Enseignement Supérieur des Techniques de l'Entreprise

C O L L E C T I O N

Algorithmes

cours et exercices résolus

S. Boutin et S. Tormento

BTS Informatique de gestion - IUT Informatique

BTS et IUT de Gestion

Bréal

BREVET DE TECHNICIEN SUPERIEUR
INSTITUT UNIVERSITAIRE DE TECHNOLOGIE

ALGORITHMES
COURS ET EXERCICES

1e édition

par

Sylvie TORMENTO
Professeur d'informatique
en classe de BTS
au lycée Turgot de Paris

et

Sophie BOUTIN
Professeur d'informatique
en classe de BTS
à l'Ecole Nationale de Commerce de Paris

Bréal

1, rue de Rome 93561 Rosny Cedex

Dans la même collection :

Eléments d'analyse appliquée à l'informatique de gestion
Cours et exercices corrigés
par
J.-P. Faure

Négociations informatiques
par
C. Guédat

ISBN : 2 85394 964 8

« Le logo ci-contre mérite une explication. Son objet est d'alerter le lecteur sur la menace que représente pour l'avenir de l'écrit tout particulièrement dans le domaine des sciences humaines et sociales (ou de sciences, techniques, médecine ; ou de droit ; ou d'enseignement), le développement massif du **photocopillage**.

Le code de la propriété intellectuelle du 1er juillet 1992 interdit en effet expressément la photocopie à usage collectif sans autorisation des ayants droit. Or, cette pratique s'est généralisée dans les établissements d'enseignement supérieur, provoquant une baisse brutale des achats de livres, au point que la possibilité même pour les auteurs de créer des œuvres nouvelles et de les faire éditer correctement est aujourd'hui menacée.

Nous rappelons donc que toute reproduction, partielle ou totale, du présent ouvrage est interdite sans autorisation de l'auteur, de son éditeur ou du Centre français d'exploitation du droit de copie (CFC, 3, rue d'Hautefeuille, 75006 Paris) ».

Dépôt légal : août 1994
ISBN : 2 85394 5928

Introduction

Réalisant le manque de littérature informatique adaptée aux élèves de BTS, nous vous proposons une démarche pédagogique de résolution de problèmes de gestion. Ce livre est composé de nombreux exercices, d'une méthode pour aborder le problème et de la solution complète suivie de remarques liées à notre expérience de professeurs d'informatique dans les classes de BTS.

La programmation ne se réduit pas à la codification dans un langage convenu des actions à réaliser pour résoudre un problème.

Elle se compose :

- D'une étape de CONCEPTION de la solution.
- D'une étape de TRADUCTION dans un langage compréhensible par la machine.

Nous développons dans ce livre uniquement la conception de la démarche algorithmique. Cette étape n'est pas superflue, car sans elle, la mise au point, la recherche d'erreurs, la maintenance et les améliorations de programme sont difficiles.

L'algorithme est un langage de description des étapes de la résolution d'un problème.
Il doit :

- Contenir un nombre fini d'actions exécutables.
- Utiliser des données connues par l'utilisateur.
- Avoir au moins un résultat.
- Et toutes les opérations doivent pouvoir être exécutées par un homme utilisant des moyens manuels.

Le point de départ de l'élaboration d'un algorithme est d'identifier les données en entrée et les résultats à obtenir. Nous définissons ensuite les moyens pour obtenir les résultats recherchés à partir des données. La méthode que nous décrivons utilise la décomposition d'un problème initial en problèmes plus petits, le programme final résultant de leurs emboîtements. Il existe souvent plusieurs algorithmes de résolution à un problème, nous choisissons le meilleur en fonction des circonstances d'application.

Nous considérons comme nécessaire de structurer le travail préalable à la traduction dans un langage évolué, le lecteur qui voudra bien lire ce livre s'en convaincra aisément.

Sylvie TORMENTO

Sophie BOUTIN

Professeurs d'informatique
en classe de BTS
au lycée Turgot
et à l'Ecole Nationale de Commerce
Paris

Méthode d'utilisation de ce livre

Les chapitres de ce livre contiennent :

- Des définitions
  - Cours
- Des exemples
  - Représentations graphiques illustrant la définition
- Des exemples – applications
  - Vérification de la bonne compréhension de la définition
  - Petit jeu de questions-réponses
- Des algorithmes – exercices
  - Enoncé du problème
  - Méthode de travail pour vous aider, si besoin, à commencer vous-même l'écriture de la solution
  - Solution complète de l'exercice
  - Remarques pour vous aider à détecter les erreurs de votre solution
- Des problèmes
  - Récapitulatif des notions vues dans un chapitre.

Bon courage !

Sommaire

1

Généralités

I – Préliminaires

1 - NOTIONS GÉNÉRALES

Soit à définir l'ensemble des opérations élémentaires à réaliser entre le moment où le réveil sonne et le moment où l'on sort travailler.

1 – Le réveil sonne
2 – Je me réveille
3 – Je me lève
4 – Je prépare le café
5 – Je déjeune
6 – J'enlève mon pyjama
7 – Je me douche
8 – Je m'habille
9 – Je mets mes chaussures
10 – Je mets mon manteau
11 – J'ouvre la porte
12 – Je sors
13 – Je ferme la porte

Nous pouvons constater sur cet exemple simple que l'ordre des opérations a de l'importance. En effet, il serait très gênant d'intervertir les actions 7 et 6, ou 9 et 7, ou encore 13 et 11.

Supposons alors que la personne décrite précédemment ait en sa possession un parapluie. La suite des opérations doit lui permettre de ne prendre son parapluie que dans l'éventualité où il pleuvrait. Pour cela, nous disposons d'une structure alternative permettant de faire un choix.

De même, cette personne répète ces mêmes gestes jusqu'à ce que, par exemple, elle soit en vacances (auquel cas le réveil ne sonnera plus !!!). Nous disposons d'une structure itérative ou répétitive permettant de répéter cette suite d'actions jusqu'à ce que le réveil ne sonne plus.

DÉFINITION

Un algorithme fait passer d'un état initial à un état final de façon déterministe. Il doit respecter les règles suivantes.

- Il est défini sans ambiguïté.
- Il se termine après un nombre fini d'opérations.
- Il doit être effectif : toutes les opérations doivent pouvoir être effectuées par un homme utilisant des moyens manuels.
- Il manipule des objets qui doivent être définis de manière très précise.

Un algorithme est une suite d'actions ordonnées en séquence qui portent sur les objets d'un univers fini.

REMARQUES SYNTAXIQUES

Nous nommons un algorithme afin de pouvoir l'appeler sans le réécrire. Nous utilisons le terme "programme" pour le faire.

Nous le délimitons afin de permettre une cohabitation éventuelle avec d'autres programmes. Nous utilisons les termes DEBUT et FIN.

Nous commentons un algorithme afin de permettre une plus grande lisibilité. Pour cela, nous utilisons les sigles (* et *) pour délimiter ces commentaires. Le premier indique le début de la remarque et le deuxième la fin.

2 - NOTION D'OBJET

L'univers des objets d'un traitement est fini. Il est donc possible de le décrire entièrement et sans ambiguïté en décrivant chaque objet.

Le traitement d'un objet concerne la valeur de cet objet. Si cette valeur ne peut pas être modifiée, nous parlons de constante, sinon nous parlons de VARIABLE.

Un objet est parfaitement défini si nous connaissons ses trois caractéristiques. A savoir :

- ***Son identificateur***

Il est représenté par une suite quelconque de caractères alphanumériques (numériques et alphabétiques sans espace) commençant obligatoirement par une lettre. De préférence, le nom est choisi en rapport avec le contenu de l'objet.

