Les types utilisateurs : exercices corrigés
Corrigé

Objectifs

- Connaître les types utilisateurs : tableaux, enregistrements et type énumérés ;
- Connaître les algorithmes fondamentaux sur les tableaux ;
- Continuer à appliquer les principes de la semaine 1.

Exercice 1 : Min et max d'un tableau
Étant donné un tableau de réels, calculer le plus petit élément et le plus grand élément du tableau.
Solution :

```
R0 : Déterminer le min et le max des éléments d'un tableau.
```

Principe : On considère les éléments du tableau les uns après les autres et on utilise deux variables accumulateurs pour stocker la valeur du max et du min.

```
Tests :
    tab          -> min  max
    1 2 3        -> 1    3
    3 2 1        -> 1    3
    5            -> 5    5
    5 2 1 9 4    -> 1    9
    vide         -> indéterminé
```

Dans les jeux de tests précédents, on constate que dans le cas limite où le tableau est vide, on ne peut pas calculer le min et le max.

```
Dictionnaire Des Données
    tab: Tableau [0..MAX] De Réel       -- le tableau considéré
    nb: Entier              -- la taille effective de tab
    min, max: Entier        -- le min et le max des éléments du tableau
    i: Entier               -- indice pour parcourir le tableau
```

On note que le tableau a une capacité de MAX + 1 éléments.

```
R1 : « Déterminer le min et le max des éléments d'un tableau »
  | Si nb > 0 Alors
  |    | min ⟵ tab[0]
  |    | max ⟵ tab[0]
```

```
|   | Pour i ← 1 JusquÀ i = nb Faire
|   |   | { Invariant : min = Min(tab(0..i-1)) Et max = Max(tab(0..i-1)) }
|   |   | Si tab[i] < min Alors
|   |   |   | min ← tab[i]
|   |   | SinonSi tab[i] > max Alors
|   |   |   | max ← tab[i]
|   |   | FinSi
|   | FinPour
| FinSi
```

C

```c
/*****************************************************************
 *  Auteur   : Xavier Crégut <cregut@enseeiht.fr>
 *  Version  : 1.1
 *  Objectif : Déterminer le min et le max d'un tableau.
 *****************************************************************/

#include <stdio.h>
#include <stdlib.h>

#define MAX     10

int main()
{
    double tab[MAX];    /* le tableau considéré */
    int nb;             /* la taille effective de tab */
    double min, max;    /* le min et le max des éléments du tableau */
    int i;              /* indice pour parcourir le tableau */
    double valeur;      /* valeur saisie au clavier */

    /* Initialiser le tableau (saisie clavier) */
    printf("Entrez une valeur nulle pour arrêter.\n");
    nb = 0;
    scanf("%lf", &valeur);
    while (valeur != 0  && nb < MAX) {
        /* ranger la valeur */
        tab[nb] = valeur;
        nb++;
        printf("valeur = %f\n", tab[nb-1]);

        /* Saisir une nouvelle valeur */
        scanf("%lf", &valeur);
    }
    if (valeur >= 0) {
        printf("Capacité du tableau atteinte.  Arrêt de la saisie\n");
    }

    /* Déterminer le min et le max du tableau */
    if (nb > 0) {
        min = max = tab[0];
        for (i = 1; i < nb; i++) {
```

```
                if (tab[i] < min) {
                    min = tab[i];
                }
                else if (tab[i] > max) {
                    max = tab[i];
                }
            }
        }

        /* Afficher le min et le max */
        if (nb > 0) {
            printf("min = %f\n", min);
            printf("max = %f\n", max);
        }

        return EXIT_SUCCESS;
    }
```

Attention : Dans la version C, la capacité du tableau n'est pas MAX+1 mais seulement MAX d'où la condition `nb < MAX` et non `nb <= MAX` !

