Chapitre 6

Variables structurées

1. Insuffisance des variables simples

Les variables qui ont été utilisées jusque là ne peuvent avoir qu'une valeur à un instant donné. Si une nouvelle valeur leur est affectée, la précédente est perdue. Ces variables sont des *variables simples*.

Certains problèmes ne peuvent pas être résolus uniquement avec des variables simples. Par exemple celui qui cherche à partager une somme d'argent entre un nombre de travailleurs, proportionnellement à leur temps de travail. Les durées de travail de chacun vont être saisies, sommées, et pour calculer les parts de chacun il faudra revenir sur les durées saisies, donc les avoir mémorisées. Le nombre de personnes n'est pas toujours le même d'une exécution à l'autre du programme. Il est donc impossible de traiter ce problème avec des variables simples, contenant chacune la durée de travail d'une personne. Pour résoudre ce type de problème, et bien d'autres, on définira des *variables de type tableau*.

Il arrive fréquemment que certaines données soient liées entre elles parce qu'elles caractérisent une même entité (par exemple les caractéristiques d'une date : jour, mois, année). Il peut être intéressant de matérialiser ce lien en faisant des différentes données des caractéristiques d'une seule variable. Pour établir ce type de lien on utilisera des *variables de type structure*.

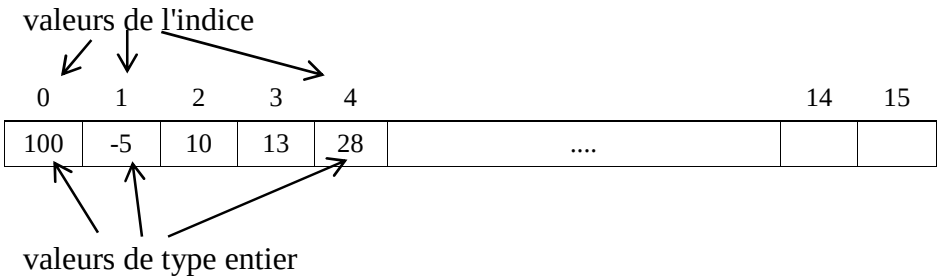
Ce chapitre présente ces deux types de variables structurées. Leur caractéristique commune est qu'elles peuvent avoir plusieurs valeurs à un instant donné. Cependant elles se définissent et s'utilisent de manières différentes.

2. Tableaux

2.1. Définitions

Un tableau à une dimension est une **variable structurée**, caractérisée par un **identificateur**, qui peut contenir à un instant donné **plusieurs** valeurs de **même type** (contrairement à une variable simple qui ne peut en contenir qu'une seule). On dit qu'un tableau a plusieurs éléments contenant chacun une valeur.

Chaque élément du tableau est repéré par un **indice**, en général un nombre entier, qui prend ses valeurs entre **deux bornes**.



Il faut noter que certains éléments peuvent être vides à un instant donné.

2.2. Déclaration

La déclaration d'un tableau comprend deux informations :

- les **bornes** : la borne inférieure et la borne supérieure que peut prendre son indice. Celles ci déterminent la réservation d'un certain nombre d'éléments pour le tableau
- le **type** de valeurs prises par ses éléments

<identificateur> : tableau [<indice_début> ... <indice_fin>] de <type>

exemple : `tab : tableau [1 .. 15] d'entiers`

Dans un langage de programmation, la déclaration a pour effet de réserver en mémoire le nombre de places voulues de la taille correspondant au type. Même si certains langages de programmation autorisent de redéfinir la taille d'un tableau en cours d'exécution du programme, un tableau est par nature une *variable statique,* à laquelle on donne une taille qui ne varie pas. On se tiendra à cet usage en algorithmique. C'est le contexte du programme qui doit permettre de déterminer la taille nécessaire. Si le problème nécessite de faire évoluer la taille, un tableau n'est pas le type de variable approprié. Il faut alors utiliser des *variables dynamiques*.

Il ne faut pas confondre *place déclarée* et *place occupée*. Un tableau peut très bien avoir été déclaré avec un intervalle d'indice de 1 à 100 et ne contenir que 20 valeurs. Lorsqu'on doit déclarer un tableau, c'est le contexte du problème qui guide sur la taille à déclarer. En effet il faut être sûr que le tableau sera assez grand pour contenir toutes les valeurs qui lui sont destinées, mais il ne faut pas non plus sur-dimensionner un tableau, ce qui conduit à un gaspillage de place en mémoire vive.