Exemple 1 :

TOTAL ; NOMBRE_DE_LIVRES ; CUMUL1 sont licites.
1CUMUL ; NOMBRE DE LIVRES sont illicites.

- ***Sa valeur :*** constante ou variable
- ***Son type***

Nous ne pouvons pas appliquer de tráitement à la valeur d'un objet si nous ne connaissons pas son type. Un type est défini par un ensemble de constantes et l'ensemble des opérations que nous pouvons leurs appliquer. Nous connaissons trois grands types d'objet

- **Booleen**

ensemble des constantes : (.VRAI., .FAUX.)
ensemble des opérateurs : ET, OU, NON

Rappel des tables de vérité de ces opérateurs

a	b	a ET b
V	V	V
V	F	F
F	V	F
F	F	F

a	b	a OU b
V	V	V
V	F	V
F	V	V
F	F	F

a	non a
V	F
F	V

remarque :
sur les tables V signifie .VRAI. et F signifie .FAUX.
Les deux constantes logiques seront notées .VRAI. et .FAUX. afin de ne pas les confondre avec des identificateurs.

- **Numérique : entier ou réel**

ensemble des constantes : $\mathbb{R}$ ou $\mathbb{Z}$

ensemble des opérateurs : toutes les opérations arithmétiques et trigonométriques. Pour les opérations les plus courantes, nous notons

+	:	addition
*	:	multiplication
–	:	soustraction
/	:	division
^	:	élévation à la puissance
DIV	:	la division entière
MOD	:	le reste d'une division
ENT	:	la partie entière d'un réel

- **Chaîne de caractères**

Une chaîne de caractères est :
- soit une chaîne vide
- soit un caractère suivi d'une chaîne de caractère.

Un caractère est :
- soit une lettre de l'alphabet (majuscules et minuscules)
- soit différents codes : codes opérations (+, –, *, ...), codes de ponctuation (;, ., ...), autres codes ($, £, §, ...)

Un caractère est lié à un code numérique (ex code ASCII) qui le représente en machine et qui permet d'établir une relation d'ordre.

Exemple 2 :

- de chaîne de caractères :
  "Bonjour
  "Je vous souhaite la bienvenue"
  "Il est 12 heures"

- de relation d'ordre.
  "A" < "a" le code numérique lié à "A" est inférieur au code numérique lié a "a"
"1988" < "1989"
  "A" < "Z"
"1988" < "3" le 1er caractère "1" est inférieur au caractère "3"

ensemble des opérateurs : nous considérons comme prédéfinies les fonctions suivantes

`SSCHAINE (<chaîne>, <pos>, <nb caractères>)`
extraction dans <chaîne> à partir de la position <pos> de <nb caractères>

`<chaîne1> // <chaîne2>`
Concaténation de deux chaînes de caractères en une seule

`LONGUEUR (<chaîne>)`
donne le nombre de caractères situés entre les deux guillemets qui délimite la chaîne.

`RANG (<chaîne1>, <chaîne2>, <pos>)`
recherche dans la chaîne <chaîne1> la sous-chaîne <chaîne2> à partir de la position <pos>. Si <chaîne2> est trouvée, la fonction produit la position du premier caractère de <chaîne2> dans <chaîne1>, sinon 0.

`CODE (<caractère>)`
restitue la valeur du code numérique lié au caractère.

`CAR (<nombre>)`
restitue le caractère correspondant à cette valeur numérique. CAR est la fonction réciproque de CODE.

`CVCHAINE (<nombre>)`
transforme une variable ou une constante de type numérique, en variable ou constante de type chaîne de caractères.

`CVNOMBRE (<caractère>)`
transforme une variable ou une constante de type chaîne de caractères, contenant une valeur numérique, en une variable ou une constante de type numérique.

Exemple 3 :

```
SSCHAINE ("informatique", 2, 3)
"BONJOUR" // " A TOUS"
LONGUEUR ("élèves")
RANG ("Informatique", "format", 2)
RANG ("Informatique", "format", 4)
CODE ("A")
CAR(65)
CAR(CODE("A"))
CVCHAINE(8)
CVNOMBRE("8")
```

Résultat :

```
"nfo"
"BONJOUR A TOUS"
6
3
0
65
"A"
"A"
"8"
8
```

REMARQUE

L'univers des objets que nous manipulons doit être décrit préalablement au mot DEBUT de l'algorithme.

Nous attribuons à chaque objet son identificateur et son type. Si plusieurs objets sont du même type, nous les regroupons. Tous les objets de valeurs constantes sont déclarés en premier, et précédés du terme CONST. Les variables sont déclarées juste après et précédées du terme VAR. Les mots réservés CONST et VAR ne figurent qu'une seule fois par algorithme.

Exemple 4 :

```
CONST   MAXI c'est   50
        ANNÉE c'est  1989
VAR     JOUR,MOIS,AN : ENTIERS
        NOM,PRENOM : CARACTERES
        EXISTE  :  BOOLEEN
```

II – Instructions élémentaires

1 - AFFECTATION

DÉFINITION

L'opération consiste à affecter une valeur à un objet. Nous représentons cette opération par une flèche orientée à gauche '←'.

FORMAT GÉNÉRAL

`<identificateur> ← <valeur>`

La valeur peut être soit

- une variable de même type
- une constante du type de l'identificateur
- une expression dont l'évaluation produit un résultat du type de l'identificateur.

❑ Exemple d'application 1.

		avant		après
– NOM ← "Bernard"	NOM		NOM	Bernard
– NOM ← "jean "//"dupont"	NOM		NOM	jean dupont
– NOM ← "rené"//NOM	NOM	Durand	NOM	renéDurand
– NOM ← JEAN	NOM	Durand	NOM	Dupondt
	JEAN	Dupondt	JEAN	Dupondt

■ Algorithme – exercice 1

Ecrire un algorithme permettant de calculer le nombre de caractères d'un mot donné.

MÉTHODE :

Nous distinguons deux phases principales

- initialisation du mot
- calcul du résultat

Nous avons donc deux objets

- le mot dont la valeur sera liée à l'identificateur NOM de type chaîne de caractères
- la longueur du mot dont la valeur sera liée à l'identificateur LGNOM de type numérique

```
programme afficheversion1
var nom : caractères
    lgnom : numérique
Début
   (* initialisation *)
   nom ← "DUPONT"
   (* calcul du résultat *)
   lgnom ← LONGUEUR(NOM)
Fin
```

Cet algorithme est limité car il travaille toujours sur la même valeur de nom. De plus, le résultat qu'il produit n'est pas accessible à l'utilisateur.

2 - INSTRUCTIONS D'ENTRÉE ET DE SORTIE

DÉFINITION

Nous disposons d'une instruction de saisie qui permet de récupérer une valeur sur un périphérique d'entrée (le clavier), et d'une instruction d'affichage qui permet l'édition d'une valeur sur un périphérique de sortie (l'écran).

FORMAT GÉNÉRAL

saisir <identificateur>	(*instruction d'entrée*)
afficher <valeur>	(*instruction de sortie*)

REMARQUE

l'identificateur est un objet déclaré
la valeur située derrière l'ordre d'affichage peut être soit

- un identificateur
- une constante
- une expression

Dans le cas d'un dialogue via le clavier et l'écran, il convient d'afficher des messages permettant de faciliter la communication.