Exercice 2 : Modélisation d'une facture simplifiée

Une facture est composée de plusieurs lignes. Chaque ligne correspond à la description d'un produit avec sa désignation, son prix unitaire hors taxe, la quantité commandée et le prix total hors taxe de ce produit. Au bas de facture apparaît le prix total hors taxe et toutes taxes comprises, ainsi que le montant des taxes.

Définir un type représentant une telle facture.

Remarque : Cette facture est dite simplifiée car nous ne tenons compte ni de l'entête (nom du fournisseur, du client, etc.), ni du bas de la facture (date, signatures, etc.).

Solution : Le texte, et plus précisément les mots utilisés dans le texte, nous permettent de définir les types de données nécessaires.

Le premier, le principal puisqu'il apparaît dans la question posée, est « facture ». Nous aurons donc un type Facture. Qu'est ce qu'une facture ? Plusieurs lignes. Plusieurs donc on peut prendre un tableau (ceci est réducteur, plus généralement, il faut prendre une structure de données de type « container » et la seule dont nous disponsons actuellement est le tableau). Si on prend un tableau, il faudra alors définir une capacité et dans le cas présent une taille effective, c'est-à-dire le nombre de lignes qui apparaissent réellement sur la facture. Le type Facture est donc un enregistrement dont l'un des champs est un tableau de lignes.

```
Constante
    MAX_LIGNE = 40      -- nombre maximal de lignes sur une facture

Type
    Facture =
        Enregistrement
            lignes: Tableau [1..MAX_LIGNE] De Ligne
            nb: Entier          -- nombre de lignes sur la facture.
        FinEnregistrement
```

Qu'est ce qu'une ligne ? C'est un produit (désignation, prix unitaire hors taxe), la quantité commandé et le prix total de la ligne. Un produit est donc un enregistrement.

```
Type
    Designation = Caractère
        -- Nous prenons un type Caractère pour rester sur un type
        -- élémentaire.  Il serait certainement plus judicieux de
        -- prendre une chaîne de caractères.

    Produit =
        Enregistrement
            designation: Designation
            prix_ht: Réel               -- prix hors taxe
        FinEnregistrement

    Ligne =     -- Une ligne d'une facture
        Enregistrement
            produit: Produit
            quantité: Entier    -- toujours > 0
            prix_ht             { prix = quantité * produit.prix_ht }
        FinEnregistrement
```

Remarque : Notons que le prix total hors taxe d'une ligne (`prix_ht`) peut se déduire de la quantité de produit commandé et du prix hors taxe de ce produit. Il ne serait donc pas nécessaire de le stocker dans le type Ligne. Le conserver utilise un peu plus de mémoire mais permet d'éviter de refaire des calculs. Il s'agit de faire un compromis espace mémoire et temps de calcul.
Remarque : L'identifiant `prix_ht` est utilisé pour désigner aussi bien le prix d'un produit et le total d'une ligne. Il ne peut cependant pas y avoir de confusion entre les deux car, dans la mesure où il s'agit du nom d'un champ, il sera toujours appliqué à une variable qui sera soit un produit, soit en ligne. Le contexte nous dira donc s'il s'agit du prix hors taxe d'un produit ou du total d'une ligne.

Concernant le total (hors taxe et ttc) de la facture ainsi que le montant des taxes, on peut soit ajouter des champs supplémentaires dans l'enregistrement du type Facture (on stocke de l'information) ou considérer qu'ils seront déduits des autres informations (on calcule l'information). Ici, avec les types proposés, nous avons choisi de ne pas les stocker et donc de les calculer quand on en a besoin.
Remarque : Si on décide de les stocker, il faudra faire attention qu'ils restent cohérents, en particulier si on ajoute une ligne à une facture, si on modifie une quantité ou le prix hors taxe d'un produit.