Chaque langage de programmation a ses règles pour l'échelle des indices. En langage C, par exemple, le premier élément est toujours à l'indice 0.

2.3. Accès à un élément de tableau

Chaque élément de tableau est repéré par une valeur de l'indice et s'écrit de la manière suivante :

<identificateur>[<indice>]

L'indice peut être :

- une constante : tab[2] vaut 10
- une variable : ind ← 3 ; tab[ind] vaut 13
- une expression : tab[ind + 1] vaut 28

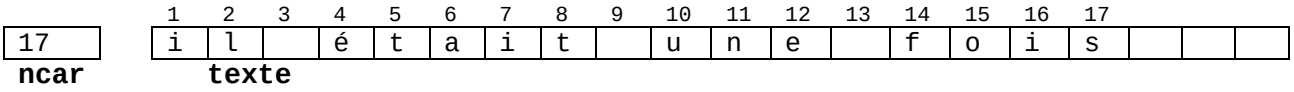
Règle générale : **un élément de tableau se comporte exactement comme une variable simple** et peut faire partie de n'importe quelle expression. On peut trouver un élément de tableau :

- dans une instruction de lecture lire(tab[4])
- dans une instruction d'écriture écrire(tab[4])
- à gauche d'une flèche d'affectation tab[5] ← 45
- dans une expression écrire(tab[5] * 10)
- à la place d'un paramètre effectif échanger(tab[1], tab[2])

2.4. Algorithmes pour remplir un tableau

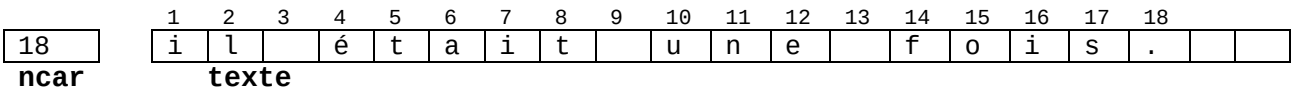
Mettre des valeurs dans un tableau (1) : nombre connu de valeurs

```
Algorithme remplir_tableau_1
  texte : tableau[1..100] de caractères                  /* le texte */
  ncar : entier                                  /* nombre de caractères */
  i    : entier                        /* variable de boucle et indice */
Début
  lire (ncar)
  pour i de 1 à nbcar faire
    lire(texte[i])
  finpour
Fin
```



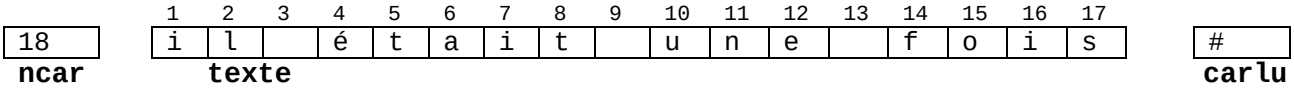
Mettre des valeurs dans un tableau (2) : dernier élément reconnu et rangé dans le tableau.

```
Algorithme remplir_tableau_2
  texte : tableau[1..100] de caractères                  /* le texte */
  ncar  : entier                                 /* nombre de caractères */
Début
  ncar ← 0
  répéter
    ncar ← ncar + 1
    lire(texte[ncar])
  jusqu'à texte[ncar] = '.'
Fin
```



Mettre des valeurs dans un tableau (3) : terminer par un élément fictif non rangé dans le tableau.

```
Algorithme remplir_tableau_3
  texte : tableau[1..100] de caractères                  /* le texte */
  ncar  : entier                                 /* nombre de caractères */
  carlu : caractère                                /* variable de saisie */
Début
  ncar ← 0
  lire (carlu)
  tant que carlu ≠ '#' faire
    ncar ← ncar + 1
    texte[ncar] ← carlu
    lire(carlu)
  fin tant que
Fin
```



