Chapitre IV

Les tableaux et les chaînes de caractères

Sommaire

I. Les tableaux à une dimension : vecteurs

A. Notion de tableau

Jusqu'à présent, nous avons manipulé quelques données, dont les valeurs successives ont été rangées dans des cases mémoires auxquelles nous avons donné des noms, comme `a`, `x`, `indice`...

Si l'on doit manipuler un grand nombre de données, comme un fichier de clients, cette méthode devient impraticable. L'idée consiste alors, lorsque les données sont de nature similaire, à les ranger dans une structure de données linéaire appelée *tableau*, ou *array* en anglais. On donne un nom au groupe d'objets, et on attribue à chaque objet un numéro d'ordre.

Dans la plupart des langages, il est nécessaire de déclarer un tableau avant de l'utiliser, et en tout cas de l'initialiser. Cela permet au système d'exploitation de réserver de la place pour les éléments.

Opération	Traduction Pascal
déclaration	**var** t : **array**[<i>..<j>] **of** <**type**>;
affectation	t[<k>] := <valeur>
valeur	t[<k>]

La première instruction déclare un tableau nommé `t` dont les indices des éléments sont compris entre `i` et `j` (ce tableau comporte donc $j - i + 1$ éléments), de type `<type>`. `i` et `j` peuvent être deux entiers, ou deux éléments d'un type énuméré.

La deuxième instruction permet d'affecter la valeur `valeur` à la case numéro `k` du tableau `t`.

Enfin, la dernière ligne montre comment utiliser la valeur stockée dans la case d'indice `k` du tableau `t`.

B. Exploration d'un tableau

Un tableau est une *structure énumérative*[1], qu'on peut voir comme une fonction de l'ensemble *ordonné* des indices dans l'ensemble du type des données stockées. Un certain nombre de méthodes sont utilisées de façon récurrente pour manipuler une telle structure de donnée.

1. Exploration de l'ensemble des éléments du tableau

Si l'on doit réaliser un traitement sur l'ensemble des éléments du tableau, le plus simple est d'utiliser une boucle indexée. Voici par exemple une fonction calculant la somme des éléments d'un tableau d'entier, de type `tableau = array[a..b] of integer` :

```
function somme_tableau(t : tableau) : integer;

var i : integer;

begin
   somme_tableau := 0;
   for i := a to b do
      somme_tableau := somme_tableau + t[i]
end;
```

La "variable" `somme_tableau` joue le rôle classique d'*accumulateur*, elle est initialisée à zéro , et on accumule à chaque itération de la boucle l'élément rencontré dans le tableau.

2. Recherche d'éléments dans un tableau

Si l'on cherche à vérifier la présence d'un élément particulier dans un tableau, le traitement peut s'arrêter dès que l'élément a été rencontré, ou bien à défaut quand on arrive à la fin du tableau. On

[1] par opposition à une *structure itérative* ; on accède de manière immédiate à n'importe quel élément de la première, alors qu'on ne peut accéder aux éléments de la seconde qu'après avoir lu tous leurs prédécesseurs. Les deux exemples de base de structures itératives sont les *listes* et les *fichiers*.

ne peut alors plus utiliser une boucle itérative, qui ne peut être interrompue avant la fin. Dans ce cas, on utilise une boucle conditionnelle. Voici un exemple d'une fonction recherchant un élément `x` dans un tableau de type similaire à celui que nous venons d'utiliser :

```
function recherche_tableau(t : tableau; x : integer) : boolean;

var i : integer;

begin
   recherche_tableau := false;
   i := a;
   while (i <= b) and (not recherche_tableau) do
      if t[i] = x
      then recherche_tableau := true
      else i := i+1
end;
```

On voit qu'ici dans le corps de la boucle on compare l'élément indexé à la valeur recherchée. Si il y a coïncidence, on change la valeur de `recherche_tableau`, de sorte que la condition d'entrée dans la boucle devient `false`, sinon on incrémente l'indice de boucle.

Nous rencontrerons d'autres méthodes en TD. En particulier, si le tableau est *trié* (ce qui est une hypothèse assez courante dans la pratique), une méthode de recherche très efficace peut être employée : la *recherche dichotomique*. Nous étudierons cette méthode très importante en TD.

II. Tableaux à deux dimensions : matrices

Il est parfois utile de repérer des éléments d'une collection d'objets à l'aide de deux nombres. Par exemple, des coordonnées dans le plan. Les matrices vues en cours de mathématiques sont un exemple abstrait de tableaux à deux dimensions.

On peut déclarer une matrice de deux façons différentes :

- soit comme un tableau à une dimension dont les éléments sont des tableaux à une dimension :

  ```
  var t : array[a..b] of array[c..d] of <type>;
  ```

  auquel cas `t[i][j]` est de type `<type>`, et `t[i]` un tableau de type `array[c,d] of <type>`,
- soit comme un tableau à deux dimensions, de type

  ```
  var t : array[a..b,c..d] of <type>;
  ```

  auquel cas on accède à un élément de `t` sous la forme `t[i,j]`, `t[i]` étant encore un tableau.

La manière la plus simple d'effectuer un traitement sur l'ensemble des éléments d'une matrice consiste à inclure ce traitement dans une paire de boucles imbriquées. Voici par exemple une fonction calculant la somme des éléments d'un tableau de type `tableau = array[1..N,1..M] of integer` :

```
function somme_matrice(t : tableau) : integer;

var i,j : integer;

begin
   somme_matrice := 0;
   for i := 1 to N do
      for j := 1 to M do
         somme_matrice := somme_matrice + t[i,j]
end;
```

On peut imaginer d'autres façons d'explorer les cases d'une matrice, et nous en verrons des exemples en TD.

III. Les chaînes de caractères

Une chaîne de caractère peut être considérée comme un tableau de caractères. Et d'ailleurs, le Pascal standard ne disposait pas d'un type spécifique. Le Turbo-Pascal et ses successeurs, ainsi que l'immense majorité des langages de programmation, ont ajouté un type spécifique permettant de manipuler ces chaînes d'usage très courant, et certains langages se sont spécialisés dans ce type de traitement, comme le langage Perl.

Une déclaration de variable chaîne se présente sous la forme suivante :

```
var <nom> : string[<longueur>];
```

La variable `nom` est déclarée comme étant de type `string`, `longueur` indique la taille *maximale* de cette chaîne. La représentation en mémoire ajoute un octet pour stocker la longueur réelle de la chaîne. C'est ce nombre de caractère qui sera affiché par une instruction `write` ou `writeln`. Si cette longueur n'est pas précisée, une longueur par défaut de 255 caractères est appliquée.

Il y a peu d'opération directement réalisables sur les chaînes de caractère. Si `s` et `s'` sont des chaînes de type `string[N]`, voici ce qu'on peut faire :

Opérateur	Traduction Algorithmique/Pascal
accès à un caractère	`s[i]`
longueur effective	`length(s)`
concaténation	`s+s'`

Le caractère `s[i]` est le i-ème caractère de la chaîne (le premier a pour numéro 1). Le résultat est de type `char`, mais il peut être affecté à une chaîne.

Concaténer deux chaînes consiste à ajouter la deuxième au bout de la première. Si la chaîne résultante est trop longue, elle est simplement *tronquée*.

Par exemple, on aurait pu écrire le programme "Hello World" de la façon suivante :

```
program hello_chaine;

var s1,s2,s3 : string;

begin
   s1 := 'Hello';
   s2 := ', ';
   s3 := 'World';
   writeln(s1+s2+s3)
end.
```