Chapitre 4

Données complexes : les tableaux

4.1 Limites des variables simples

Dans les exercices sur les boucles, nous avons envisagé le calcul d'une moyenne. Nous allons imaginer un contexte plus précis pour ce petit programme :

- les nombres représentent des notes obtenues par des étudiants lors d'une épreuve
- il y a un maximum de 100 étudiants
- les notes ne sont jamais négatives

Nous ajouterons une petite requête, qui va pourtant nous compliquer la tâche : le programme devra déterminer combien d'étudiants obtiennent une note supérieure à la moyenne.

Déclaration des variables

Nous travaillons avec des variables de type flottant, pour donner une plus grande précision à la notation. La variable `iNbreSupMoyenne` nous servira à compter le nombre d'étudiants ayant plus que la moyenne. Nous allons introduire un nouveau type de variable : les `Indices`. Ce sont des variantes des entiers qui nous servirons dans un premier temps à contrôler l'exécution d'une boucle. La variable indice `iI`, une simple variante d'un entier, permettra de contrôler une boucle dans la phase finale du programme.

```
Variable Entière iNbreEtudiants
Variable flottante fNote
Variables flottantes fSomme, fMoyenne
Variable indice iI
Variable entière iNbreSupMoyenne
```

Phase initiale

Ce programme ne nécessite pas de préparation particulière. J'ai ajouté arbitrairement un petit message pour placer une instruction avant la boucle.

```
Début du traitement
   /* 1. Accueil et explications */
   Afficher "Statistiques d'une interrogation" ~|
```

©2010-2011 - JACQUES THOORENS - HTTP://WWW.THOORENS.NET

Calcul de la moyenne

Le calcul de la moyenne se réalise au moyen d'une simple boucle. Il faut bien entendu initialiser à 0 le nombre d'étudiants déjà traités et l'accumulateur servant à calculer la somme [1]. Pour arrêter la boucle, nous utiliserons une *sentinelle*, c'est-à-dire une valeur arbitrairement choisie pour signaler que la dernière donnée a été entrée. Dans ce contexte, nous pouvons utiliser n'importe quelle valeur négative. Comme le test utilise une donnée fournie par l'utilisateur, il ne peut se placer qu'en fin de boucle. Le corps de la boucle contiendra un message (reprenant la consigne d'arrêt), la lecture d'une donnée, l'addition de cette dernière dans la somme et l'incrémentation du nombre d'étudiants. Lors de l'entrée de la sentinelle, cette donnée négative ne doit pas être ajoutée aux autres et le nombre d'étudiants ne doit pas être incrémenté.

```
/* 2. Entrée des données et calcul de la somme */
Initialisation
   fSomme <- 0.
   iNbreEtudiants <- 0
Répéter
   Afficher "Note suivante (négative pour arrêter):"
   Moteur Lire fNote
   Si fNote>=0. alors
      fSomme<-fSomme+fNote
      Incrémenter iNbreEtudiants
   Fin de si
Boucler tant que fNote>=0.
```

À la fin de la boucle, `fSomme` contient le total de tous les points encodés et `iNbreEtudiants` le nombre de notes saisies. Nous sommes prêts à calculer la moyenne.

Comptage des étudiants obtenant plus de points que la moyenne

Le calcul de la moyenne est apparemment trivial. Cependant, avant de faire la division, nous devons nous assurer que le diviseur (`iNbreEtudiants`) n'est pas nul. Comme l'utilisateur peut avoir fourni une sentinelle lors de la première itération, nous devons placer un test. Toute la fin du programme n'a de sens que si au moins une note a été encodée. Nous choisissons cette fois une boucle avec compteur, puisque nous savons combien de notes nous avons.

Comme nous avons utilisé une même variable pour encoder toutes les données, nous n'avons aucun moyen de comparer chacune des notes d'étudiants à la moyenne pour effectuer le comptage des meilleurs résultats. Nous sommes obligés de reposer la question à l'utilisateur (les deux instructions en gras). Ce dernier risque de ne pas apprécier.

```
/* 3. Comparaison des notes et de la moyenne */
Si iNbreEtudiants>0 alors
   fMoyenne<-fSomme/iNbreEtudiants
   Initialisation
      iNbreSupMoyenne<-0
   Compter avec iI de 1 à iNbreEtudiants
      Afficher "Relire note suivante "
      Lire fNote
      Si fNote>fMoyenne alors
```

1. La notion d'accumulateur a été vue dans le précédent chapitre.

©2010-2011 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

```
            Incrémenter iNbreSupMoyenne
         Fin de si
      Fin de compter
      Afficher iNbreSupMoyenne & " étudiants ont une note supérieure à "
         & fMoyenne & " (moyenne de la classe)"
      Fin de ligne
Fin de si
Fin du traitement
```

