

Module BASES DE L'ALGORITHMIQUE

TD 4

Les tableaux

Hervé Owsinski
Herve.Owsinski@uphf.fr

Semestre 1



2026-2027

Cours : Tableaux (statique)

Rappel : type scalaire

Un **type scalaire** correspond à une **donnée atomique** (non décomposable) : un entier, un réel, un booléen, un caractère....

Chaque numéro d'étudiants numEtu pourrait être stocké dans une variable de type scalaire Entier.

Lors d'un examen, on devrait alors créer 200 variables pour les 200 notes des étudiants. Lourd...

Rappel : type structuré

Un **type structuré** correspond à une **donnée composite** : constituée d'un assemblage de plusieurs données.

Lors d'un examen, on pourrait créer une seule variable **examen** de type structuré contenant les 200 notes.

Remarque : **certains langages** distinguent caractère et chaîne de caractères :

- un caractère est alors fourni en utilisant des guillemets simples :
'a', 'A', ' ' ... Type scalaire.
- une chaîne de caractères est fournie en utilisant des guillemets doubles :
"Bonjour". Type composé / structuré.

Définition 1 : Tableau (statique)

Un **tableau** est une structure de données permettant de regrouper une **collection finie et indexée de valeurs ayant toutes le même type**.

La **taille du tableau** est le nombre de cases qu'il peut contenir.

Lors de la déclaration d'un tableau, la **taille** et le **type des valeurs** qu'il contient sont définitivement fixés.

On schématise un tableau par un ensemble de "cases" auxquelles sont associées des **indices** (numéros de cases).

Cet exemple illustre un tableau appelé **T** pouvant contenir jusqu'à 10 valeurs entières :

<i>Indice</i>	0	1	2	3	4	5	6	7	8	9
<i>Valeur</i>	8	2	4	10	-7	0	5			

Utilisation des indices

<i>indice</i>	0	1	2	3	4	5	6	7	8	9
<i>valeur</i>	8	2	4	10	-7	0	5			

Les **indices** servent à repérer de façon non ambiguë les cases.

En algorithmique, les indices commencent **toujours à 0**.

L'accès à une valeur se fait avec l'utilisation des crochets [indice]. Ainsi :

- l'**indice 3 du tableau T** est associé à la **valeur 10** veut dire
- **T[3]** contient la valeur **10**
- **T[0]** contient la valeur **8**
- **T[4]** contient la valeur **-7**

On remarque que les cases d'indice 7, 8 et 9 ne contiennent pas de valeur : un tableau est créé avec un nombre maximum de cases, mais il n'y a aucune obligation à toutes les utiliser. Certaines peuvent donc être "vides", c'est à dire contenir un contenu n'ayant aucune signification.

Vital pour la suite

0	1	2	3	4	5	6	7	8	9
8	2	4	10	-7	0	5			

Il est **fondamental** de distinguer ces trois notions :

- **T** désigne le tableau (le conteneur);
- **i** désigne un indice de case (son numéro);
- **T[i]** désigne la valeur stockée dans la case i du tableau T.

Attention aux notations

Notez bien qu'en algorithmique, on note T[i] et pas juste [i] ou i[T].

Premier exemple

Mettre 20 à tout le monde

Soit un tableau notes de 10 cases contenant des entiers.

Les indices i sont donc pris dans $[0; 9]$.

i	0	1	2	3	4	5	6	7	8	9
note	8	2	4	10	-7	0	5			

Nous voudrions mettre 20 à tous les étudiants :

pour i de 0 à 9 **faire**

 notes[i] $\leftarrow$ 20;

fin pour

Résultat ?

i	0	1	2	3	4	5	6	7	8	9
note	20	20	20	20	20	20	20	20	20	20

Domage : tout le monde à 20, même les cases vides initialement.

Définition 2 : déclaration d'un tableau

Pour utiliser un tableau, il faut d'abord le **déclarer**.

On peut en particulier définir un **type tableau** en fonction des besoins de l'énoncé. Imaginons un tableau pouvant contenir au maximum 10 entiers : on doit définir un TYPE **avant de fournir le programme**.

La syntaxe à respecter sur notre pseudo-code est :

```
1
2  TYPE Tab = tableau[10] de Entier ; /* Notez le = comme avec CONST */
3  Programme exempletableau
4      VAR notes : Tab;           /* notes est un tableau de type Tab */
5      i : Entier;
6  debut
7      pour i de 0 à 9 faire
8          notes[i] ← 0;
9      fin pour
10     ...
11 fin
```

Définition 2bis : bonne pratique de déclaration d'un tableau

Si on décide ensuite d'avoir un tableau avec 100 cases, il faut modifier le code et transformer les 10 en 100 et les 9 en 99.