C

```c
#define MAX_LIGNE 40    /* Nombre maximal de lignes sur une facture */

typedef char Designation;       /* Désignation d'un produit */

struct Produit {
    Designation designation;
    double prix_ht;     /* prix hors taxe */
};

typedef struct Produit Produit;

struct Ligne {
    Produit produit;
    int quantite;       /* > 0 */
    double prix_ht;     /* prix_ht = quantite * produit.prix_ht */
};

typedef struct Ligne Ligne;

struct Facture {
    Ligne lignes[MAX_LIGNE];
    int nb;     /* nombre effectif de lignes sur la facture */
};

typedef struct Facture Facture;
```

Exercice 3 : Lecture d'un tableau

Écrire un programme qui initialise un tableau d'entiers en lisant des valeurs au clavier. On considère que l'utilisateur commence par donner la taille du tableau puis les valeurs elles-mêmes.

Solution :

```
R0 : Saisir un tableau

Principe : demander d'abord la taille puis les éléments. Ceci est imposé
par le sujet.

R1 : Raffinage De « Saisir un tableau »
  | Demander le nombre d'éléments (contrôle)    nb: out Entier
  | { (nb >= 0) Et (nb <= CAPACITE)}
  | Saisir les éléments du tableau

R2 : Raffinage De « Saisir les éléments du tableau »
  | Pour i ← 1 JusquÀ i = nb Faire
  |   | Lire(tab[i])
  | FinPour
```

C

```c
/*****************************************************************
 *  Auteur   : Xavier Crégut <cregut@enseeiht.fr>
 *  Version  : 1.4
 *  Objectif :
 *        saisir un tableau en demandant d'abord le nombre d'éléments
 *        puis les éléments eux-mêmes.
 *****************************************************************/

#include <stdio.h>
#include <stdlib.h>

#define CAPACITE        10      /* capacité du tableau à lire */

int main()
{
    int tab[CAPACITE];  /* la tableau */
    int nb;             /* la taille effective du tableau */
    int i;

    /* Saisir le tableau */

    /* Demander le nombre d'éléments */
    do {
        printf("Nb d'éléments : ");
        scanf("%d", &nb);
        if (nb < 0) {
            printf("Le nb d'éléments doit être > 0 !\n");
        }
        else if (nb > CAPACITE) {
            printf("Désolé, le nombre d'éléments ne doit pas dépasser %d.\n",
```

```
                    CAPACITE);
            }
        } while (nb < 0 || nb > CAPACITE);

        /* Saisir les éléments du tableau */
        for (i = 0; i < nb ; i++) {
            /* saisir tab[i] */
            printf("\ttab[%d] = ", i);
            scanf("%d", &tab[i]);
        }

        /* Afficher le tableau */
        printf("[");
        if (nb > 0) {       /*{ le tableau n'est pas vide }*/
            int i;  /* pour parcourir les indices du tableau */

            /* afficher le premier élément */
            printf("%d", tab[0]);

            /* afficher les autres éléments */
            for (i=1; i < nb; i++) {
                printf("; %d", tab[i]);
            }
        }
        printf("]");

        return EXIT_SUCCESS;
    }
```

Exercice 4 : Recherche d'occurrences

Dans cet exercice, nous considérons un tableau d'entiers et nous nous intéressons aux occurrences de ses éléments.

4.1 Indiquer si un entier donné est présent dans le tableau ou non.

Solution : On suppose, comme pour les questions suivantes, que le tableau est déjà initialisé. Il est défini par un nom de variable, une taille effective. On suppose que les indices commencent à 1. Dans le cas contraire, il suffirait de faire une translation.

```
Constante
    CAPACITÉ = 100      -- la capacité du tableau
Variable
    tab: Tableau [1..CAPACITÉ] De Entier        -- le tableau
    nb: Entier          -- la taille effective du tableau
```