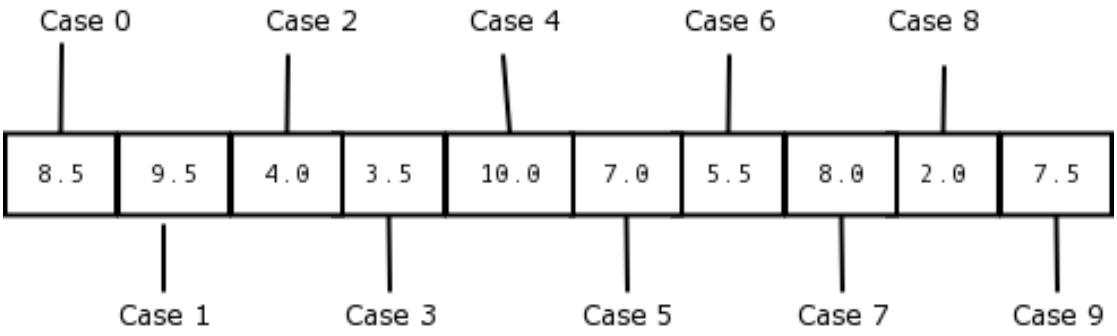
Le traitement que nous avons envisagé n'est pas gérable autrement. En effet, lors de la première boucle, nous ne connaissons pas encore la moyenne [2]. Il n'est donc pas possible de déterminer combien de notes lui sont supérieures.

4.2 Définition et emploi d'un *tableau*

Dans la section précédente, nous avons vu que la nécessité de retaper les données était particulièrement inconfortable pour l'utilisateur. Il faudrait en fait que nous disposions d'autant de variables qu'il y a d'itérations. Malheureusement, ces variables auraient des noms différents, ce qui rendrait impossible l'utilisation des mêmes instructions. Il nous faut donc autre chose. Nous avons besoin d'une variable complexe – on dit aussi *structurée* – capable de contenir plusieurs valeurs.

Les *tableaux* sont des variables complexes qui permettent de ranger un certain nombre de valeurs dans des zones mémoire contiguës, accessibles au moyen de deux informations :

- un nom de variable, commun à toutes les zones mémoire ;
- un indice, propre à chaque zone.



Nos instructions internes à la boucle pourront alors s'écrire très simplement : il suffira d'employer le nom du tableau et un indice différent à chaque itération. L'utilisation d'une variable d'indice permettra d'assurer cette différence.

Un tableau est donc une variable multiple, comportant autant d'éléments que prévus lors de sa déclaration, chacun d'entre eux étant désigné par un indice. Dans presque tous les langages, les indices sont consécutifs [3]. Selon le langage, le programmeur possède plus ou moins de

2. Le calcul d'une moyenne provisoire ne présente aucun intérêt. Ce n'est qu'après avoir encodé toutes les données qu'on connaît la moyenne. Soient les trois notes 10, 5, 0. Selon l'ordre d'encodage, les moyennes provisoires seront très différentes :

Notes	Moyenne partielle	Notes	Moyenne partielle	Notes	Moyenne
10	10	10,5	7,5	10,5,0	5
0	0	0,10	5	0,10,5	5
5	5	5,0	2,5	5,0,10	5

3. La seule exception qui m'est connue est PHP, réputé pour sa gestion particulièrement souple des tableaux.

©2010-2011 - JACQUES THOORENS - HTTP://WWW.THOORENS.NET

liberté dans la définition des limites de ces indices. Certains langages permettent de définir non seulement la taille du tableau, mais l'indice initial. On peut donc avoir un tableau d'indices 0 à 9, un tableau d'indices 1 à 10, voire un tableau d'indice 21 à 30. Pascal autorise même d'utiliser des caractères comme indices (tableau d'indices 'a' à 'z'). Nous allons voir que C se contente de spécifier la dimension du tableau et commence toujours à l'indice 0. Notre pseudo-code suivra cette pratique.

4.2.1 Déclaration d'un tableau

La déclaration d'un tableau spécifie trois choses :
- son nom
- le type des éléments qui le constituent
- sa dimension (ou les limites des indices si c'est possible)

Remarquons que la déclaration de type concerne tous les éléments du tableau, ce qui entraîne que tous les éléments partagent un seul type. Connaissant la taille du tableau (N), nous pouvons spécifier que les indices seront compris entre 0 et N-1. Examinons quelques exemples.