Cela est long et peut créer pas mal d'erreurs.

On utilise donc plutôt une **CONSTANTE** pour définir la taille du tableau.

On déclare bien entendu la **CONST** avant le **TYPE** qui va l'utiliser.

```
1  CONST MAX = 10;
2  TYPE Tab = tableau[MAX] de Entier ; /* Notez le = comme avec CONST */
3  Programme exempletableau
4      VAR notes : Tab;           /* notes est un tableau de type Tab */
5      i : Entier;
6  debut
7      pour i de 0 à (MAX - 1) faire
8          notes[i] ← 0;
9      fin pour
10     ...
11 fin
```

Signification des instructions en langage naturel

L1 : On déclare et définit une constant valant 10.

L2 : On déclare et définit un nouveau type nommé Tab qui représente des tableaux de MAX cases contenant des entiers.

L4 : On déclare une variable notes de type Tab.

L7 : Pour tous les indices du tableau.

L8 : On initialise le contenu de la case i du tableau notes à 0.

```
1  CONST MAX = 10;
2  TYPE Tab = tableau[MAX] de Entier ; /* Notez le = comme avec CONST */
3  Programme exempletableau
4      VAR notes : Tab;                /* notes est un tableau de type Tab */
5          i : Entier;
6  debut
7      pour i de 0 à (MAX - 1) faire
8          notes[i] ← 0;
9      fin pour
10     ...
11 fin
```

Exercices

Exo 1

Exo 1-A

Expliquer ce que produisent ces algorithmes et indiquer s'ils peuvent être simplifiés.

```
1  CONST MAX = 5 ;
2  TYPE TAB = tableau[MAX] de Entier ;
3
4  Programme exo1a
5      VAR i : Entier ;
6          t : TAB ;
7  début
8      i ← 0;
9      tant que (i ≤ 4) faire
10         t[i] ← i * i ;
11         i ← i + 1;
12      fin tant que
13      pour i de 0 à 4 pas 1 faire
14         Écrire(t[i]) ;
15      fin pour
16  fin
```

Exo 1

Exo 1-A

Expliquer ce que produisent ces algorithmes et indiquer s'ils peuvent être simplifiés.

```
1  CONST MAX = 5 ;
2  TYPE TAB = tableau[MAX] de Entier ; /* déclaration du type TAB */
3
4  Programme exo1a
5      VAR i : Entier ;
6          t : TAB ;
7  début
8      i ← 0;                                /* on commencera en case 0 */
9      tant que (i ≤ 4) faire                /* tant que l'indice est valide */
10         t[i] ← i * i ;                    /* mettre i2 dans la case i */
11         i ← i + 1;                        /* passer à la case suivante */
12      fin tant que
13      pour i de 0 à 4 pas 1 faire        /* pour chaque indice */
14         Écrire(t[i]) ;                  /* afficher le contenu de la case */
15      fin pour
16  fin
```

Exo 1

Exo 1-A Explications

On obtient un tableau dont chaque case d'indice i contient la valeur i^2 .

0	1	2	3	4
0	1	4	9	16

Cela revient donc à calculer certaines valeurs de la fonction suivante :

$$\begin{array}{rcl} f & : & I \rightarrow \mathbb{N} \\ & & x \mapsto x^2 \end{array}$$

Vous connaissez peut-être les compréhensions du type `list` en Python.

```
t = [i**2 for i in range(5)]
```

Pratique en Python mais on ne fournit pas d'équivalent en pseudo-code : le but est justement de savoir le faire seul.

Exo 1-A - les améliorations

Trois "améliorations" possibles :

- ➊ Généraliser le programme en utilisant la constante MAX dans le programme ;
- ➋ Remplacer la boucle **Tant Que** par une boucle **Pour** puisqu'on peut prédire sur quelles cases on veut agir ;
- ➌ Optionnel : gérer traitement et affichage dans la même boucle : cela évite d'avoir à parcourir deux fois le tableau. Avec 10 cases, ça va, avec 10 milliards, c'est plus délicat. Désavantage néanmoins : on mélange traitement des données et affichage des données.