Pour savoir si une valeur x est présente dans le tableau, il suffit de comparer toutes les valeurs du tableau à x. On s'arrête soit quand une valeur correspond (x est présent dans le tableau) ou que toutes les valeurs ont été testées sans succès (x n'est pas présent). On utilise donc une boucle **TantQue** : on ne sait pas à priori combien d'itérations sont nécessaires et le tableau peut être vide (aucune comparaison à faire).

```
Variable
    x: Entier           -- entier à chercher dans le tableau
    i: Entier           -- parcourir les indices du tableau
    présent: Booléen    -- x est-il présent dans le tableau ?
Début
    i ← 1;
    présent ← false
    TantQue (i <= nb)           -- il reste des éléments à comparer
            Et (non présent)    -- on n'a pas trouvé x
    Faire
        { Invariant : présent = x est présent dans tab[1..i-1] }
        { Variant : nb - i + 1 }
        présent ← (tab[i] = x)
        i ← i + 1
    FinTQ
Fin.
```

Remarque : Il serait possible de résoudre ce problème en utilisant un **Pour** mais ce serait maladroit (et donc faux pour nous !) car dès qu'on a trouvé une occurrence de x, il est inutile de regarder les valeurs qui suivent dans le tableau.

C Les indices du tableau commencent nécessairement à 0. Ceci modifie légèrement les algorithmes.

```
/* Déterminer si une valeur est présente dans le tableau */
i = 0;
present = false;
while (i < nb && ! present) {
    present = tab[i] == x;
    i++;
}
```

Remarquons que ceci peut s'écrire plus simplement si on suppose que les opérateurs booléen ont une évaluation paresseuse (en court-circuit). Ceci garantit que l'on accède toujours à une case valide du tableau.

```
Variable
    x: Entier           -- entier à chercher dans le tableau
    i: Entier           -- parcourir les indices du tableau
    présent: Booléen    -- x est-il présent dans le tableau ?
Début
    i ← 1;
    TantQue (i <= nb)           -- il reste des éléments à comparer
            Et (tab[i] <> x)    -- on n'a pas trouvé x
    Faire
        i ← i + 1
    FinTQ
    { Non ((i <= nb) Et (tab[i] <> x)) <==> (i > nb) Ou (tab[i] = x) }
    présent ← (i <= nb)
Fin.
```

Notons que l'on commence par vérifier que i est indice valide **avant** de comparer la valeur dans le tableau à x. Si on fait l'inverse, on risque d'accéder au tableau avec un indice invalide ce qui est une erreur.

Pour conclure sur la présence de l'élément il faut exploiter la condition de sortie du **TantQue** mais, attention, il ne faut pas tester `tab[i] = x` car i n'est peut être pas un indice valide ! Il faut tester `i <= nb`. Si `i <= nb` c'est nécessairement que `tab[i] = x` puisqu'on est sortie de la boucle !

C

```
/* Déterminer si une valeur est présente dans le tableau */
i = 0;
while (i < nb && ! tab[i] != x) {
    i++;
}
present = i < nb;
```

Une autre version de la première version serait la suivante :

```
Variable
    x: Entier           -- entier à chercher dans le tableau
    i: Entier           -- parcourir les indices du tableau
    présent: Booléen    -- x est-il présent dans le tableau ?
Début
    i ← 0;
    présent ← false
    TantQue (i < nb)            -- il reste des éléments à comparer
            Et (non présent)    -- on n'a pas trouvé x
    Faire
        { Invariant : présent = x est présent dans tab[1..i] }
        { Invariant : i <= nb }
        { Variant : nb - i }
        i ← i + 1
        présent ← (tab[i] = x)
```

```
    FinTQ

    Résultat ← présent
Fin.
```

4.2 Indiquer l'indice, dans le tableau, de la première occurrence d'un entier donné.

Solution : Le principe est le même que précédemment, sauf que le résultat n'est plus un booléen mais l'indice de l'élément.

Une question se pose : quelle valeur donner à indice si l'entier cherché n'est pas présent dans le tableau ? Par *convention*, on considérera que si l'élément cherché n'est pas présent on a indice qui est plus grand que le dernier indice valide, donc nb + 1.