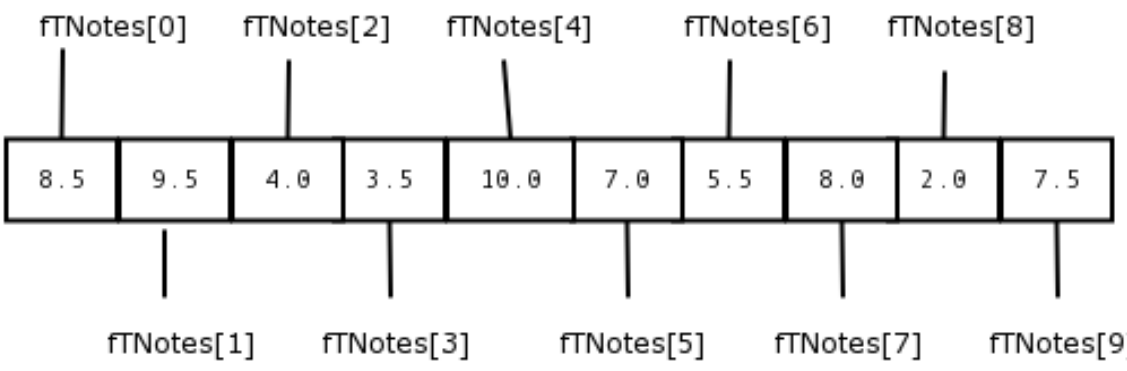
```
Tableau Entiers iT Taille 10
Tableau Flottants fRacine Taille 15
Tableau Caractères cLettres Taille 20
```

Le dernier exemple correspond à un tableau de caractères (à ne pas confondre avec une chaîne de caractères).

Il est très important de noter que la déclaration du tableau figure dans la partie déclarative d'un programme et qu'il n'est donc pas possible de spécifier la taille du tableau au moment de l'exécution. La plupart des langages de programmation impose cette contrainte. Quelques rares langages, au nombre desquels BASIC, offrent un moyen de gérer dynamiquement la taille des tableaux. C, comme Pascal, obligent à l'utilisation d'astuces pour obtenir une telle souplesse. Dans le cadre de ce cours, on veillera donc à créer des tableaux suffisamment grands pour contenir toutes les données nécessaires à un type de problème.

4.2.2 Accès à un élément

Pour accéder à un tableau, il est indispensable de connaître le nom du tableau et l'indice correspondant à la valeur recherchée. La plupart des langages utilisent la même convention, qui diffère uniquement par l'emploi de parenthèses ou de crochets. Les langages C et Pascal emploient des crochets :



©2010-2011 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

```
/* placer une valeur dans le premier élément (indice 0)*/
iT[0] Devient 15
/* afficher le contenu du deuxième élément (indice 1) */
Afficher iT[1]
/* additionner le contenu des deux premiers éléments dans le 3me */
iT[2] <- iT[0] + iT[1]
```

Remarque 1 : nous avons vu que la déclaration d'un tableau de taille N permettait d'utiliser des indices de 0 à N- 1. Malheureusement, rien n'interdit au programmeur de dépasser les limites du tableau. Comme dans le programme suivant :

```
Tableau Entiers iTab Taille(5)
Début du Traitement
  Afficher Entier iTab[5]
  iTab[100] <- 25
Fin du Traitement
```

Seuls les éléments 0 à 4 sont définis. Lors de l'impression de `iTab[5]`, le programme va calculer l'emplacement du sixième élément, qui n'existe pas, et tentera de lire les données contenues en fait dans les mémoires adjacentes au cinquième élément. Le comportement du programme va dépendre du langage employé :

- soit le langage autorise une telle lecture, mais les données affichées seront totalement dépourvues de sens.
- soit il l'interdit et le programme s'arrête avec une erreur d'exécution.

Le langage C choisit la première attitude. Le langage Pascal autorise les deux, selon la manière dont on a réglé les options du compilateur. Notons que la deuxième technique entraîne un contrôle de portée lors de chaque accès à un élément de tableau, ce qui se paie par une diminution des performances.

L'instruction d'affectation de valeur dans le 101ème élément, si elle est acceptée, risque par contre de provoquer un plantage du programme. A cet endroit, il peut y avoir autre chose, par exemple, un morceau du programme ou du système d'exploitation, qu'il serait vraiment malvenu de vouloir modifier[4].

Remarque 2 : en langage C, il n'est pas possible de manipuler les tableaux comme des ensembles de données et par exemple de copier le contenu d'un tableau dans l'autre en une seule instruction. Notre pseudo code, si on veut le compiler et l'exécuter, impose la même restriction.

```
Tableau Entiers iT1 Taille(5)
Tableau Entiers iT2 Taille(5)

Début du Traitement
   /* instruction refusée par le compilateur !!! */
   iT1 Devient iT2
Fin du Traitement
```

4. On peut constater que sous Linux les dépassements de taille d'un tableau entraîne souvent une erreur dénommée *segmentation fault*. C'est qu'en effet initialement conçu comme un système multi-utilisateurs, Linux n'autorise pas qu'on joue dans la zone mémoire réservée au voisin. Ces erreurs ne sont cependant pas systématiquement sanctionnées.