2.5. Algorithmes pour parcourir et exploiter un tableau

Exploiter un tableau (1) : remplacer les 'i' par des 'x' : nombre d'éléments connus :

```
Algorithme parcourir_tableau_1
  texte : tableau[1..100] de caractères              /* le texte */
  ncar  : entier                              /* nombre de caractères */
  k     : entier                               /* variable de boucle */
Début
  lire(ncar) /* ou ncar a une valeur */
  pour k de 1 à ncar faire
    si texte[k] = 'i'
      alors texte[k] ← 'x'
    finsi
  finpour
Fin
```

Exploiter un tableau (2) : remplacer les 'i' par des 'x' : le dernier élément du tableau est connu et doit être traité :

```
Algorithme parcourir_tableau_2
  texte : tableau[1..100] de caractères              /* le texte */
  k     : entier                               /* variable de boucle */
Début
  k ← 0
  répéter
    k ← k + 1
    si texte[k] = 'i'
      alors texte[k] ← 'x'
    finsi
  jusqu'à texte[k] = '.'
Fin
```

Exploiter un tableau (3) : remplacer les 'i' par des 'x' : le dernier élément du tableau est connu et ne doit pas être traité :

```
Algorithme parcourir_tableau_2
  texte : tableau[1..100] de caractères              /* le texte */
  k     : entier                               /* variable de boucle */
Début
  k ← 1
  tant que texte[k] <> '.' faire
    si texte[k] = 'i'
      alors texte[k] ← 'x'
    finsi
    k ← k + 1
  fin tant que
Fin
```

2.6. Rechercher un élément dans un tableau

```
Algorithme rechercher_elt_tableau
  texte : tableau[1..100] de caractères              /* le texte */
  elt : caractères                            /* élément recherché */
  k     : entier                               /* variable de boucle */
Début
  lire(elt)
  k ← 1
  tant que texte[k] <> 'elt' et texte[k] <> '.' faire
    k ← k + 1
  fin tant que
  ecrire(elt, ‘ est en ‘, k, ‘ième place du tableau’)
Fin
```

2.7. Tableaux à plusieurs dimensions

Les plus utilisés sont les tableaux à deux dimensions, mais on peut très bien concevoir des tableaux à trois dimensions ou plus. Pour déclarer un tableau à *n* dimensions, il faut donner *n* intervalles de valeurs pour les indices, ainsi que le type des valeurs rangées dans le tableau.

Syntaxe : (bi pour borne inférieure**, bs** pour borne supérieure**)**

<identificateur> : tableau [<bi$_1$. .. bs$_1$>, <bi$_n$..bs$_n$>] de <type>

L'accès se fait avec les valeurs des *n* indices :

<identificateur>[<valeur i$_1$>, <valeur i$_n$>]

Les règles d'utilisation des éléments de tableau sont les mêmes que pour les tableaux à une dimension. Exemple :

	0	1	2	3	4	5
0	A	C	J	H	R	U
1	E	B	P	I	S	H
2	O	D	K	Q	T	G
3	N	M	L			

```
mat : tableau[0..3, 0..5] de caractères
```

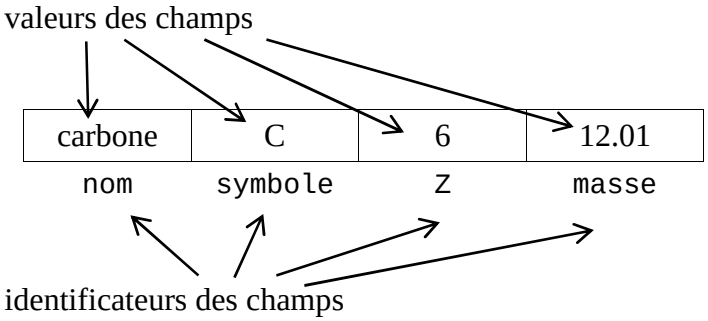
L'élément `mat[2, 3]` contient la valeur 'Q'

3. Structures

3.1.Définitions

Un autre type de variable structurée dite Structure est caractérisée par un *identificateur*, qui peut contenir, à un instant donné, **plusieurs valeurs de types différents.**

Chaque valeur est rangée dans un champ et repérée par un **identificateur de champ.**



3.2.Déclaration

Déclarer une variable structure revient à déclarer chaque champ sous la forme d'un identificateur et d'un type.

