été copiés !
             */
        destination[indice] = '\0';
    }
}
```

Remarque : Ces opérations existent déjà dans le module `string.h` qui contient des fonctions telles que strlen, strcpy, strcmp, strcat, strncpy, etc.

Exercice 2 : Livres

L'objectif de cet exercice est d'écrire un programme pour gérer une bibliothèque personnelle de taille réduite. On ne développera qu'une interface homme/machine minimale.

2.1 *Définition d'un livre.* Un livre est caractérisé par son titre, son nombre de pages et son genre. Le genre est soit roman, soit BD, soit art, soit technique, etc. Il faudrait bien sûr bien d'autres informations pour décrire complètement un livre.

2.1.1 Donner le type Livre.

Solution : D'après sa caractérisation, un livre peut être représenté par un type énuméré avec des champs titre (une chaîne de caractères), son nombre de page (entier) et son genre (un type énuméré puisqu'il peut prendre une parmi plusieurs valeurs).

```
Constante
    LG_TITRE = 50       -- Longueur maximal pour le titre d'un livre
Type
    Genre = (ROMAN, BD, ART, TECHNIQUE)

    Livre =
        Enregistrement
            titre: Chaîne[LG_TITRE]
            nb_pages: Entier
            genre: Genre
        FinEnregistrement
```

C

```
#define LG_TITRE 50     /* Longueur maximal pour le titre d'un livre */

enum Genre { ROMAN, BD, ART, TECHNIQUE };

typedef enum Genre Genre;

struct Livre {
    char titre[LG_TITRE+1];
    int nb_pages;
    Genre genre;
};

typedef struct Livre Livre;
```

2.1.2 Écrire un sous-programme qui permet d'initialiser un livre à partir de son titre, de son nombre de pages et de son genre.

Solution : Ce sous-programme prend en paramètre :

- le livre à initialiser (**out**) ;
- le titre du livre (**in**) ;
- le nombre de pages (**in**)
- et le genre du livre (**in**).

Comme il s'agit d'un sous-programme d'initialisation, nous en faisons une procédure dont voici la spécification.

```
Procédure initialiser_livre(livre: out Livre;
                            titre: in Chaîne;
                            npage: in Entier;
                            genre: in Genre) Est
    -- Initialiser un livre à partir de son titre, son nombre de pages
    -- et son genre.
    --
    -- Nécessite
    --  npage > 0
    --
    -- Assure
    --  livre.titre = titre
    --  livre.nb_pages = npages
```

```
14      --  livre.genre = genre
```

Le code du sous-programme est alors immédiat puisqu'il s'agit d'initialiser tous les champs de l'enregistrement.

```
1  Début
2      livre.titre ← titre
3      livre.nb_pages ← npages
4      livre.genre ← genre
5  Fin
```

C

```
1  /* Initialiser un livre à partir de son titre, son nombre de pages et
2   * son genre.
3   *
4   * Nécessite
5   *      npages > 0;
6   *
7   * Assure
8   *      strcmp(livre->titre, titre) == 0;
9   *      livre->nb_pages == npages;
10  *      livre->genre == genre;
11  */
12 void initialiser_livre(Livre *livre,
13                      const char titre[], int npages, Genre genre)
14 {
15     assert(npages > 0);
16
17     strncpy(livre->titre, titre, LG_TITRE);
18     livre->titre[LG_TITRE] = '\0';
19     livre->nb_pages = npages;
20     livre->genre = genre;
21
22     assert(strcmp(livre->titre, titre) == 0);
23     assert(livre->nb_pages == npages);
24     assert(livre->genre == genre);
25 }
```

2.1.3 Écrire un sous-programme qui permet d'afficher un livre.

Solution : Ce sous-programme n'a qu'un seul paramètre, le livre qui est en entrée. De plus il fait de l'affichage. Il s'agit donc nécessairement d'une procédure.

```
1  Procédure afficher_livre(livre: in Livre) Est
2      -- Afficher un livre
```

Le code consiste à afficher chacun des champs.