©2010-2011 - JACQUES THOORENS - HTTP://WWW.THOORENS.NET

4.2.3 Initialisation d'un tableau

Il existe de nombreux cas où les tableaux manipulés doivent contenir des valeurs connues au moment de l'écriture du programme. Certains langages utilisent pour cela des tableaux de constantes. Nous nous inspirerons de la technique du C pour initialiser le tableau dès sa déclaration [5].

```
Tableau Entiers iValeurE Taille 5
    <* = { 2, 4, 6 , 8, 10} *>
Tableau Entiers iPremier Taille 100
    <* ={ 1, 2, 3, 5, 7, 11, 13, 17} *>
```

Les cinq éléments de `iValeurE` contiendront respectivement les valeurs 2, 4, 6, 8 et 10. Pour ce qui concerne `iPremier`, seuls les huit premiers éléments contiendront des valeurs connues, les 92 éléments suivants contiendront des valeurs aléatoires. Il est à noter que la pratique d'initialiser partiellement un tableau, bien qu'autorisée par le compilateur, ne semble pas compatible avec une programmation saine et rigoureuse. Elle est donc à éviter, sauf cas d'espèce.

Remarquons également que l'initialisation, à la mode du C, n'est possible que lors de la déclaration du tableau. Si on veut réinitialiser un tableau en cours de programme, deux techniques sont disponibles :

- écrire autant d'instructions d'affectation qu'il y a d'éléments dans le tableau ;
- utiliser un autre tableau, préalablement initialisé, et le recopier dans le tableau à modifier au moyen d'une boucle (tableaux à une dimension) ou plusieurs (tableaux à plusieurs dimensions)

4.2.4 Parcours d'un tableau

Le parcours complet d'un tableau se fait généralement avec une instruction `Compter Avec`, puisque les limites en sont connues. Supposons qu'on désire remplir les éléments d'un tableau de 10 éléments avec les carrés de 10 premiers nombres entiers. Nos connaissances des boucles nous autorisent à réaliser cette opération sans trop de mal.

```
Tableau Entiers iValeurE Taille 10
Variable Entière iI

Début Traitement
   Init
   Compter Avec iI De 0 A 9
      iValeurE[iI] <- (iI+1) * (iI+1)
   Fin Compter
Fin du Traitement
```

4.3 Nouvelle version de notre programme de moyenne

Nous allons ici prévoir une déclaration de constante (`MAX`) de manière à pouvoir gérer élégamment la taille du tableau. Nous remplacerons également la variable `fNote` par un tableau `fTNotes`. La taille arbitraire de 100 correspond à une limite jamais atteinte dans le cas d'une classe du secondaire.

5. Cette faculté est un ajout tardif incorporé au langage laf. Il faut noter que le contenu du contexte marqué par les opérateurs d'insertion directe n'est pas analysé.

©2010-2011 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

```
Constante MAX = 100
Variable Entière iNbEtud
Tableau flottants fTNotes Taille MAX
Variables flottantes fNote, fSomme, fMoyenne
Variable indice iI
Variable entière iNbreSupMoyenne

Début du traitement
 /* 1. Accueil et explications */
 Afficher "Statistiques sur les résultats d'une interrogation" ~|

 /* 2. Entrée des données et calcul de la somme */
 Initialisation
    fSomme <- 0.
    iNbEtud <- 0
 Répéter
    Afficher "Note suivante (négative pour arrêter):"
    Moteur Lire fNote
    Si fNote >=0. alors
       fTNotes[iNbEtud]<- fNote
       fSomme<-fSomme+fNote
       Incrémenter iNbEtud
    Fin de si
 Boucler tant que fNote >= 0.
```

La seule innovation consiste à mémoriser la note lue dans une case du tableau.