Exo 1

Exo 1-A - le nouveau programme

```
1  CONST MAX = 5 ;
2  TYPE TAB = tableau[MAX] de Entier ;
3
4  Programme exo1a
5      VAR i : Entier ;
6          t : TAB ;
7  début
8      pour i de 0 à (MAX - 1) pas 1 faire
9          t[i] ← i * i ;
10         Écrire(t[i]) ;
11      fin pour
12  fin
```

Exo 1

Exo 1-B

Expliquer ce que produisent ces algorithmes et indiquer s'ils peuvent être simplifiés.

```
1  CONST MAX = 5 ;
2  TYPE TAB = tableau[MAX] de Entier ;
3
4  Programme exo1b
5      VAR i : Entier ;
6          t : TAB ;
7  début
8      t[0] ← 1 ;
9      i ← 1;
10     tant que (i ≤ 4) faire
11         t[i] ← t[i-1] + 2 ;
12         i ← i + 1;
13     fin tant que
14     pour i de 0 à 4 pas 1 faire
15         Écrire(t[i]) ;
16     fin pour
17 fin
```

Exo 1

Exo 1-B : explications

Expliquer ce que produisent ces algorithmes et indiquer s'ils peuvent être simplifiés.

```
1  CONST MAX = 5 ;
2  TYPE TAB = tableau[MAX] de Entier ; /* déclaration du type TAB */
3
4  Programme exo1b
5      VAR i : Entier ;
6          t : TAB ;          /* déclaration du tableau t de type TAB */
7  début
8      t[0] ← 1 ;              /* on place 1 dans la case 0      */
9      i ← 1 ;                 /* on passe à la suivante  */
10     tant que (i ≤ 4) faire /* tant que i est un indice valide */
11         t[i] ← t[i-1] + 2 ; /* stocke +2 par rapport à la précédente */
12         i ← i + 1 ;        /* on passe à la suivante */
13     fin tant que
14     pour i de 0 à 4 pas 1 faire
15         Écrire(t[i]) ;
16     fin pour
17 fin
```

Exo 1

Exo 1-B : explications

On obtient un tableau contenant les **MAX** premiers entiers impairs.

0	1	2	3	4
1	3	5	7	9

Puisque qu'un nombre impair peut s'écrire $n = 2 \times k + 1$ avec $k \in \mathbb{N}$, notre algorithme calcule certaines valeurs de la fonction suivante :

$$\begin{aligned} f : I &\rightarrow \mathbb{N}^* \\ x &\mapsto 2 \times x + 1 \end{aligned}$$

Exo 1

Exo 1-B : améliorations

```
1  CONST MAX = 5 ;
2  TYPE TAB = tableau[MAX] de Entier ; /* déclaration du type TAB */
3
4  Programme exo1b
5      VAR i : Entier ;
6          t : TAB ;          /* déclaration du tableau t de type TAB */
7  début
8      pour i de 0 à (MAX-1) pas 1 faire /* pour toutes les cases */
9          t[i] ← 2*i + 1 ; /* stocke l'entier impair 2i+1 */
10         Écrire(t[i]) ;
11     fin pour
12 fin
```

On peut notamment directement calculer la valeur voulue plutôt que d'aller lire le contenu de la case précédente.

Exo 2

Exo 2

Expliquer ce que fait l'algorithme suivant.

```
1  CONST MAX = 7 ;  
2  TYPE TAB = tableau[MAX] de Entier ;  
3  
4  Programme exo2  
5      VAR i : Entier ;  
6          t : TAB ;  
7  début  
8      t[0] ← 1 ;  
9      t[1] ← 1 ;  
10     pour i de 2 à 6 pas 1 faire  
11         t[i] ← t[i-1] + t[i-2] ;  
12     Écrire(t[i]) ;  
13     fin pour  
14 fin
```

Exo 2

Exo 2 - explications

Expliquer ce que fait l'algorithme suivant.

```
1  CONST MAX = 7 ;
2  TYPE TAB = tableau[MAX] de Entier ; /* déclaration du type TAB */
3
4  Programme exo2
5      VAR i : Entier ;
6          t : TAB ;          /* déclaration d'un vrai tableau de type TAB */
7  début
8      t[0] ← 1 ;              /* On place 1 en case 0 */
9      t[1] ← 1 ;              /* On place 1 en case 1 */
10     pour i de 2 à 6 pas 1 faire /* Pour les indices valides  $\geq 2$  */
11         t[i] ← t[i-1] + t[i-2] ; /* on stocke la somme des 2 précédentes
12         Écrire(t[i]) ;          /* on affiche la nouvelle valeur */
13     fin pour
14 fin
```