Il suffit de remplacer la dernière ligne de l'algorithme précédent par la ligne suivante :

```
indice ← i - 1;
```

ou la ligne suivante pour la deuxième version de l'algorithme

```
indice ← i;
```

C

```
/* Déterminer l'indice de la première occurrence */
i = 0;
while (i < nb && tab[i] != x) {
    i++;
}
present = i < nb;
indice = i;
```

4.3 Indiquer l'indice, dans le tableau, de la dernière occurrence d'un entier donné.

Solution : Deux solutions :

1. soit on parcourt les indices de nb à 1. Dans ce cas, on peut utiliser le même algorithme que ci-dessus.

   C

   ```
   /* Déterminer l'indice de la première occurrence */
   i = nb - 1;
   while (i >= 0 && tab[i] != x) {
       i--;
   }
   present = i >= 0;
   if (present) {
       indice = i;
   }
   else {
       indice = nb;
   }
   ```

2. soit on parcourt les indices de 1 à nb. Dans ce cas, il faut regarder toutes les valeurs (on peut donc utiliser une boucle **Pour**) et conserver l'indice de la dernière valeur trouvée. On initialise donc naturellement `indice` à `nb+1` (l'élément n'est pas trouvé à priori).

   C

```
/* Déterminer l'indice de la première occurrence */
indice = nb;        /* indice hors limite */
for (i = 0; i < nb; i++) {
    if (tab[i] == x) {      /* élément trouvé */
        indice = i;
    }
}
present = indice < nb;
```

Cette deuxième solution n'est bien sûr à utiliser que dans les cas où il est coûteux de trouver le dernier élément (par exemple avec une chaîne de caractères à zéro terminal).

4.4 Indiquer l'indice, dans le tableau, de la n^{e} occurrence d'un entier donné.

Solution : Dans ce cas, il ne suffit pas de s'arrêter sur la première occurrence trouvée mais sur la n^{e}.

L'algorithme devient alors

```
Variable
    x: Entier           -- entier à chercher dans le tableau
    n: Entier           -- le numéro de l'occurrence à trouver
    i: Entier           -- parcourir les indices du tableau
    nb_x: Booléen       -- nb de fois où x a été trouvé dans tab[1..i-1]
Début
    i ← 1;
    nb_x ← 0;   -- pas d'occurrences trouvées dans tab[1..0]
    TantQue (i <= nb)           -- il reste des éléments à comparer
            Et (nb_x < n)       -- on n'a pas trouvé la bonne occurrence
    Faire
        { Invariant : nb_x = nb d'occurrences de x dans tab[1..i-1] }
        { Variant : nb - i + 1 }
        Si tab[i] = x Alors             -- une nouvelle occurrence
            nb_x ← nb_x + 1
        FinSi
        i ← i + 1
    FinTQ

    Résultat ← i - 1
Fin.
```

C

```
/* Déterminer l'indice de la nè occurrence de x */
i = 0;
nb_x = 0;   /* pas d'occurrences trouvées */
while (i < nb && nb_x < numero) {
    if (tab[i] == x) {
        nb_x++;
    }
    i++;
}
present = nb_x == numero;
indice = i - 1;     /* valide seulement si present est vrai */
```

Exercice 5 : Palindrome

Un *palindrome* est un groupe de mots qui peut se lire indifféremment de gauche à droite ou de droite à gauche en conservant le même sens.

Voici quelques exemples de palindromes : « radar », « kayak », « elle », « Ésope reste ici et se repose », « élu par cette crapule », etc.

Bien sûr, seuls les lettres sont prises en compte pour déterminer s'il s'agit d'un palindrome. Ainsi les espaces et les caractères de ponctuation sont ignorés.

Pour simplifier le traitement, on considère que la phrase ne contient que des lettres de même casse (que des minuscules par exemples), sans accents.

Écrire un programme qui indique si une phrase est un palindrome ou non sachant que la phrase ne contient que des lettre minuscules, des espaces et des caractères de ponctuations. On suppose de plus que la phrase se termine nécessairement par un point (le caractère « . »).

Solution :

```
R0 : Déterminer si une phrase est un palindrome
            phrase: in Chaîne
            palindrome: out Booléen
```

On suppose que l'on dispose de la phrase qui se termine par un point et on doit positionner une variable palindrome de type **Booléen**.