Exemple 5:

```
Saisir   NOM
Saisir   AGE
Saisir   NOMBRE1, NOMBRE2
Afficher NOM
Afficher NOM," a aujourd'hui ",AGE," ans"
Afficher "la somme des 2 nombres est ", NOMBRE1 + NOMBRE2
```

❑ Exemple d'application 2

Reprenons l'exercice précédent et rendons-le plus convivial.

MÉTHODE

Nous distinguons deux étapes supplémentaires

– Obtention du nom à partir du clavier

– Affichage du résultat

Nous ne créons aucun objet supplémentaire.

```
programme afficheversion2
VAR  nom : caractères
     lgnom : entier
DEBUT
   (* entrée au clavier du nom *)
   afficher "entrez un nouveau nom "
   saisir nom
   (* calcul du résultat *)
   lgnom ← LONGUEUR(nom)
   (* affichage du résultat à l'écran *)
   afficher "la longueur du mot est ", lgnom
FIN
```

■ Algorithme – exercice 2

A partir de la saisie de trois nombres, écrire l'algorithme permettant d'en effectuer la somme, le produit et la moyenne.

MÉTHODE :

Nous distinguons trois phases :

- Obtention des trois nombres

  Il nous faudra trois objets de type réel pour recevoir ces nombres

- Calcul des différents résultats

  Il nous faudra pour cela déclarer trois objets de type réel qui reçoivent le résultat de chacune des opérations : SOMME, PRODUIT, MOYENNE

- Edition des résultats

```
programme calcul
var somme, produit, moyenne, nb1, nb2, nb3 : réels
DEBUT
   (* saisie des trois nombres *)
   affichez "entrez vos trois nombres "
   saisir nb1, nb2, nb3
   (* réalisation des différentes opérations *)
   somme ← nb1 + nb2 + nb3
   produit ← nb1 * nb2 * nb3
   moyenne ← somme / 3
   (* édition des résultats *)
   afficher "la somme de ces trois nombres est ", somme
   afficher "le produit de ces trois nombres est ", produit
   afficher "la moyenne de ces trois nombres est ", moyenne
FIN
```

REMARQUE

Nous pouvons afficher directement le résultat des différentes opérations sans passer par des variables intermédiaires (SOMME, PRODUIT et MOYENNE).

```
afficher "la somme de ces trois nombres est ", nb1 + nb2 + nb3
afficher "le produit de ces trois nombres est ", nb1 * nb2 * nb3
afficher "la moyenne de ces trois nombres est ", (nb1 + nb2 + nb3) / 3
```

■ Algorithme – exercice 3

Extraire les initiales d'une personne dont le nom et le prénom sont saisis au clavier.

METHODE

Nous distinguons trois phases à l'algorithme

- Saisie du nom et du prénom
- Recherche des initiales :

nous disposons pour cela de la fonction SSCHAINE. Nous aurons également besoin de deux identificateurs de type chaîne de caractères qui contiendront les initiales :

```
INOM ← SSCHAINE(NOM,1,1)
IPRENOM ← SSCHAINE(PRENOM,1,1)
```

- Affichage du résultat

```
programme initialeversion1
var nom, prénom, inom, iprénom : caractères
DEBUT
   (* saisie du Nom *)
   afficher "entrez le nom de la personne "
   saisir nom
   (* saisie du prénom *)
   afficher "entrez le prénom de la personne"
   saisir prénom
   (* extraction des initiales *)
   inom ← SSCHAINE(nom,1,1)
```

```
    iprénom ← SSCHAINE(prénom,1,1)
    (* affichage des initiales à l'écran *)
    afficher "les initiales de cette personne sont ",inom//iprénom
FIN
```

REMARQUE

L'énoncé ne précise pas si le nom et le prénom sont saisis dans une seule et même variable, ou séparément. Utilisant un objet de type chaîne de caractères, CHAINE, contenant le nom et le prénom separés par un asterisque, nous nous proposons d'extraire le nom, le prénom et les initiales et de les afficher.

Pour pouvoir extraire le nom et le prénom de CHAINE, il faut repérer au préalable la position du caractère '*' dans CHAINE. C'est la fonction RANG qui va le permettre.

```
POS1 ← RANG(CHAINE,"*",1)
```

Or le nom s'arrête juste avant le caractère '*', donc à POS1 – 1. D'où

```
NOM ← SSCHAINE(CHAINE,1,POS1   1)
```

Il faut également repérer la fin de la variable CHAINE, c'est la fonction LONGUEUR qui le permet.

```
POS2 ← LONGUEUR(CHAINE)
```

Or le prénom commence juste après le caractère '*', c'est-à-dire à POS1 + 1, et se termine à POS2, donc possède POS2 – POS1 caractères. D'où

```
PRENOM ← SSCHAINE(CHAINE,POS1+1,POS2 - POS1)
```

```
programme initialeversion2
var nom, prénom, chaîne, inom, iprénom : caractères
    pos1, pos2 : entiers
DEBUT
   (* saisie du nom et du prénom simultanément *)
   afficher "entrez le nom et le prénom de la personne en les
             séparant par une '*' "
   saisir chaîne
   (* recherche de la position du 1'* indiquant la fin du nom *)
   pos1 ← RANG(chaîne,"*",1)
   (* recherche de la position du dernier caractère du prénom *)
   pos2 ← LONGUEUR(chaîne)
   (* extraction du nom et du prénom *)
   nom←SSCHAINE(chaîne,1,pos1 - 1)
   prénom ← SSCHAINE(chaîne,pos1 + 1,pos2 - pos1)
   (* extraction des initiales *)
   inom ← SSCHAINE(nom,1,1)
   iprénom ← SSCHAINE(prénom,1,1)
   (* affichage du nom, du prénom et des initiales *)
   afficher "le nom de cette personne est ",nom
   afficher "le prénom de cette personne est ",prénom
   afficher "les initiales de cette personne sont ",inom//iprénom
FIN
```

III – Les structures conditionnelles

1 - LA STRUCTURE ALTERNATIVE

FORMAT GÉNÉRAL

```
SI    <condition>
ALORS <action1>
SINON <action2>
FSI
```

Lorsque l'évaluation de la condition produit la valeur

.VRAI. : l'action1 est exécutée

.FAUX. : l'action2 est exécutée

Action1, comme action2, peuvent être soit

- une instruction
- un ensemble d'instructions
- un algorithme

❑ Exemple d'application 3

Ecrire l'algorithme permettant d'afficher la valeur absolue de la différence entre deux nombres saisis au clavier.

MÉTHODE

Nous déclarons trois objets de type réel. Les deux premiers A et B reçoivent les valeurs des nombres traités, le troisième C reçoit la différence.

Pour que la différence soit toujours positive il faut calculer

A – B si A > B, et B – A si A < B. Nous avons donc une structure alternative.

```
programme valabsolueversion1
VAR A, B, C : réels
DEBUT
   (* saisie des nombres A et B *)
   afficher "entrez deux nombres "
   saisir A, B
   (* calcul de la différence *)
   SI A > B
   ALORS C ← A - B
   SINON C ← B - A
   FSI
   (* édition du résultat *)
   afficher "la valeur absolue de la différence est ", C
FIN
```

■ Algorithme – exercice 4

Comparer deux lettres saisies au clavier et afficher la première dans l'ordre alphabétique

MÉTHODE

Nous avons besoin de deux objets de type chaîne de caractères LETTRE1 et LETTRE2. Nous utilisons une structure alternative dont la condition sera LETTRE1 > LETTRE2.

```
programme comparaison
VAR lettre1, lettre2 : caractères
DEBUT
   (* saisie des deux lettres *)
   afficher "entrez deux lettres "
   saisir lettre1, lettre2
   (* comparaison et édition du résultat *)
   SI lettre1 > lettre2
   ALORS afficher lettre1
   SINON afficher lettre2
   FSI
FIN
```

2 – LA STRUCTURE CONDITIONNELLE

FORMAT GÉNÉRAL

```
SI <condition>
ALORS <action>
FSI
```

Cette structure présente la particularité de ne pas avoir de traitement à effectuer lorsque l'évaluation de la condition produit la valeur .FAUX.