La fin du programme est conforme à nos attentes et ne se différencie de la première version que par la disparition du deuxième encodage et l'emploi de `fTNotes[iI]` à la place de `fNote`.

```
/* 3. Comparaison des notes et de la moyenne */
Si iNbEtud>0 alors
   fMoyenne<-fSomme/iNbEtud
   Initialisation
      iNbreSupMoyenne<-0
   Compter avec iI de 0 à iNbEtud-1
      Si fTNotes[iI]>fMoyenne alors
         Incrémenter iNbreSupMoyenne
      Fin de si
   Fin de compter
   Afficher iNbreSupMoyenne & " étudiants ont une note supérieure à "
      & fMoyenne & " (moyenne de la classe)"
   Fin de ligne
Fin de si
Fin du traitement
```

©2010-2011 - JACQUES THOORENS - HTTP://WWW.THOORENS.NET

4.4 Tableaux à 2 ou n dimensions

Les tableaux que nous avons abordés jusqu'ici n'avaient qu'une seule dimension. On peut prévoir des tableaux à deux dimensions, à trois voire plus [6]. On se contentera d'énumérer les différentes dimensions séparées par deux points.

```
Tableau entiers iAnnee Taille 12:31
```

Pour accéder à un tableau à deux dimensions, on utilisera deux indices. Le premier désigne conventionnellement la *ligne* et le second la *colonne*.

```
/* initialisation de la première cellule de la première ligne */
iAnnee[0,0] <- 10
/* initialisation de la cinquième cellule de la septième ligne */
iAnnee[6,4] <- 45
```

On notera qu'il est ennuyeux de devoir opérer mentalement ces conversions (le premier correspond à 0, le septième correspond à 6), mais il faut s'y faire [7].

À titre d'exemple, voici un tableau à deux dimensions représentant un jeu de combat naval et le programme qui remplit le tableau avec des 0 avant la partie.

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

```
Tableau entiers iMer Taille 10:10
Variables indices iLigne,iColonne
Début du traitement
  /* Remplir lignes 0 à 9 */
  Init
  Compter avec iLigne de 0 à 9
    /* Remplir cases 0 à 9 */
    Init
    Compter avec iColonne de 0 à 9
      iMer[iLigne,iColonne]<-0
    Fin de compter
  Fin de compter
Fin du traitement
```

4.5 Recherche d'un élément dans un tableau

4.5.1 Version classique

Considérons un tableau contenant une certain nombre de valeurs quelconques :

65	78	12	15	69	48	47	51	34	7

Nous allons imaginer qu'un utilisateur décide de rechercher la position d'un nombre dans ce tableau. Nous supposerons que chaque valeur est unique. La technique est relativement simple et s'apparente à la recherche d'une personne dans un bâtiment :

6. Le compilateur laf2c limite à 5 le nombre de dimensions. Cela devrait suffire à nos exercices. Les tableaux à plusieurs dimensions sont de grand consommateurs de mémoire. Un tableau d'entiers à six dimensions dont les indices iraient de 0 à 9 nécessiterait déjà 4 Mo de mémoire. En pratique, je n'ai jamais personnellement utilisé de tableaux à plus de trois dimensions.

7. J'ai l'habitude de citer l'informaticien qui, devant compter jusqu'à dix, répond sans sourciller : « 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ».

©2010-2011 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

– examiner une à une les cases du tableau/les pièces du bâtiment
– s'arrêter quand on a trouvé le nombre/la personne ou quand on a exploré tout le tableau/-toutes les pièces.

On notera qu'une fois le travail répétitif terminé, il convient de se demander pourquoi la boucle s'est achevée : parce qu'on a trouvé ou parce qu'on est sûr de ne plus pouvoir trouver ?

```
Constante MAX = 10
Tableau Entiers iTableau Taille MAX
<* = {65,78,12,15,69,48,47,51,34,7}*>
Variable Entière iNombre
Variable Indice iI

Début du traitement

   /* 1. Demande du nombre à trouver */
   /* 2. Boucle de recherche */
   /* 3. Analyse du résultat */

Fin du traitement
```

Demande du nombre à trouver

Cette première partie du programme n'appelle aucune remarque.

```
Début du traitement
   /* 1. Demande du nombre à trouver */
   Afficher "Nombre à rechercher : "
   Lire iNombre
```

Recherche proprement dite

Nous nous trouvons en présence d'une boucle pratiquement inconsistante : le corps ne contient que la seule instruction d'incrémentation. Tout se situe dans l'évaluation de la condition. Nous devons considérer s'il reste un élément du tableau à évaluer et si cet élément est différent de la valeur recherchée. Dans ce cas, il ne nous reste qu'à nous préparer à l'examen de l'élément suivant.

```
/* 2. Boucle de recherche */
   Initialisation
      iI<-0
   Tant que iI<MAX  ET iNombre<>iTab[iI]
   Répéter
      Moteur Incrémenter iI
   Fin de boucle
```

La condition de maintient de la boucle doit bien s'exprimer comme nous l'avons fait. Il est particulièrement intéressant d'examiner la condition examinée lorsque `iI` vaut 10 :

```
10 < MAX ET iNombre <> iTableau[10]
```