Exo 2

Exo 2 - Suite de Fibonacci

La suite de Fibonacci est définie ainsi :

$$F(0) = 0,$$

$$F(1) = 1 \text{ puis}$$

$$F(n) = F(n-1) + F(n-2) \text{ pour } n \geq 2, \text{ ce qui est bien la somme des deux précédentes.}$$

On obtient ainsi un tableau contenant **presque** les premiers termes de la suite de Fibonacci : on a mis de côté le premier terme $F(0) = 0$. Donc, attention : il y aura un décalage de 1 case pour obtenir la valeur voulue.

i	0	1	2	3	4	5	6	...	i
valeur stockée	1	1	2	3	5	8	13	...	$t[i-1] + t[i-2]$
qui correspond à	$F(1)$	$F(2)$	$F(3)$	$F(4)$	$F(5)$	$F(6)$	$F(7)$	...	$F(i+1)$

Conclusion : on obtient un tableau dont chaque case d'indice i contient le terme $F(i+1)$ de la suite de Fibonacci.

$$\begin{aligned} f : I &\rightarrow \mathbb{N} \\ x &\mapsto F(x+1) \end{aligned}$$

Cours (la suite) : stockage du nombre de cases utilisées

Deuxième exemple de parcours

Rajouter 2 à ceux qui ont eu au moins 10

<i>i</i>	0	1	2	3	4	5	6	7	8	9
<i>valeur</i>	8	2	4	10	-7	0	5			

```
pour i de 0 à 9 faire  
  si (notes[i] ≥ 10) {alors  
    notes[i] ← notes[i] + 2;  
  fin si  
fin pour
```

<i>i</i>	0	1	2	3	4	5	6	7	8	9
<i>valeur</i>	8	2	4	12	-7	0	5	??	??	??

Ca ne fonctionne pas : que veut dire "compare vide à deux" ou "rajouter 2 à vide"
si la case est vide ? **SOLUTION** : mémoriser qu'à partir de 7 les cases sont vides.

Mémoriser la taille réelle utilisée (méthode générale) : A CONNAITRE

<i>i</i>	0	1	2	3	4	5	6	7	8	9
<i>valeur</i>	8	2	4	10	-7	0	5	↓		

Mémorisons dans une variable **n** le **nombre de cases successives réellement utilisées**.
n correspond également à **la prochaine case à utiliser pour ajouter une valeur**.

Sur l'exemple : **n = 7** car

- la prochaine case à remplir est la case d'indice 7 ;
- et il y a 7 valeurs dans les cases d'indice 0 à 6.

On remarquera qu'il faut nécessairement $n \leq \text{MAX}$.

Principe d'utilisation :

- On initialise n à 0 : les cases sont toutes vides au départ.
- Lorsqu'on veut rajouter une valeur, on l'affecte dans la case d'indice n puis on incrémente n.
- On utilise n plutôt que MAX pour savoir jusqu'à quelle case aller lors d'un parcours.

Mémoriser la taille réelle utilisée (exemple)

<i>i</i>	0	1	2	3	4	5	6	7	8	9
<i>valeur</i>	↓									

```
n ← 0;
t[n] ← 8 ; n ← n + 1 ; /* remplit la case 0 puis n passe à 1 */
t[n] ← 2 ; n ← n + 1 ; /* remplit la case 1 puis n passe à 2 */
t[n] ← 4 ; n ← n + 1 ; /* remplit la case 2 puis n passe à 3 */
t[n] ← 10 ; n ← n + 1 ; /* remplit la case 3 puis n passe à 4 */
t[n] ← -7 ; n ← n + 1 ; /* remplit la case 4 puis n passe à 5 */
t[n] ← 0 ; n ← n + 1 ; /* remplit la case 5 puis n passe à 6 */
t[n] ← 5 ; n ← n + 1 ; /* remplit la case 6 puis n passe à 7 */
pour i de 0 à (n-1) faire
    si (notes[i] ≥ 10) alors
        notes[i] ← notes[i] + 2;
    fin si
fin pour
```

0	1	2	3	4	5	6	7	8	9
8	2	4	12	-7	0	5	↓		

Cela fonctionne car nous connaissons l'indice de la dernière case réellement remplie.