❑ Exemple d'application 4

Reprendre l'exemple de la structure alternative (la valeur absolue).

MÉTHODE

Nous constatons que l'analyse aurait pu être faite d'une autre manière. En effet, nous pouvons effectuer systématiquement A – B, l'affecter à C, corriger la valeur de C seulement si celle-ci est négative. En effet (A – B) = – (B – A)

```
programme valabsolueversion2
VAR A, B, C : réels
DEBUT
   (* saisie des deux nombres *)
   afficher "entrez vos deux nombres"
   saisir A, B
   (* initialisation de C *)
   C ← A - B
```

```
   (* correction de la valeur de C si celle-ci est négative *)
   SI C < 0
   ALORS C ← (-C)
   FSI
   (* édition du résultat *)
   afficher "la valeur absolue de la différence est ", C
FIN
```

■ Algorithme – exercice 5

A partir de la saisie d'une année de naissance, un message particulier s'affiche pour les enfants de moins de trois ans.

MÉTHODE

Nous créons un objet ANNAISS de type entier qui recevra l'année de naissance. Nous calculons l'âge en effectuant la différence entre l'année en cours et l'année de naissance et nous le comparons avec la valeur 3.

```
programme âge
CONST ancours c'est 1989
VAR annaiss : entier
DEBUT
   (* saisie de l'année de naissance *)
   afficher "entrez une année de naissance "
   saisir annaiss
   (* édition du message pour les moins de trois ans *)
   SI (ancours - annaiss) < 3
   ALORS afficher "Tous nos compliments, vous serez primé en
                   fin d'année !"
   FSI
FIN
```

3 – IMBRICATIONS DE SI

Il se peut, dans certains cas, que l'expression de la condition d'un SI ne suffise pas pour exprimer tous les cas de figures. Nous pouvons alors exprimer un SI après le ALORS d'un SI, et/ou après le SINON d'un SI. Nous pouvons également mélanger les structures alternatives et conditionnelles.

❑ Exemple d'application 5

Reprendre le thème précédent en vérifiant que l'année de naissance donnée n'est pas supérieure à l'année en cours.

MÉTHODE

Nous reprenons la même structure que précédemment en ajoutant une comparaison de l'année de naissance et de l'année en cours.

```
programme âgeversion2
CONST ancours c'est 1989
VAR annaiss : entier
DEBUT
   (* saisie de l'année de naissance *)
   afficher "entrez l'année de naissance "
   saisir annaiss
   (* vérification de la vraisemblance de cette année *)
   SI annaiss > ancours
   ALORS afficher "attention, vous n'êtes pas encore nés !"
   SINON (* vérification de l'âge de la personne *)
         SI (ancours - annaiss) < 3
         ALORS afficher "félicitations !"
         FSI
   FSI
FIN
```

■ Algorithme – exercice 6

Un robot conduit une voiture. Il peut exécuter 3 actions "s'arrêter", "ralentir", "passer" en fonction de la couleur des feux qui sera une variable saisie.

MÉTHODE

Nous découpons le travail en deux étapes :

- obtention de la couleur des feux. Il nous faut pour cela un objet de type chaîne de caractères qui reçoit la couleur du feu.
- si le feu est rouge le robot s'arrête, si le feu est orange le robot ralentit, si le feu est vert le robot passe.

```
programme robotversion1
var couleur : caractères
DEBUT
   (* saisie de la couleur du feu *)
   afficher "quelle est la couleur du feu ?"
   saisir couleur
   (* si le feu est rouge *)
   SI    couleur = "rouge"
   ALORS afficher "je m'arrête"
   SINON (* si le feu est orange *)
         SI    couleur = "orange"
         ALORS afficher "je ralentis"
         SINON (* le feu est vert *)
               afficher "je passe"
         FSI
   FSI
FIN
```

REMARQUE

Si nous faisons une erreur d'appréciation de la couleur des feux, le robot passe et risque d'écraser des piétons !!!

■ Algorithme – exercice 7 *

Reprendre l'énoncé précédent en tenant compte de la remarque, et, du fait que lorsque le feu est vert, il faut regarder si un piéton est engagé.

MÉTHODE

Nous reprenons les mêmes objets que précédemment, en ajoutant un objet supplémentaire de type chaîne de caractères, qui réceptionne la réponse à la question "y a-t-il des piétons ?"
Si le feu est rouge le robot s'arrête, si le feu est orange le robot ralentit, si le feu est vert il existe deux possibilités, il y a des piétons le robot ralentit, sinon il passe.

```
programme robotversion2
var couleur, réponse : caractères
DEBUT
   (* saisie de la couleur du feu *)
   afficher "de quelle couleur est le feu ?"
   saisir couleur
   SI couleur = "rouge"
   ALORS afficher "je m'arrête"
   SINON (* si le feu est orange *)
         SI couleur = "orange"
         ALORS afficher "je ralentis"
         SINON (* si le feu est vert *)
               SI couleur = "vert"
               ALORS (* présence de piétons *)
                     afficher "y a-t-il des piétons (oui/non) ?"
                     saisir réponse
                     SI réponse = "oui"
                     ALORS afficher "je ralentis"
                     SINON afficher "je passe"
                     FSI
               SINON afficher "erreur de jugement de couleur"
               FSI
         FSI
   FSI
FIN
```

REMARQUE

Comme précédemment, si la réponse à la question "y a-t-il des piétons ?" n'est ni oui, ni non, la voiture écrasera des piétons. Il faut donc introduire une nouvelle alternative après le sinon du si réponse = "oui" :

```
(* le début du programme est identique *)
SI couleur = "rouge"
ALORS afficher "je m'arrête"
SINON SI couleur = "orange"
      ALORS afficher "je ralentis"
```

```
        SINON SI  couleur = "vert"
              ALORS afficher "y a-t-il des piétons ?"
                    saisir réponse
                    SI réponse = "oui"
                    ALORS afficher "je ralentis"
                    SINON SI réponse = "non"
                          ALORS afficher "je passe"
                          SINON afficher "erreur dans votre réponse"
                          FSI
                    FSI
              SINON afficher "erreur de jugement de couleur"
              FSI
        FSI
FSI
```

■ Algorithme – exercice 8

A partir de la saisie de deux nombres a et b, résoudre l'équation "ax + b = 0"

MÉTHODE

Nous utilisons deux objets de type réels a et b et nous appliquons les règles mathématiques suivantes :
Si a = 0 et b = 0, alors l'ensemble des solutions est $\mathbb{R}$
Si a = 0 et b ≠ 0, alors l'ensemble des solutions est l'ensemble vide
Si a ≠ 0, alors la solution est (–b/a)

```
programme équation
var a, b : réels
DEBUT
   (* saisie des deux nombres a et b *)
   afficher "entrez les coefficients a et b"
   saisir a, b
   (* résolution de l'équation *)
   SI    a = 0
   ALORS SI    b = 0
         ALORS afficher "l'ensemble des solutions est ℝ"
         SINON afficher "l'ensemble des solutions est l'ensemble vide"
         FSI
   SINON afficher "la solution unique est ",(-b/a)
   FSI
FIN
```

■ Algorithme – exercice 9

A partir de la saisie du prix unitaire d'un produit (PU) et de la quantité commandée (QTCOM), afficher le prix à payer (PAP), en détaillant le port (PORT) et la remise (REM), sachant que :

- le port est gratuit si le prix des produits (TOT) est supérieur à 500 F. Dans le cas contraire, le port est de 2 % du TOT.