Syntaxe de la déclaration :

```
<identificateur> : structure
                       <identificateur champ> : <type>
                       ...

                       <identificateur champ> : <type>
                       <identificateur champ> : <type>
                   fin
```

Exemple :

```
elt :  structure
           nom : chaîne de caractères          /* nom de l'élément */
           symbole : chaîne de caractères      /* symbole chimique */
           Z : entier                          /* numéro atomique */
           masse : réel                        /* masse atomique */
       finstructure
```

Remarque : il est utile d'indiquer les rôles des champs (comme pour les variables simples)

3.3.Accès aux champs

Syntaxe :

<identificateur>.<identificateur champ>

point

Règle générale : un champ d'une variable structure se manipule comme une variable simple.

On peut le trouver :

- dans une instruction de lecture: `lire(elt.symbole)`
- dans une instruction d'écriture `écrire(elt.symbole)`
- à gauche d'une flèche d'affectation `elt.symbole ← 'Na'`
- dans une expression `m ← elt.masse * 4`
- à la place d'un paramètre réel dans un appel de procédure ou de fonction :
  `échanger(elt1.Z, elt2.Z)`

4. Définition de types

Il peut être pratique de définir des types puis d'utiliser ces noms dans des déclarations de variables.

Syntaxe pour définir un type :

<type> = <définition du type>

Les déclarations qui utilisent un type défini se font comme avec les types prédéfinis.

Exemples de définitions de type suivies de déclarations de variables

Définition de type :
```
t_tab = tableau [0..15] d'entiers
```
Déclaration de variable :
```
tab : t_tab
```

Définition de type :
```
t_matrice = tableau[0..3, 0..5] de caractères
```
Déclaration de variable :
```
mat : t_matrice
```

Définition de type
```
t_elt = structure
            nom : chaîne de caractères          /* nom de l'élément */
            symbole : chaîne de caractères      /* symbole chimique */
            Z : entier                          /* numéro atomique */
            masse : réel                        /* masse atomique */
      finstructure
```
Déclaration de variable :
```
elt : t_elt
```

Remarque : les noms de types suivent les règles des identificateurs. C'est par simple commodité, pour les distinguer des identificateurs des variables, que ceux des exemples commencent par `t_`.

5. Types complexes

On peut construire des types complexes en composant tableaux et structures :

- un élément de tableau peut être une structure,
- un champ de structure peut être un tableau

Il n'y a pas de limite à la composition, les deux paragraphes suivants donnent des exemples simples.

5.1. Tableaux de structures

Exemple : table des éléments chimiques

	Nom	symbole	Z	masse
0	Hydrogène	H	1	1
1	Hélium	He	2	4
2	Lithium	Li	3	6,9
3	Béryllium	Be	4	9
4	Bore	B	5	10,8
5	Carbone	C	6	12,01

Définitions des types :
```
t_elt = structure
            nom : chaîne de caractères          /* nom de l'élément */
            symbole : chaîne de caractères      /* symbole chimique */
            Z : entier                          /* numéro atomique */
            masse : réel                        /* masse atomique */
      finstructure
t_table = tableau[0..120] de t_elt
```

Déclaration :
`table_periodique : t_table`

Accès à un élément : `table[4].masse` contient la valeur 10,8
`table[2].symbole` contient la valeur 'Li'

5.2. Structure avec tableaux

Exemple : liste de températures avec nombre, minimum et maximum.

	temp	nb	tmin	tmax
0	15,8	5	6,3	21,5
1	10,4			
2	21,5			
3	6,3			
4	16,8			

Définitions des types :

```
t_temp = tableau[0..100] de réel
t_liste =   structure
                  temp : t_mesure  /* tableau des températures */
                  nb : entier      /* nombre de températures */
                  tmin : réel      /* température minimale */
                  tmax : réel      /* température maximale */
            finstructure
```

Déclaration :
`mesures : t_liste`

Accès à un élément
`mesures.nb` vaut 5
`mesures.temp[3]` vaut 6,3

6. Conclusion : structures de données

Les tableaux et les structures permettent de composer des structures complexes dans lesquelles peuvent être organisés des rangements des données qu'on appelle *structures de données*. D'autres types de données, en particulier des types de variables dynamiques, peuvent contribuer à construire des structures de données adaptées au problème à résoudre. Les structures de données et les méthodes algorithmiques qui les exploitent sont le plus souvent dépendantes les unes des autres.