©2010-2011 - JACQUES THOORENS - HTTP://WWW.THOORENS.NET

D'une manière évidente, la première partie du test est fausse (10 n'est pas inférieur à 10). Par contre, la seconde partie de la condition n'a pas de sens puisque la case 10 n'existe pas. Nous jouons avec une particularité du `ET` dans plusieurs langages : ne pas évaluer la seconde partie d'une conjonction lorsque le premier conjoint [8] est faux (en effet la valeur du second ne peut pas rendre vraie la conjonction). Un tel `ET` est appelé *ET conditionnel*. De la même manière, un `OU` sera dit conditionnel s'il n'examine pas le second disjoint lorsque le premier est vrai. L'incapacité d'utiliser des connecteurs logiques conditionnels rend l'écriture de certains algorithmes plus difficiles dans certains langages [9].

Le caractère conditionnel de `ET` fait que l'expression suivante de la condition risquerait de planter le programme au cas où le nombre ne serait pas contenu dans le tableau :

```
iNombre <> iTableau[iI] ET iI < MAX
```

Quand `iI` vaut 10, l'évaluation du premier conjoint n'a pas de sens. On constate donc, avec étonnement, que le ET n'est pas commutatif !

Analyse du résultat

Ce n'est pas tout de sortir d'une boucle : encore faut-il savoir pourquoi on en est sorti ? Dans notre exemple, est-on sorti après avoir trouvé ou faute d'avoir d'autres cases à explorer ? Ces deux questions entraînent deux formulations du test :

```
iNombre == iTableau[iI] et iI == MAX
```

Ces deux formulations ne sont pas équivalentes : en effet, la première n'a de sens que si `iI` est inférieur à 10, ce dont nous sommes précisément en train de douter. C'est donc le test sur l'indice que nous allons utiliser pour réaliser nos affichages.

```
/* 3. Analyse du résultat */
   Si iI<MAX alors
      Long texte "Nombre trouvé en position " & iI FinTexte
   sinon
      Afficher "Le nombre n'est pas dans le tableau"
   Fin de si
   Fin de ligne
Fin du traitement
```

4.5.2 Version optimisée

Nous avons vu que la première partie de la condition (vérifier que l'indice `iI` désigne toujours un élément existant du tableau) est absolument nécessaire au fonctionnement du programme : il s'agit d'empêcher l'examen d'une case inexistante. Malheureusement ce test n'est vraiment utile que lorsqu'on a parcouru tout le tableau. La tentation est grande de s'en passer. Si nous étions sûr de rencontrer la valeur cherchée dans le tableau, nous pourrions laisser tomber cette première comparaison sans remords. Pour être sûr de trouver la valeur, le plus sûr moyen est de l'y placer nous même. Il suffit d'agrandir le tableau d'un élément et de placer la valeur à trouver dans cet emplacement. Nous obtenons ainsi une version optimisée du programme (il faut compter une diminution d'environ 40% du temps d'exécution du programme).

8. On nomme parfois *conjoint* le terme d'une conjonction logique (deux conditions unies par ET). Le terme *disjoint* s'emploie à propos des termes d'une disjonctions (deux conditions unies par OU).

9. Dans le but de ne pas alourdir l'exposé, je ne donnerai pas de solution adaptée à un tel contexte. Il faut utiliser une variable de type logique et lui affecter une valeur dans le corps de la boucle. Une telle solution fait perdre beaucoup d'élégance et de naturel à l'algorithme.

©2010-2011 - JACQUES THOORENS - HTTP://WWW.THOORENS.NET

Le léger gaspillage de mémoire est largement compensé par la simplification du code et par l'accroissement des performances.

65	78	12	15	69	48	47	51	34	7	**X**

```
Constante MAX = 10
Variable indice iI
Variable entière iNombre
Tableau Entiers iTab Taille MAX+1
<* ={65,78,12,15,69,48,47,51,34,7} *>
Début du traitement
   /* 1. Demande du nombre à trouver */
   Afficher "Nombre à rechercher : "
   Lire iNombre

   /* 2. Boucle de recherche */
   Initialisation
      iI<-0
      /* placer la valeur à trouver en dernière case */
      iTab[MAX]<-iNombre
   Tant que iTab[iI]<>iNombre
   Répéter
      Moteur Incrémenter iI
   Fin de boucle

   /* 3. Analyse du résultat */
   Si iI<MAX alors
      Long texte "Nombre trouvé en position " & iI FinTexte
   sinon
      Afficher "Le nombre n'est pas dans le tableau"
   Fin de si
   Fin de ligne
Fin du traitement
```

Nous voyons que le test final reste capable de distinguer les cas où la recherche s'est arrêtée avant l'examen de la dernière véritable valeur (on a trouvé) de celui où cette valeur finale, que nous avons placée avant de pénétrer dans la boucle, a été découverte (ce qui correspond à un échec de la recherche).