Erreur courante : nombre de cases et indice maximal

0	1	2	3	4	5	6	7	8	9
8	2	4	10	-7	0	5			

Ce tableau de 10 cases possède un indice maximal de ... 9 ! Attention.

Taille des tableaux

- Sur ce module, on ne se donne pas d'opérateur permettant de savoir si une case contient une valeur ou pas.
- On manipule donc un tableau en **mémorisant son nombre de cases utilisées**. On nomme souvent cette variable n .
- Lorsque l'on ne connaît pas, le nombre exact de valeurs qui seront stockées dans le tableau, on **surdimensionne** celui-ci pour être sûr d'avoir assez de place.

Exercices (suite)

Exo 3 - Contexte

On considère une liste de notes d'un DS (noté sur 20) d'un groupe d'étudiants de L1 que l'on souhaite saisir au clavier et stocker dans un tableau, pour pouvoir faire différents traitements.

Les notes sont saisies et ajoutées une à une dans le tableau.

On ne connaît pas à l'avance le nombre d'étudiants dans le groupe (ce nombre peut être différent en fonction des groupes).

Exo 3 - Questions

- 1 Définir un type, appelé TAB, tableau permettant de stocker au plus 40 notes.
- 2 Écrire un programme qui va : demander à l'utilisateur le nombre de notes à saisir, puis faire la saisie des notes (dans un tableau) avec une boucle pour. Enfin, le programme affichera les notes saisies pour vérification, avec une boucle tant que.
- 3 On veut maintenant calculer la moyenne du groupe. Pour cela, on doit dans un premier temps faire la somme des notes du tableau. Compléter le programme précédent pour faire ce calcul (avec une boucle jusqu'à) et afficher la somme des notes et la moyenne.
- 4 Proposer une autre version en utilisant une boucle faire ... tant que.
- 5 Écrire un autre algorithme qui, après avoir fait la saisie des notes, calcule et affiche le nombre d'étudiants ayant eu la moyenne au DS (en utilisant une boucle tant que).
- 6 Réécrire l'algorithme précédent en utilisant une boucle pour.

Exo 3.1

- ❶ Définir un type, appelé TAB, tableau permettant de stocker au plus 40 notes.

```
1  CONST MAX = 40 ;  
2  TYPE TAB = tableau[MAX] de Entier ;  
3  
4  ...
```

- 2 Écrire un programme qui va : demander à l'utilisateur le nombre de notes à saisir, puis faire la saisie des notes (dans un tableau) avec une boucle pour. Enfin, le programme affichera les notes saisies pour vérification, avec une boucle tant que.

Exo 3.2

```
1  CONST MAX = 40 ;
2  TYPE TAB = tableau[MAX] de Réel ;
3
4  Programme exo3
5      VAR i, n : Entier ;
6          t : TAB ;
7  début
8      faire
9          Ecrire("Combien de notes voulez-vous ? [0 à ", MAX, "]" ) ;
10         Lire(n) ; /* n contient la taille finale */
11     tant que ( n < 0 OU n > MAX);
12
13     pour i de 0 à (n-1) pas 1 faire
14         Ecrire("Entrer la note n°", i + 1, " : " ) ;
15         Lire(t[i]);          /* note i dans la case d'indice i */
16     fin pour
17
18     i ← 0;
19     tant que ( i < n) faire
20         Ecrire(t[i]) ;      /* on affiche le contenu de la case i */
21         i ← i + 1 ;        /* on passe à la case suivante */
22     fin tant que
23
24     ...
```

Exo 3.3

- 3 On veut maintenant calculer la moyenne du groupe. Pour cela, on doit dans un premier temps faire la somme des notes du tableau. Compléter le programme précédent pour faire ce calcul (avec une boucle jusqu'à) et afficher la somme des notes et la moyenne.

Exo 3.3

```
4  Programme exo3
5      VAR i, n : Entier ;
6          t : TAB ;
7          somme, moyenne : Réel ;
```

```
24      i ← 0;
25      somme ← 0;
26      répéter
27          somme ← somme + t[i] ; /* on ajoute la note n°i      */
28          i ← i + 1 ;          /* on passe à la case suivante */
29      jusqu'à (i = n) ;        /* i = n est hors tableau   */
30
31      moyenne ← somme / n ;
32      Ecrire("La somme des notes est ", somme) ;
33      Ecrire("La moyenne du groupe est ", moyenne) ;
34
35      fin
```

Remarques sur cette question :

- N'oubliez pas d'initialiser i et somme avant la boucle ;
- Pensez à incrémenter le i après chaque tour de boucle Jusqu'à ;
- Pensez à mettre votre point-virgule après la condition du jusqu'à ;
- Attention : le calcul de la moyenne devra se faire après le calcul de la somme. Puisqu'on connaît le nombre de notes, nous aurions pu faire autrement, mais le résultat serait-il clair ?