```
1  Début
2      Afficher livre.titre
3      Afficher livre.nb_pages
4      Afficher livre.genre
5  Fin
```

Pour afficher le genre du livre, on peut écrire un nouveau sous-programme.

C

```c
/* Afficher le genre d'un livre */
void afficher_genre(Genre genre)
{
    switch (genre) {
        case ROMAN:
            printf("ROMAN");
            break;
        case BD:
            printf("BD");
            break;
        case ART:
            printf("ART");
            break;
        case TECHNIQUE:
            printf("TECHNIQUE");
            break;
        default:
            printf("ERREUR !!!!");
    }
}

/* Afficher un livre. */
void afficher_livre(Livre livre)
{
    printf("%50s %3d pages ", livre.titre, livre.nb_pages);
    afficher_genre(livre.genre);
}
```

2.2 *La bibliothèque.* Nous nous limitons à une bibliothèque qui ne peut pas contenir plus de 1000 ouvrages qui sont rangés dans l'ordre alphabétique de leur titre.

2.2.1 Définir le type bibliothèque.

Solution : Nous représentons la bibliothèque par un tableau de capacité 1000. Cependant, pour connaître le nombre d'ouvrages effectivement présents, il faut gérer la taille (entier). La bibliothèque est donc un type enregistrement.

```
Constante
    MAX_LIVRES = 1000   -- nb maximal de livres dans la bibliothèque
Type
    Bibliothèque =
        Enregistrement
            livres: Tableau [1..MAX_LIVRES] De Livre
            nb_livres: Entier
        FinEnregistrement
```

C

```c
#define MAX_LIVRES 1000
        /* nombre maximal de livres dans la bibliothèque */

```

```
4  struct Bibliotheque {
5      Livre livres[MAX_LIVRES];
6      int nb_livres;
7  };
8
9  typedef struct Bibliotheque Bibliotheque;
```

2.2.2 Écrire un sous-programme pour initialiser une bibliothèque à vide.
Solution : Ce sous-programme prend en paramètre la bibliothèque (**out**). Il s'agit d'une procédure.

```
1  Procédure initialiser_bibliothèque(b: out Bibliothèque) Est
2      -- Initialiser la bibliothèque à vide
3      --
4      -- Assure
5      --  b.nb_livres = 0 -- bibliothèque vide
```

Le code est donc immédiat.

```
1  Début
2      b.nb_pages ← 0
3  Fin
```

Notons qu'il n'est pas nécessaire d'initialiser les livres du tableau puisque la champs `nb_pages` mis à zéro indique qu'on ne peut accéder à aucune case du tableau `livres`.

C

```
1  /* Initialiser une bibliothèque à vide.
2   *
3   * Assure
4   *      b->nb_livres == 0
5   */
6  void initialiser_bibliotheque(Bibliotheque *b)
7  {
8      b->nb_livres = 0;
9
10     assert(b->nb_livres == 0);
11 }
```

2.2.3 Écrire un sous-programme pour afficher le contenu de la bibliothèque.
Solution : Ce sous-programme prend un seul paramètre, la bibliothèque (en **in**). Il s'agit donc d'une procédure.

```
1  Procédure afficher_bibliothèque(b: in Bibliothèque) Est
2      -- Afficher les livres de la bibliothèque.
```

Pour l'implémentation, nous choisissons d'utiliser une répétition **Pour** puisque nous connaissons le nombre de livres à afficher.

```
1  Variable
2      i: Entier   -- parcourir les livres
3  Début
4      Pour i ← 1 JusquÀ i = b.nb_livres Faire
```

```
5           afficher_livres(b.libres[i])
6       FinPour
7   Fin
```

C

```c
/* Afficher les livres de la bibliothèque. */
void afficher_bibliotheque(Bibliotheque b)
{
    int i;      /* parcourir les livres */

    for (i = 0; i < b.nb_livres; i++) {
        afficher_livre(b.livres[i]);
        printf("\n");
    }
}
```

2.2.4 Écrire un sous-programme pour ajouter un ouvrage dans la bibliothèque.

Solution : Ce sous-programme prend en paramètre :

- la bibliothèque dans laquel le livre doit être ajouté (**in out**) ;
- le livre à ajouter (**in**).

Il s'agit donc d'une procédure.

```
Procédure ajouter(b: in out Bibliothèque; l: in Livre) Est
    -- ajouter le livre l dans la bibliothèque b
```

Les livres sont rangés dans la bibliothèque dans l'ordre lexicographique des titres. Il s'agit donc de faire un tri par insersion pour trouver la place de du nouveau livre. Nous optons pour une insertion séquentielle dont voici le premier niveau de raffinage.