4.6 Bref retour sur les conditions

Il arrive fréquemment que nous soyons amenés à comparer le contenu d'une valeur entière ou d'un caractère à plusieurs valeurs constantes. Le cas typique est un menu proposé à l'utilisateur.

Le programme acceptera un choix basé sur le numéro de l'option ou sur le première lettre (majuscule ou minuscule). On voit que l'on obtiendra une cascade de `si` avec des conditions complexes.

©2010-2011 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

```
    Choisissez votre option :

            1 - Explications
            2 - Saisie
            3 - Calculs
            4 - Afficher les résultats
            9 - Quitter
```

```
Si cChoix == '1' OU cChoix == 'E' OU cChoix == 'e' Alors
    ....
Sinon
   Si cChoix == '2' OU cChoix == 'S' OU cChoix == 'S' Alors
    ...
   Sinon
      Si ...
          ...
      Fin de si
   Fin de si
Fin de si
```

Une solution plus élégante consiste à examiner globalement les conditions.

```
Examen si cChoix
   Vaut '1','E','e'
        ...
   Fin de cas
   Vaut '2','S','s'
        ...
   Fin de cas
   ...
   Défaut
     /* quoi faire pour un cas non prévu */
   Fin de cas
Fin de examen
```

La présence du groupe `Défaut` ... `Fin de cas` permet de traiter tous les cas non prévus. Ce groupe doit impérativement se placer le dernier. Il correspond à ce qu'on appelle habituellement une *comportement par défaut,* c'est-à-dire en l'absence de consigne explicite.

Voici un exemple de programme complet illustrant le menu vu plus haut. Il reste à programmer les différentes options...

```
Variable caractère cOption
Début du traitement
   Initialisation
      /* ... */
   Répéter
      Effacer écran
      Afficher "\t1. Explications\n"
      Afficher "\t2. Saisie\n"
      Afficher "\t3. Calcul\n"
```

©2010-2011 -JACQUES THOORENS - HTTP://WWW.THOORENS.NET

```
      Afficher "\t4. Afficher les résultats\n"
      Afficher "\t9. Quitter\n"
      Moteur Lire cOption
      Examen si cOption
         Vaut '1','E','e'
            Afficher "Explication en cours..."
            Attente
         Fin de cas
         Vaut '2'
            Afficher "Saisie en ..."
            Attente
         Fin de cas
         Vaut '3','C','c'
            Afficher "Calcul en cours..."
            Attente
         Fin de cas
         Vaut '4','A','a'
            Afficher "Affichage en cours..."
            Attente
         Fin de cas
         Vaut '9','Q','q'
            cOption <- '9'
         Fin de cas
         Défaut
            Afficher "Option non valide"
            Attente
         Fin de cas
      Fin de examen
   Boucler tant que cOption<>'9'
Fin du traitement
```

On remarque que le cas où l'option Quitter a été choisie, on modifie la variable `cChoix` de manière à simplifier le test final. Des millions de programmes ont été bâtis sur ce modèle.

Exercices

IV-1. Remplir un tableau de 10 flottants avec dix nombres saisis au clavier.

IV-2. Remplir un tableau de 10 entiers avec les carrés des dix premiers nombres.

IV-3. Rechercher et afficher le plus grand élément d'un tableau d'entiers.

IV-4. Rechercher et afficher les deux plus petits éléments d'un tableau.

IV-5. Il a été exposé au cours qu'une instruction `Devient` ne s'applique pas à un tableau entier (limite imposée par le substrat C de notre pseudo-code). Écrire un programme qui recopie un tableau de 5 nombres entiers dans un autre tableau similaire. Astuce : s'il n'est pas possible de copier toutes les données d'un coup, il faut bien se résoudre à les copier une à une.

IV-6. Un programme contient deux tableaux, dont le premier est rempli avec des valeurs quelconques. Écrire un programme qui place dans les éléments du deuxième tableau la somme cumulée des éléments correspondants du premier tableau. Par exemple, si le premier tableau contient 5, 3 et 2, le second contiendra 5, 8 et 10. (Vous pouvez utiliser ces données pour initialiser votre tableau).

©2010-2011 - JACQUES THOORENS - HTTP://WWW.THOORENS.NET

5	3	2	1	10	11	3	5	4	2
5	8	10	11	21	32	35	40	44	46

IV-7. Soit un tableau d'entiers. Écrire un programme qui inverse les éléments du tableau (le premier devient le dernier, le deuxième l'avant-dernier etc..). La modification doit se faire dans un seul tableau (on ne peut pas utiliser de tableau intermédiaire) [10]. Par exemple, le tableau suivant :

10	20	30	40	50	60	70	80	90	100

deviendra :

100	90	80	70	60	50	40	30	20	10

IV-8. Réaliser un tableau carré qui contienne la table de multiplication de notre enfance. Le programme devra le remplir et l'afficher.