• la remise est de 5 % si TOT est compris entre 200 et 1000 francs et de 10 % au-delà

MÉTHODE

Six objets de type réel sont manipulés (PU, QTCOM, PAP, PORT, REM, TOT)
Les calculs sont les suivants :

- calcul du total
  TOT ← PU * QTCOM
- calcul du port
  nous comparons TOT et 500. Deux cas sont possibles :
  • le port est gratuit
  • le port est de 2% de TOT
- calcul de la remise
  nous comparons TOT avec les valeurs 1000 et 200. Trois cas sont possibles :
  • pas de remise
  • remise de 5 % de TOT
  • remise de 10 % de TOT
- calcul du prix à payer en fonction de TOT, PORT ET REM
- édition de la facture dans le détail

```
programme facture
var pu, qtcom, rem, port, tot, pap : réels
DEBUT
   (*saisie du prix unitaire et de la quantité commandée *)
   afficher "entrez le prix unitaire et la quantité commandée"
   saisir pu, qtcom
   (* calcul du total net *)
   tot ← pu * qtcom
   (* calcul du port *)
   SI tot > 500
   ALORS port ← 0
   SINON port ← tot * 0,02
   FSI
   (calcul de la remise *)
   SI tot > 1000
   ALORS rem ← tot * 0,1
   SINON SI tot >= 200
         ALORS rem ←  tot * 0,05
         SINON rem  ← 0
         FSI
   FSI
   (* calcul du prix à payer *)
   pap ← tot + port - rem
   (* édition de la facture *)
   afficher "le prix des produits est ", pu, "*", qtcom, "=", tot
   afficher "le port du est ", port
   afficher "la remise est de ", rem
   afficher "total du ", pap
FIN
```

Attention les instructions suivantes

```
SI tot >= 200 et tot <= 1000
ALORS rem ← tot * 0,05
SINON rem ← tot * 0,1
FSI
```

accordent une remise de 10 % pour un total inférieur à 200 !!!

COMPLÉMENT :
si nous ajoutons aux conditions de départ, la condition suivante "la valeur minimale du port à payer est de 6 francs", nous obtenons très facilement la modification :

```
(* reprise du calcul du port *)
SI tot > 500
ALORS port ← 0
SINON port ← tot * 0,02
      SI port < 6
      ALORS port ← 6
      FSI
FSI
```

Attention les instructions suivantes :

```
SI tot > 500
ALORS port ← 0
SINON port ← tot * 0,02
FSI
SI port < 6
ALORS port ← 6
FSI
```

affecte le port de la valeur 6 lorsque le total est supérieur à 500 francs !!!

Nous constatons qu'il est préférable de ne pas résoudre tous les problèmes simultanément. En effet, une approche aurait pu consister en ceci

	0	200	500	1000
port	2%	2%	0	0
remise	0	5%	5%	10%

mais la moindre modification sur les conditions de départ, entraîne des difficultés de mise à jour du programme. Si la valeur minimale du port est de 6 francs, il y a deux structures conditionnelles à ajouter : une dans l'intervalle (0,200) et la même dans l'intervalle (200, 500). Il est donc très important de partitionner un algorithme.

4- STRUCTURE DE CHOIX

FORMAT GÉNÉRAL

```
SUIVANT <variable ou expression> FAIRE
  <valeur1>  :  <action1>
  <valeur2>  :  <action2>
  <valeur3>  :  <action3>
      .             .
```

```
        .                .
        .                .
    <valeurn>  :  <actionn>
    SINON : <action par défaut>
FINSUIVANT
```

La variable (ou l'évaluation de l'expression) est comparée aux différentes constantes (valeur 1 à valeur n)et les actions entreprises dépendent de cette valeur. Nous disposons d'une action par défaut pour le cas où la variable n'est égale à aucune des constantes énumérées.

REMARQUE

La structure de choix permet une présentation plus claire d'un ensemble d'alternatives imbriquées.

Plusieurs valeurs peuvent entraîner un même traitement. Nous pouvons alors énumérer ces valeurs en les séparant par des virgules, avant de décrire le traitement à entreprendre.

```
Suivant nombre faire
   0 : afficher " nombre nul (et pair)"
   1, 3, 5 : afficher " nombre impair"
   2,4 : afficher "  nombre pair"
finsuivant
```

Une action peut être une instruction, ou une suite d'instructions, ou encore un algorithme.

❑ Exemple d'application 6

Reprenons l'énoncé du robot qui doit identifier la couleur des feux (algorithme - exercice 6.)

MÉTHODE

Les objets manipulés sont les mêmes que dans l'algorithme robotversion1, mais cette fois nous allons hiérarchiser les actions à entreprendre. Nous distinguons une réponse valide (vert, rouge ou orange), d'une réponse non valide.

```
suivant la couleur du feu
rouge  : le robot s'arrête
orange : le robot ralentit
vert   : le robot passe
sinon  : la couleur n'est pas une couleur existante pour les feux
```

```
programme robotversion3
var couleur : caractères
DEBUT
   (* couleur du feu *)
   affichez "de quelle couleur est le feu ?"
   saisir couleur
   (* hiérarchisation des traitements *)
   SUIVANT couleur FAIRE
     "vert"   :  afficher "je passe"
     "orange" :  afficher "je ralentis"
     "rouge"  :  afficher "je m'arrête"
     SINON    :  afficher "cette couleur n'est pas une couleur de feu"
   FINSUIVANT
FIN
```

Algorithme – exercice 10

A partir d'un menu affiché à l'écran, effectuer la somme ou le produit ou la moyenne de trois nombres. Nous appelons "menu" l'association d'un numéro sequentiel aux différents choix proposés par un programme.

MÉTHODE

Nous utilisons quatre objets de type numérique (3 réels, 1 entier) pour recevoir respectivement les trois valeurs et le choix de l'opération à effectuer.

Suivant choix
  multiplication : affichez nb1 * nb2 * nb3
  somme : affichez nb1 + nb2 + nb3
  moyenne : affichez (nb1 + nb2 + nb3) / 3
  sinon : affichez opération inconnue

```
programme menuversion1
var nb1, nb2, nb3 : réels
    choix : entier
DEBUT
   (* saisie des trois nombres *)
   afficher "entrez trois nombres"
   saisir nb1, nb2, nb3
   (* affichage du menu et saisie du choix *)
   afficher " 1 - pour la multiplication"
   afficher " 2 - pour la somme "
   afficher " 3 - pour la moyenne"
   afficher "votre choix "
   saisir choix
   (* hiérarchisation du choix *)
   SUIVANT choix FAIRE
     1 : afficher "le produit des 3 nombres est " ,nb1 * nb2 * nb3
     2 : afficher "la somme des 3 nombres est " ,nb1 + nb2 + nb3
     3 : afficher "la moyenne des 3 nombres est " ,(nb1 + nb2 + nb3)/3
     SINON : afficher "cette saisie est incorrecte"
   FINSUIVANT
FIN
```

Algorithme – exercice 11

Complétez le menu de l'algorithme - exercice 10 en proposant d'effectuer la recherche du minimum ou du maximum des trois nombres.