```
R1 : Raffinage De « ajouter »
  | Déterminer la position du livre     posision: out Entier
  | Décaler les livres compris entre posision et b.nb_livres
  | Ranger le nouveau livre
```

Ces étapes correspondant à des algorithmes classiques sur les tableaux, il ne sont pas détaillés en algorithmique.

C

```c
/* Ajouter le livre l dans la bibliothèque b.
 *
 * Nécessite
 *      b->nb_livres < MAX_LIVRES;      -- non pleine
 */
void ajouter(Bibliotheque *b, Livre l)
{
    int position;       /* position où doit être rangé le livre l */
    int i;              /* parcourir les livres */

    /* Déterminer la position du livre */
    position = 0;
    while (position < b->nb_livres
           && strcmp(l.titre, b->livres[position].titre) >= 0)
```

```c
    {
        position++;
    }

    /* Décaler les livres */
    for (i = b->nb_livres - 1; i >= position; i--) {
        b->livres[i+1] = b->livres[i];
    }

    /* Ranger le livre */
    b->livres[position] = l;
    b->nb_livres++;
}
```

2.2.5 Écrire un sous-programme pour indiquer combien il y a d'ouvrages d'un genre donné.

Solution : Ce sous-programme a pour paramètres :

- la bibliothèque dans laquelle on veut recenser les ouvrages (**in**) ;
- le genre cherché (**in**) ;
- le nombre d'ouvrages du genre cherché (**out**).

Un seul paramètre en sortie, deux en entrée. On peut donc en faire une fonction.

```
Fonction nb_livres_genre(b: in Bibliothèque; genre: in Genre): Entier Est
    -- Nombre de livres de la bibliothèque b qui sont du genre spécifié.
```

Concernant l'implantation, il faut parcourir tous les ouvrages (boucle **Pour**) et comptabiliser ceux qui sont du genre demandé.

```
Variable
    i: Entier   -- parcourir les livres
Début
    Résultat ← 0
    Pour i ← 1 JusquÀ i = b.nb_livres Faire
        Si b.livres[i].genre = genre Alors
            Résultat ← Résultat + 1
        FinSi
    FinPour
Fin
```

C

```c
/* Nombre de livres de la bibliothèque b qui sont du genre spécifié. */
int nb_livres_genre(Bibliotheque b, Genre genre)
{
    int resultat;
    int i;      /* parcourir les livres */

    resultat = 0;
    for (i = 0; i < b.nb_livres; i++) {
        if (b.livres[i].genre == genre) {
            resultat++;
        }
    }
```

```
13      return resultat;
14  }
```

2.3 *Programme de test.* Écrire un programme de test des sous-programmes écrits ci-dessus.

Solution :

C Voici un programme de test minimal qui ajoute quelques livres dans une bibliothèque et affiche le nombre de livres des différents genres.

```
1  /* Programme de test minimal */
2  int main()
3  {
4      Bibliotheque b;
5      Livre l;
6
7      initialiser_bibliotheque(&b);
8      afficher_bibliotheque(b);
9
10     printf("\n\nAjouter le livre :\n");
11     initialiser_livre(&l, "FFF", 54, BD);
12     afficher_livre(l);
13     printf("\nLa bibliothèque contient :\n");
14     ajouter(&b, l);
15     afficher_bibliotheque(b);
16
17     printf("\n\nAjouter le livre :\n");
18     initialiser_livre(&l, "XXX", 120, ART);
19     afficher_livre(l);
20     printf("\nLa bibliothèque contient :\n");
21     ajouter(&b, l);
22     afficher_bibliotheque(b);
23
24     printf("\n\nAjouter le livre :\n");
25     initialiser_livre(&l, "CCC", 540, TECHNIQUE);
26     afficher_livre(l);
27     printf("\nLa bibliothèque contient :\n");
28     ajouter(&b, l);
29     afficher_bibliotheque(b);
30
31     printf("\n\nAjouter le livre :\n");
32     initialiser_livre(&l, "DDD", 140, TECHNIQUE);
33     afficher_livre(l);
34     printf("\nLa bibliothèque contient :\n");
35     ajouter(&b, l);
36     afficher_bibliotheque(b);
37
38     printf("Nb livres ART : %d\n", nb_livres_genre(b, ART));
39     printf("Nb livres BD : %d\n", nb_livres_genre(b, BD));
40     printf("Nb livres ROMAN : %d\n", nb_livres_genre(b, ROMAN));
41     printf("Nb livres TECHNIQUE : %d\n", nb_livres_genre(b, TECHNIQUE));
42
43     return EXIT_SUCCESS;
44 }
```