	1	*2*	*4*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
1	1	2	3	4	5	6	7	8	9	*10*
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

IV-9. Relire le programme suivant et signaler les erreurs qu'il contient :

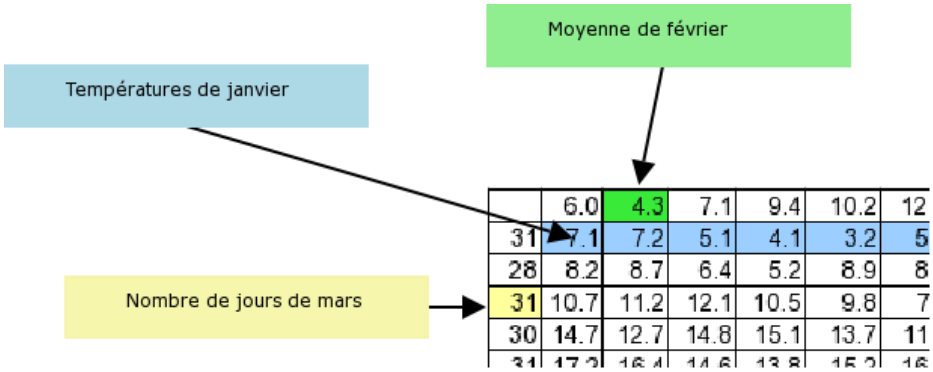
```
Tableau Entiers iMesures Taille 10
Tableau Entiers iCopie Taille 20
Début du Traitement
/* Initialiser le tableau de mesures */
        Init
        Compter Avec iCompteur de 1 A 10
                Lire iMesures[iCompteur]
        Fin Compter
/* En faire une copie */
        Init
        Compter Avec iCompteur de 0 A 20
                iCopie[iCompteur] <- iMesures[iCompteur]
                Incrementer iCompteur
        Fin de Compter
/* Copie plus rapide */
        iCopie <- iMesures
Fin du Traitement
```

10. Dans une situation réelle, l'utilisation d'un tableau intermédiaire n'est toujours pas possible, surtout si le tableau est de grande taille.

©2010-2011 - JACQUES THOORENS - HTTP://WWW.THOORENS.NET

IV-10. Créer un programme chargé de gérer les températures relevées dans une ville. On utilisera notamment un tableau à deux dimensions reprenant les températures par mois (lignes) et par jour (colonnes). On pourrait utiliser la première colonne (d'indice 0) pour noter le nombre de jours du mois et la première ligne pour les températures moyennes des douze mois, de manière à adresser les cases d'une manière plus intuitive : le 18 juillet correspondrait à la paire [7,18].

– créer le tableau
– remplir les cases du tableau avec des températures aléatoires (en nombres flottants, il faut trouver une astuce)
– calculer les moyennes mensuelles
– citer le mois ayant la température la plus haute et celui ayant la plus basse.



IV-11. Comment faudrait-il modifier le programme précédent pour gérer les températures relevées dans 20 villes du pays ?

IV-12. Créer un tableau comportant 20 mots français et 20 mots d'une autre langue de votre choix. Créez un petit programme qui réalise une interrogation portant sur cinq mots français tirés au hasard que l'utilisateur est prié de traduire. On affichera le score.

```
Interrogation d'anglais.
Veuillez répondre à cinq questions.
Comment dit-on être en anglais ? to be
C'est exact
Comment dit-on chanter en anglais ? to sing
C'est exact
Comment dit-on devenir en anglais ? to devenir
devenir se dit en anglais to become
Comment dit-on bleu en anglais ? blue
C'est exact
Comment dit-on hors en anglais ? out
C'est exact
Vous avez obtenu un score de 4 sur 5. C'est bien.
```

IV-13. Améliorer le programme précédent pour qu'il ne fournisse jamais deux questions identiques lors d'une interrogation.

IV-14. Modifier le programme précédent pour pouvoir faire l'interrogation de vocabulaire dans les deux sens, voire en panachant thème et version au sein d'une même interrogation.

©2010-2011 - JACQUES THOORENS - HTTP://WWW.THOORENS.NET

Vocabulaire

DYNAMIQUE, ET CONDITIONNEL, INDICE, OCCURRENCE, SENTINELLE, TABLEAU.

©2010-2011 - JACQUES THOORENS - HTTP://WWW.THOORENS.NET