MÉTHODE

Nous repartons sur la même base que précédemment avec deux choix supplémentaires :

Si le choix est la recherche du minimum, alors nous supposons que nb1 est le minimum relatif. Nous comparons la valeur relative du minimum avec le nombre nb2. Si nb2 est inférieur, il devient le minimum relatif. Nous comparons ensuite nb3 et le minimum relatif. Si nb3 est inférieur, il devient le minimum relatif. Nous avons alors le minimum absolu. Si le choix est la recherche du maximum, alors nous supposons que nb1 est le maximum relatif. Si nb2 est supérieur au maximum relatif, il devient le maximum relatif. Nous comparons ensuite nb3 et le maximum relatif. Si nb3 est supérieur au maximum

relatif, il devient le maximum relatif. Nous avons alors le maximum absolu.

```
programme menuversion2
var nb1, nb2, nb3, min, max : réels
   choix : entier
DEBUT
   (* saisie des trois nombres *)
   affichez "entrez les trois nombres "
   saisir nb1, nb2, nb3
   (* affichage du menu *)
   afficher "1 - pour la multiplication"
   afficher "2 - pour la somme"
   afficher "3 - pour la moyenne"
   afficher "4 - pour la recherche du minimum"
   afficher "5 - pour la recherche du maximum"
   afficher "votre choix "
   saisir choix
   (* hiérarchisation des traitements *)
   SUIVANT choix FAIRE
     1 : afficher "le produit des 3 nombres est ", nb1 * nb2 * nb3
     2 : afficher "la somme des 3 nombres est ", nb1 + nb2 + nb3
     3 : afficher "la moyenne des 3 nombres est ", (nb1 + nb2 + nb3) / 3
     4 : min ← nb1
         SI min > nb2
         ALORS min ← nb2
         FSI
         SI min > nb3
         ALORS min ← nb3
         FSI
         afficher"le plus petit des 3 nombres est ", min
     5 : max ← nb1
         SI max < nb2
         ALORS max ← nb2
         FSI
         SI max < nb3
         ALORS max ← nb3
         FSI
         afficher "le plus grand des 3 nombres est ", max
    SINON : afficher "erreur dans votre choix"
  FINSUIVANT
FIN
```

IV – La structure itérative

FORMAT GÉNÉRAL :

```
TANT QUE <condition>
   <action>
FTQ
```

Cette structure permet la répétition d'une (ou de plusieurs) action(s) tant qu'une condition est satisfaite. La condition est testée avant la première exécution de l'action définie. La

relatif, il devient le maximum relatif. Nous avons alors le maximum absolu.

```
programme menuversion2
var nb1, nb2, nb3, min, max : réels
   choix : entier
DEBUT
   (* saisie des trois nombres *)
   affichez "entrez les trois nombres "
   saisir nb1, nb2, nb3
   (* affichage du menu *)
   afficher "1 - pour la multiplication"
   afficher "2 - pour la somme"
   afficher "3 - pour la moyenne"
   afficher "4 - pour la recherche du minimum"
   afficher "5 - pour la recherche du maximum"
   afficher "votre choix "
   saisir choix
   (* hiérarchisation des traitements *)
   SUIVANT choix FAIRE
    1 : afficher "le produit des 3 nombres est ", nb1 * nb2 * nb3
    2 : afficher "la somme des 3 nombres est ", nb1 + nb2 + nb3
    3 : afficher "la moyenne des 3 nombres est ", (nb1 + nb2 + nb3) / 3
    4 : min ← nb1
        SI min > nb2
        ALORS min ← nb2
        FSI
        SI min > nb3
        ALORS min ← nb3
        FSI
        afficher"le plus petit des 3 nombres est ", min
    5 : max ← nb1
        SI max < nb2
        ALORS max ← nb2
        FSI
        SI max < nb3
        ALORS max ← nb3
        FSI
        afficher "le plus grand des 3 nombres est ", max
    SINON : afficher "erreur dans votre choix"
   FINSUIVANT
FIN
```

IV – La structure itérative

FORMAT GÉNÉRAL :

```
TANT QUE <condition>
   <action>
FTQ
```

Cette structure permet la répétition d'une (ou de plusieurs) action(s) tant qu'une condition est satisfaite. La condition est testée avant la première exécution de l'action définie. La

relatif, il devient le maximum relatif. Nous avons alors le maximum absolu.

```
programme menuversion2
var nb1, nb2, nb3, min, max : réels
   choix : entier
DEBUT
   (* saisie des trois nombres *)
   affichez "entrez les trois nombres "
   saisir nb1, nb2, nb3
   (* affichage du menu *)
   afficher "1 - pour la multiplication"
   afficher "2 - pour la somme"
   afficher "3 - pour la moyenne"
   afficher "4 - pour la recherche du minimum"
   afficher "5 - pour la recherche du maximum"
   afficher "votre choix "
   saisir choix
   (* hiérarchisation des traitements *)
   SUIVANT choix FAIRE
    1 : afficher "le produit des 3 nombres est ", nb1 * nb2 * nb3
    2 : afficher "la somme des 3 nombres est ", nb1 + nb2 + nb3
    3 : afficher "la moyenne des 3 nombres est ", (nb1 + nb2 + nb3) / 3
    4 : min ← nb1
        SI min > nb2
        ALORS min ← nb2
        FSI
        SI min > nb3
        ALORS min ← nb3
        FSI
        afficher"le plus petit des 3 nombres est ", min
    5 : max ← nb1
        SI max < nb2
        ALORS max ← nb2
        FSI
        SI max < nb3
        ALORS max ← nb3
        FSI
        afficher "le plus grand des 3 nombres est ", max
    SINON : afficher "erreur dans votre choix"
   FINSUIVANT
FIN
```

IV – La structure itérative

FORMAT GÉNÉRAL :

```
TANT QUE <condition>
  <action>
FTQ
```

Cette structure permet la répétition d'une (ou de plusieurs) action(s) tant qu'une condition est satisfaite. La condition est testée avant la première exécution de l'action définie. La

condition exprimée après l'instruction TANT QUE permet l'exécution de l'action définie. Pour l'exprimer, nous remarquons qu'il est souvent plus aisé de rechercher la condition d'arrêt du traitement. Nous prenons alors la proposition contraire, en tenant compte des tables de l'algèbre de Boole, énoncées dans la première partie de ce chapitre. Nous utilisons une structure itérative lorsque l'action peut être exécutée 0 à N fois. Autrement dit, lorsqu'il n'est pas exclu que les événements décrits se produisent 0 fois.

❑ Exemple d'application 7

Ecrire l'algorithme permettant d'afficher à l'écran une suite de noms saisis au clavier. Nous souhaitons également compter le nombre de noms saisis.

MÉTHODE

Nous distinguons trois phases :

– initialisation d'un compteur de noms à zéro;

– saisir les noms, les compter et les afficher;

– éditer le nombre de noms saisis.

Deux objets sont manipulés dans ce traitement :

– NBNOM, du type numérique, contenant le nombre de noms;

– NOM, du type chaîne de caractères, contenant le nom saisi au clavier.