Exo 3.4

- 4 Proposer une autre version en utilisant une boucle faire ... tant que.

Exo 3.4

```
24  i ← 0;
25  somme ← 0;
26  faire
27      somme ← somme + t[i] ; /* on ajoute la note n°i */
28      i ← i + 1 ;          /* on passe à la case suivante */
29  tant que ( i < n ) ;
30
31  moyenne ← somme / n ;
32  Ecrire('La moyenne du groupe est ', moyenne) ;
33  Ecrire("La somme des notes est ", somme) ;
34
35  fin
```

Exo 3.5

- 5 Écrire un autre algorithme qui, après avoir fait la saisie des notes, calcule et affiche le nombre d'étudiants ayant eu la moyenne au DS (en utilisant une boucle tant que).

Exo 3.5

```
1  Programme exo3q5
2      VAR i, n, nbr : Entier ;
3      t : TAB ;
4
5  début
6      i ← 0; nbr ← 0;
7
8      faire
9          Ecrire("Combien de notes ? [0 à ", MAX, "]" ) ;
10         Lire(n) ;
11     tant que ( n < 0 OU n > MAX);
12
13     tant que ( i < n) faire
14         Écrire("Entrer la note n°", i + 1, " : " ) ;
15         Lire(t[i]);
16         si (t[i] ≥ 10) alors
17             nbr ← nbr + 1 ;
18         fin si
19         i ← i + 1 ;
20     fin tant que
21
22     Ecrire(nbr, " personnes ont eu la moyenne à ce DS" ) ;
23 fin
```

Exo 3.6

- 6 Réécrire l'algorithme précédent en utilisant une boucle pour.

Exo 3.6

```
1  Programme exo3q6
2      VAR i, n, nbr : Entier ;
3          t : TAB ;
4
5  début
6      nbr ← 0;
7
8      faire
9          Ecrire("Combien de notes ? [0 à ", MAX, "]" ) ;
10         Lire(n) ;
11     tant que ( n < 0 OU n > MAX);
12
13     pour i de 0 à (n-1) pas 1 faire
14         Écrire("Entrer la note n°", i + 1, " : " ) ;
15         Lire(t[i]);
16         si (t[i] ≥ 10) alors
17             nbr ← nbr + 1 ;
18         fin si
19     fin pour
20
21     Ecrire(nbr, " étudiant(s) ont eu la moyenne à ce DS" ) ;
22 fin
```

Reprenons le contexte de l'exercice 12 de la fiche de TD3.

- ❶ Définir un type, appelé TEMP, permettant de stocker les températures de la serre chaque heure au cours d'une journée (par exemple, la température à 3h du matin est de 15,2 degrés).
- ❷ Ecrire un programme qui saisit les températures durant les N premières heures de la journée, les place dans un tableau de type TEMP, puis calcule et affiche la moyenne des températures sur ces N premières heures. Le nombre d'heures N sera saisi en début de programme.
- ❸ Compléter le programme de la question 2 en ajoutant la saisie d'une valeur de température idéale et un parcours du tableau pour identifier les heures où l'écart entre la température et la température idéale est supérieur à 5 degrés sur les N premières heures. Le programme affichera le nombre d'heures concernés.

Exo 4.1

- 1 Définir un type, appelé TEMP, permettant de stocker les températures de la serre chaque heure au cours d'une journée (par exemple, la température à 3h du matin est de 15,2 degrés).

```
1 TYPE TEMP = tableau[24] de Réel ;
```

On peut ainsi stocker les températures des heures allant de 0 à 23.

Exo 4.2

- 2 Ecrire un programme qui saisit les températures durant les N premières heures de la journée, les place dans un tableau de type TEMP, puis calcule et affiche la moyenne des températures sur ces N premières heures. Le **nombre d'heures N** sera saisi en début de programme.