Principe : On utilise deux indices, l'un sur le début de la phrase et l'autre sur la fin. À chaque étape, on fait progresser les indices (en sautant les blancs, les ponctuations, etc.) et on vérifie que les deux caractères correspondant ont même valeur. Si oui, on continue en faisant de nouveau progresser les deux indices, si non on sait que ce n'est pas un palindrome.

```
R1 : Raffinage De « Détermine si une phrase est un palindrome »
  | Initialiser l'indice début          début: out Entier
  | Initialiser l'indice fin            fin: out Entier
  | palindrome ⟵ VRAI
  | Répéter
  |   | Avancer début sur le prochain caractère
  |   | Avancer fin sur le prochain caractère
  | JusquÀ (debut > fin)        -- tous les caractères ont été regardés
  |        Ou (phrase[début] <> phrase[fin])   -- pas un palindrome
  | palindrome ⟵ debut > fin

R2 : Raffinage De « Initialiser l'indice début »
  | Initialiser début avant la première position

R2 : Raffinage De « Initialiser l'indice fin »
  | Initialiser fin avec la première position
  | TantQue phrase[debut] <> '.' Faire
  |   | fin ⟵ fin + 1
  | FinTQ

R2 : Raffinage De « Avancer début sur le prochain caractère »
  | Répéter
  |   | début ⟵ début - 1
  | JusquÀ (debut > fin) Ou (phrase[début] est lettre)
```

```
R2 : Raffinage De « Avancer fin sur le prochain caractère »
  | Répéter
  |   | fin ← fin - 1
  | JusquÀ (fin < debut) Ou (phrase[fin] est lettre)
```

C

```c
/*****************************************************************
 *  Auteur   : Xavier Crégut <cregut@enseeiht.fr>
 *  Version  : 1.1
 *  Objectif : Déterminer si une phrase est un palindrome
 *****************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define MAX 80  /* longueur maximale de la phrase */
    /* Attention : On doit prendre MAX >= 1, car dans la phrase il y a
     * au moins un point.
     */

int main()
{
    char phrase[MAX+1]; /* La phrase à décoder (+ 1 pour '\0') */
    int debut, fin;     /* indices des deux caractères à comparer */
    bool palindrome;    /* est-ce que phrase est un palindrome  ?*/
    char touche;        /* un caractère lu au clavier */
    int nb;             /* nombre de caractères saisis */

    /* saisir la phrase (sans contrôle) */
    printf("Seules les lettres minuscules sont prises en compte !");
    printf("Donner une phrase qui se termine par un point : ");
    nb = 0;
    do {
        scanf("%c", &touche);
        phrase[nb] = touche;
        nb++;
    } while (touche != '.' && nb < MAX);
    if (touche != '.') {
        /* forcer le point */
        printf("La saisie a été tronquée !\n");
        phrase[nb-1] = '.';
    }
    phrase[nb] = '\0';
            /* pour marquer la fin de la chaîne et pouvoir utiliser
             * printf("%s", phrase).
             */

    /* Afficher la phrase tapée par l'utilisateur (avec le point final) */
    printf("phrase = %s\n", phrase);
```

```
    /* Initialisation l'indice debut */
    debut = -1;

    /* Initialiser l'indice fin */
    fin = 0;
    while (phrase[fin] != '.') {
        fin++;
    }

    do {
        /* avancer debut */
        do {
            debut++;
        } while (debut <= fin && (phrase[debut] < 'a' || phrase[debut] > 'z'));

        /* avancer fin */
        do {
            fin--;
        } while (fin >= debut && (phrase[fin] < 'a' || phrase[fin] > 'z'));

    } while (debut <= fin && phrase[debut] == phrase[fin]);

    palindrome = debut > fin;

    if (palindrome) {
        printf("C'est un palindrome.\n");
    }
    else {
        printf("Ce N'est PAS un palindrome.\n");
    }

    return EXIT_SUCCESS;
}
```