Pour chaque nom saisi au clavier, nous devons l'afficher et le compter. Pour arrêter le traitement, trouvons un nom fictif, appellons-le valeur sentinelle, et fixons lui la valeur "zzz". D'où une première approche possible :

```
saisir nom
si nom <> "zzz"
alors afficher nom
     nbnom ← nbnom + 1
     saisir nom
     si  nom <> "zzz"
     alors  afficher nom
            nbnom ← nbnom + 1
            saisir nom
            si nom <> "zzz"
            alors afficher nom
                  nbnom ← nbnom + 1
                  saisir nom
                  si …
                     …
```

Nous voyons immédiatement que cette structure est mal adaptée à une suite d'objets dont on ne connaît pas le nombre. Nous utilisons alors la structure itérative. La condition d'arrêt du traitement est que le nom soit égal à la valeur "zzz". La condition contraire, permettant la réalisation du traitement, est que le nom soit différent de la valeur "zzz". Ce qui donne cette première écriture :

```
TANT QUE nom <> "zzz"
   saisir nom
   afficher nom
   nbnom ← nbnom + 1
FTQ
```

Or, cet algorithme ne peut pas fonctionner. Pour pouvoir réaliser le premier test sur le nom, il faut que l'identificateur contienne une donnée. Il convient de mettre une première saisie du nom avant la structure itérative.

L'algorithme est le suivant

```
Saisir nom
TANT QUE nom <> "zzz"
   saisir nom
   afficher nom
   nbnom ← nbnom + 1
FTQ
```

Nous voyons que cet algorithme présente deux défauts

- il ne traite pas le premier nom (affichage et comptage)
- il compte la valeur sentinelle comme un nom correct.

Nous devons donc déplacer la saisie contenue dans la boucle en dernière instruction du bloc répétitif.

```
programme compterversion1
var nom : caractères
    nbnom : entier
DEBUT
   (* saisie du premier nom *)
   afficher "entrez un nom (ou zzz pour arrêt) "
   saisir nom
   (* initialisation du compteur de nom *)
   nbnom ← 0
   (* début de la boucle de traitement *)
   TANT QUE nom <> "zzz"
     (* édition du nom saisi *)
     afficher nom
     (* incrémentation du compteur de nom *)
     nbnom ← nbnom + 1
     (* saisie d'un autre nom *)
     afficher "entrez un nom (ou zzz pour arrêt)"
     saisir nom
   FTQ
   (* édition du compteur de nom *)
   afficher "vous avez saisi " ,nbnom ,"nom (s)"
FIN
```

■ Algorithme – exercice 12

Ecrire l'algorithme permettant la saisie de la réponse à la question : "aimez-vous l'informatique (O/N) ?". Nous devons afficher un message en cas de mauvaise réponse, et renouveller la question jusqu'à ce que la réponse soit correcte.

MÉTHODE

Nous utilisons un objet du type caractère recevant la réponse.

Nous distinguons le premier affichage de la question, des suivants. La réponse peut être correcte dès le départ, auquel cas nous n'affichons pas de message d'erreur. Nous répétons l'affichage du message d'erreur et la saisie d'une nouvelle réponse autant de fois que nécessaire. La condition d'arrêt du traitement est la suivante : réponse = "O" ou réponse = "N".

```
programme question
var rep : caractère
DEBUT
   (* premier affichage de la question *)
   afficher "aimez-vous l'informatique (O/N) ?"
   saisir rep
   (* boucle de test de la réponse *)
   TANT QUE rep <> "O" et rep <> "N"
     (* affichage du message d'erreur *)
     afficher "erreur dans votre réponse, recommencez "
     (* ré-affichage de la question *)
     afficher "aimez-vous l'informatique (O/N) ?"
     saisir rep
   FTQ
FIN
```

V – La structure répétitive

FORMAT GÉNÉRAL

```
REPETER
  <action>
JUSQU'A <condition>
```

Cette structure permet la répétition d'une action jusqu'à ce qu'une condition soit vérifiée. Elle ressemble à la structure itérative, à cette différence près que la condition exprimée permet l'arrêt du traitement. De plus, elle n'est testée qu'après une première exécution de l'action définie. Les instructions sont exécutées au moins une fois. La condition étant évaluée en fin de boucle, la structure répétitive ne nécessite pas, comme la structure itérative, une action préalable au test hors de la boucle.

❑ Exemple d'application 8

Ecrire l'algorithme permettant de calculer l'âge d'une personne à partir de son année de naissance. Nous recommençons autant de fois que l'utilisateur le désire.

MÉTHODE

Nous distinguons trois phases :
- saisie de l'année de naissance;
- calcul de l'âge;
- saisie de la réponse à la question "voulez-vous recommencer ?"

Nous utilisons deux objets :
- ANNAISS, de type entier, qui reçoit l'année de naissance;
- REP, de type chaîne de caractères, qui reçoit la réponse à la question.

L'âge est obtenu par différence entre l'année de naissance et l'année en cours.

Nous arrêtons le traitement principal lorsque la réponse à la question est N (non).

```
programme ageversion1
var  annaiss : entier
     rep : caractère
DEBUT
   (* répétitive principale *)
   REPETER
     (* saisie de l'année de naissance *)
     afficher "entrez une année de naissance "
     saisir annaiss
     (* calcul et affichage de l'âge *)
     afficher "vous avez ", 1989 - annaiss, "ans "
     (* saisie de la réponse à la question *)
     afficher "voulez-vous recommencer (O/N) ?"
     saisir rep
   JUSQU'A rep = "N"
FIN
```

Algorithme – exercice 13

L'utilisateur peut faire une erreur lors de la saisie de l'année de naissance. Nous nous proposons de valider sa réponse en lui demandant s'il confirme la valeur donnée.

Nous complétons l'exercice d'application 8.

MÉTHODE

Nous utilisons les mêmes objets que précédemment. Nous ajoutons une répétitive : la saisie de l'année de naissance doit être recommencée jusqu'à ce qu'elle soit validée.

```
programme ageversion2
var  annaiss : entier
     rep : caractères
DEBUT
   (* répétitive principale *)
   REPETER
     (* répétitive sur l'année de naissance *)
     REPETER
         (* saisie de l'année de naissance *)
         afficher "entrez l'année de votre naissance "
```

```
        saisir annaiss
        afficher "confirmez-vous la valeur (O/N) ?"
        saisir rep
      JUSQU'A rep = "O"
      (* affichage de l'âge *)
      afficher "vous avez ", 1989 - annaiss ," an(s) "
      (* saisie de la réponse *)
      afficher "voulez-vous recommencer (O/N) ?"
      saisir rep
    JUSQU'A rep = "N"
FIN
```

■ Algorithme – exercice 14

Ecrire l'algorithme permettant l'affichage d'un menu proposant de calculer la différence, ou le produit, ou la somme de deux nombres.

MÉTHODE

Cet algorithme se rapproche de l'algorithme 10. Nous utilisons deux objets de type réel (a, b) et un de type entier (choix). Nous affichons le menu suivant :

> Nous vous proposons de calculer :
> 1 – la somme de ces deux nombres;
> 2 – le produit de ces deux nombres;
> 3 – la différence de ces deux nombres.

Nous avons à contrôler que l'opérateur a bien répondu une des trois valeurs autorisées, sinon nous réaffichons le menu jusqu'à ce que la réponse soit valide. En fonction de la réponse obtenue, nous réalisons a + b, ou a * b, ou a – b.

```
programme menu1
var choix :  entier
    a, b : réels
DEBUT
   (* saisie des nombres *)
   afficher "entrez deux nombres "
   saisir a, b
   (* répétitive d'affichage du menu *)
   REPETER
     (* affichage du menu *)
     afficher "nous vous proposons de calculer "
     afficher "1 - la somme de ", a, " et de ", b
     afficher "2 - le produit de ", a, " et de ", b
     afficher "3 - la différence de ", a, " et de ", b
     afficher "votre choix "
     saisir choix
   JUSQU'A choix > 0 et choix < 4
   (* calcul du résultat *)
   SUIVANT choix FAIRE
```

```
        1 : afficher "la somme de ", a, " et de ", b , " est ", a+b
        2 : afficher "le produit de ", a, " et de ", b, " est ", a*b
        3 : afficher "la différence de ", a, " et de ", b, " est ", a - b
      FINSUIVANT
FIN
```

VI – La structure POUR

FORMAT GÉNÉRAL

```
POUR <identificateur> ← <valeur initiale> JQA  <valeur finale>
                                           PAS DE <incrément>
        <action>
FPOUR
```

Cette structure permet de répéter une action un nombre connu de fois.