Exo 4.2

```
1  TYPE TEMP = tableau[24] de Réel ;
2
3  Programme exo4
4      VAR N, heure : Entier ;
5          t : TEMP ;
6          total, moyenne : Réel ;
7
8  début
9      moyenne ← 0; total ← 0;
10     répéter
11         Ecrire("Combien d'heures voulez-vous fournir (1 à 24) : ") ;
12         Lire(N) ;
13     jusqu'à (N ≥ 1 ET N ≤ 24);
14
15     pour heure de 0 à (N-1) pas 1 faire
16         Ecrire("Entrer la température à ", heure, " h : ") ; Lire(t[heure]);
17         total ← total + t[heure] ;
18     fin pour
19
20     moyenne ← total / N ;
21     Ecrire("Température moyenne : ", moyenne) ;
22 fin
```

Exo 4.3

- 3 Compléter le programme de la question 2 en ajoutant la saisie d'une valeur de température idéale et un parcours du tableau pour identifier les heures où l'écart entre la température et la température idéale est supérieur à 5 degrés sur les N premières heures.
Le programme affichera le nombre d'heures concernés.

Exo 4.3

```
1  Rajout à l'en-tête :
2      VAR nbr : Entier ;
3          ideale : Réel ;
4  ...
5      nbr ← 0 ;
6      Ecrire("Quelle est la température idéale ? ") ;
7      Lire(ideale) ;
8      pour heure de 0 à (N-1) pas 1 faire
9          si ( ((t[heure] - ideale) > 5) OU ((t[heure] - ideale) < -5) ) faire
10             nbr ← nbr + 1 ;
11             Ecrire("Ecart de plus de 5°C à : ", heure);
12         fin si
13     fin pour
14     Ecrire("Nombre d'heures avec un écart de plus de 5°C : ", nbr) ;
15 fin
```

Exo 5

On souhaite compter le nombre de voyelles et le nombre de consonnes dans une phrase, représentée par une suite de caractères. Ces caractères sont saisis au clavier, un à un, et sont stockés dans un tableau. Pour simplifier on suppose que la phrase se termine nécessairement par un '.', et ce caractère '.' ne sera pas considéré comme un caractère de la phrase (il ne sera pas ajouter dans le tableau).

- 1 Définir un type, appelé `TAB_C`, tableau permettant de stocker au plus 50 caractères.
- 2 Écrire un algorithme de saisie des caractères et de remplissage du tableau fonctionnant sur le principe suivant : le programme répète la saisie d'un caractère (et son ajout dans le tableau), tant que l'utilisateur n'entre pas le caractère '.'. L'algorithme doit aussi compter le nombre de caractères saisis, et afficher ce nombre à la fin.
- 3 Ajouter, à la suite de l'algorithme précédent, les instructions permettant de compter le nombre d'occurrences (le nombre d'apparitions dans le tableau) du caractère 'e', et d'afficher ce nombre.
- 4 Comment procéder pour compter le nombre de 'e' et le nombre de 'u' ?
- 5 Proposer un autre algorithme capable de résoudre le problème initial, c'est-à-dire calculer et afficher le nombre de voyelles et le nombre de consonnes.

Exo 5.1

- 1 Définir un type, appelé TAB_C, tableau permettant de stocker au plus 50 caractères.

```
1  CONST MAX = 50 ;  
2  TYPE TAB_C = tableau[MAX] de Caractère ;
```

Exo 5.2

- 2 Écrire un algorithme de saisie des caractères et de remplissage du tableau fonctionnant sur le principe suivant : le programme répète la saisie d'un caractère (et son ajout dans le tableau), tant que l'utilisateur n'entre pas le caractère '.'. L'algorithme doit aussi compter le nombre de caractères saisis, et afficher ce nombre à la fin.

Exo 5.2

```
1  CONST MAX = 50 ;
2  TYPE TAB_C = tableau[MAX] de Caractère ;
3
4  Programme exo5
5      VAR n : Entier ;
6          phrase : TAB_C ;
7          c : Caractere ;
8  début
9      n ← 0; /* n nombre de caractères réellement saisis */
10     Ecrire("Entrer un par un les caractères de votre phrase, . pour finir");
11     Ecrire("Attention : 50 caractères au maximum.");
12     faire
13         Lire(c) ;
14         si (c ≠ '.') alors
15             phrase[n] ← c;      /* on stocke c dans la case n */
16             n ← n + 1 ;          /* on passe à la case suivante */
17         fin si
18     tant que (c ≠ '.' ET n < 50) ;
19
20     Ecrire("Vous avez saisi ", n, " caracteres.");
21     /* la suite ici */
22 fin
```

Exo 5.3

- ➊ Ajouter, **à la suite** de l'algorithme précédent, les instructions permettant de compter le nombre d'occurrences (le nombre d'apparitions dans le tableau) du caractère 'e', et d'afficher ce nombre.

A la suite, donc on complète avec une nouvelle boucle.

Exo 5.3

```
1  Programme exo5
```

```
2      VAR n, nbr, i : Entier ;  
3          phrase : TAB_C ;  
4          c : Caractere ;
```

```
21     ...
```

```
22     nbr ← 0;
```

```
23     pour i de 0 à (n-1) pas 1 faire
```

```
24         si (phrase[i] = 'e') faire
```

```
25             nbr ← nbr + 1 ;
```

```
26         fin si
```

```
27     fin pour
```

```
28     Ecrire("Il y a ", nbr, " e dans la phrase.") ;
```

```
29 fin
```

Exo 5.4

- ④ Comment procéder pour compter le nombre de 'e' et le nombre de 'u' ?

Notons que cette phrase est ambiguë / ambigüe.

Veut-on compter les e d'un côté et le u de l'autre ? Ou cumuler les deux ?

Je fais le choix du cumul, plus intéressant d'un point de vue pédagogique.

Exo 5.4

Il suffit d'utiliser l'opérateur logique OU ou alors réaliser des if successifs :

Méthode 1 : avec un OU

```
si ( (phrase[i] = 'e') OU (phrase[i] = 'u') ) faire  
    nbr ← nbr + 1 ;  
fin si
```

Méthode 2 : avec deux if successifs (non imbriqués).

```
si (phrase[i] = 'e') faire  
    nbr ← nbr + 1 ;  
fin si  
si (phrase[i] = 'u') faire  
    nbr ← nbr + 1 ;  
fin si
```

Exo 5.5

- 5 Proposer un autre algorithme capable de résoudre le problème initial, c'est-à-dire calculer et afficher le nombre de voyelles et le nombre de consonnes.

L'énoncé semble supposer qu'on ne gère pas les majuscules puisqu'il parle de 'e' et de 'u' mais jamais de majuscules.

Question difficile car on ne possède pas d'opérateur d'appartenance (le **in** de Python). Il faut donc tester **une par une** toutes les voyelles et toutes les consonnes ! Pas compliqué mais 26 tests !

Solution alternative : utiliser la complémentarité voyelle ou consonne sur l'ensemble des lettres 'a' à 'z'.

Vérifier si **c** est entre 'a' et 'z'.

Si vrai, on teste si **c** est une voyelle ("aeuio").

Si faux, **c** est nécessairement une consonne appartenant à l'ensemble "a..z".

Exo 5.5

```
1  Programme exo5
2      VAR n, nbrC, nbrV, i : Entier ;
3          p : TAB_C ; /* p plus court que phrase */
4          c : Caractere ;
5          est_voyelle : Booléen ;
6  début
7      ...
8      nbrC ← 0; nbrV ← 0;
9
10     pour i de 0 à (n-1) pas 1 faire
11         si (p[i] ≥ 'a' ET p[i] ≤ 'z') alors
12             est_voyelle ← p[i] = 'a' OU p[i] = 'e' OU
13                 p[i] = 'y' OU p[i] = 'u' OU
14                 p[i] = 'i' OU p[i] = 'o';
15             si (est_voyelle) alors
16                 nbrV ← nbrV + 1 ;
17             sinon
18                 nbrC ← nbrC + 1 ;
19             fin si
20         fin si
21     fin pour
22     Ecrire("Il y a ", nbrV, " voyelles dans la phrase.") ;
23     Ecrire("Il y a ", nbrC, " consonnes dans la phrase.") ;
24 fin
```

Bilan

Pour la semaine prochaine

Avec Moodle :

- ❶ Réaliser **sérieusement** le QCM final de cette partie.
- ❷ Réaliser vos fiches de cours : bientôt le DS ;
 - Affectation et séquentialité ;
 - Alternative ;
 - Boucles ;
 - Tableaux ;

- Savoir définir un type tableau ;
- Savoir qu'un tableau de n cases possède des indices de 0 à $(n-1)$;
- Distinguer clairement t , i et $t[i]$;
- Savoir lire un tableau case par case à l'aide d'une boucle quelconque ;
- ALGO : recherche du maximum, du minimum dans un tableau t ;
- ALGO : recherche la moyenne des valeurs dans un tableau t ;