L'identificateur est du type entier. La valeur initiale et la valeur finale sont des variables (ou constantes) de type entier.

L'incrément est la valeur d'augmentation progressive de l'identificateur. La valeur par défaut du pas est de 1 (auquel cas nous ne le précisons pas).

Si la valeur de l'incrément est positive et la valeur initiale strictement supérieure à la valeur finale, ou bien, si la valeur de l'incrément est négative et la valeur initiale strictement inférieure à la valeur finale, alors l'action définie dans la structure POUR n'est pas exécutée.

Si la valeur initiale est égale à la valeur finale, l'action définie est exécutée une seule fois.

La structure POUR est une simplification de la structure suivante :

```
<identificateur> ← <valeur initiale>
TANT QUE <identificateur>    < =    <valeur finale> FAIRE
         <action>
         <identificateur> ← <identificateur> + <incrément>
FTQ
```

❑ Exemple d'application 9

Nous voulons afficher la table de multiplication par 9.

MÉTHODE

Remarquons que la table de multiplication par 9 est construite de la manière suivante :

```
1  *  9  =  9
2  *  9  =  18
3  *  9  =  27        donc de la forme générale
4  *  9  =  36        variable1 * 9
5  *  9  =  45        avec variable1 qui varie de 1 en 1 jusqu'à 10
.....
10 *  9  =  90
```

```
programme tabledemultiplication
var variable1 : entiers
DEBUT
   POUR variable1 ← 1 JQA 10
        afficher variable1, " * 9 = ", variable1 * 9
   FPOUR
FIN
```

Algorithme – exercice 15

Ecrire l'algorithme qui compte le nombre de voyelles d'un mot saisi au clavier.

MÉTHODE

Il s'agit de traiter le mot caractère par caractère à l'aide de la fonction SSCHAINE. Nous commençons à la première lettre du mot pour s'arrêter à la dernière située à la position LONGUEUR(mot). Pour chacune de ces lettres, nous testons si c'est une voyelle à l'aide de la fonction RANG. RANG("aeiouy",lettre,1) rend la valeur 0 si la lettre n'est pas une voyelle et une valeur différente de 0 sinon.

```
programme comptevoyelle
var mot, lettre : caractères
    nbvoy, i : entier
DEBUT
   (* saisie du mot *)
   afficher "entrez un mot"
   saisir mot
   (* initialisation du compteur de voyelles *)
   nbvoy ← 0
   (* boucle de recherche des voyelles *)
   POUR i ← 1 JQA LONGUEUR(mot)
        (* extraction d'une lettre *)
        lettre ← SSCHAINE(mot,i,1)
        (* la lettre est-elle une voyelle ? *)
        SI RANG("aeiouy",lettre,1) > 0
        ALORS nbvoy ← nbvoy + 1
        FSI
   FPOUR
   (* affichage du résultat *)
   afficher "le nombre de voyelles du mot est ", nbvoy
FIN
```

VII – Exercices de synthèse

Algorithme – exercice 16

Ecrire l'algorithme permettant l'affichage des dix tables de multiplication.

MÉTHODE

Nous reprenons l'algorithme exercice 15, et nous le complétons par une structure POUR supplémentaire permettant de refaire 10 fois le même travail.

```
programme touttable
var i, j : entiers
DEBUT
   (* structure POUR concernant les dix tables *)
   POUR i ← 1 JQA 10
        (* structure POUR concernant chacune des tables *)
        afficher "table de multiplication de ",i
        POUR j ← 1 JQA 10
         afficher j, " * ",i, " = ", i*j
        FPOUR
   FPOUR
FIN
```

■ Algorithme – exercice 17

Ecrire un algorithme qui permet de convertir en base deux un nombre exprimé en base dix.

MÉTHODE

Nous opérons par divisions successives par 2. Nous recommençons le traitement jusqu'à ce que le quotient soit nul. Nous reconstituons le nombre en base deux en concaténant à gauche les restes successifs convertis en chaîne de caractères.

Nous contrôlons que le nombre saisi est positif.

```
programme basedeux
var  nb1, reste, svgd : entiers
     base2 : caractères
DEBUT
   REPETER
     (* saisie du nombre à convertir *)
     afficher "entrez un nombre positif ou nul"
     saisir nb1
     svgd ← nb1
   JUSQU'A nb1   >= 0
   (* initialisation de la chaîne base2 *)
   base2 ← " "
   (* répétitive des divisions successives par deux *)
   REPETER
     reste ← nb1 mod 2
     base2 ← cvchaîne(reste) // base2
     nb1 ← nb1 div 2
   JUSQU'A nb1 = 0
   (* affichage du résultat *)
   afficher svgd, "convertit en base deux : ",base2
FIN
```

■ Algorithme – exercice 18

Ecrire un algorithme qui compte les occurrences d'une lettre dans un mot.

MÉTHODE :

Nous utilisons :
– deux objets de type chaîne de caractères (mot, lettre)
– trois objets de type entier (compteur, pos, long)
Nous saisissons le mot et le caractère.
Nous initialisons compteur, pos et long à leur valeur respective de départ.
Nous analysons le mot à partir du premier caractère. A l'aide de la fonction RANG, nous repérons la position de la première occurrence du caractère dans le mot. Nous recommençons la même opération à partir de cette position jusqu'à ce que l'on atteigne la longueur du mot. Pour le cas où le caractère est absent du mot, la fonction rang restitue la valeur 0. D'où les deux conditions d'arrêt du traitement :

– nous atteignons la longueur du mot
– la fonction rang restitue la valeur 0.

Attention, si le caractère figure en dernière position du mot, la fonction RANG restitue la valeur longueur(mot) (différente de 0) et nous n'avons pas dépassé la longueur du mot. Nous affectons alors pos de la valeur longueur(mot)+1. Mais la fonction RANG ne peut pas être évaluée. Aussi, une possibilité consiste à utiliser une variable booléenne affectée du résultat de la comparaison entre la fonction RANG et le caractère.

```
programme occurrence
var mot, lettre : caractères
    compteur, pos, long : entiers
    absent : booléen
DEBUT
   (* saisie du mot et du caractère *)
   afficher "entrez un mot et un caractère"
   saisir mot, lettre
   (* initialisation du compteur *)
   compteur ← 0
   (* calcul de la longueur du mot *)
   long ← longueur(mot)
   (* initialisation du booléen d'absence du caractère dans le
    mot et de pos *)
   absent ← rang(mot, lettre, 1) = 0
   pos ← rang (mot, lettre, 1) + 1
   (* itérative de recherche des autres occurrences *)
   TANT QUE non absent et pos < long
     (* incrémentation du compteur *)
     compteur ← compteur + 1
     (* mise à jour de la variable booléenne *)
     absent ← rang(mot, lettre, pos) = 0
     (* calcul de la nouvelle position de recherche *)
     pos ← rang(mot, lettre, pos) + 1
   FTQ
   SI non absent
   ALORS (* cas particulier : la lettre est le dernier caractère du mot *)
        compteur ← compteur + 1
   FSI
   (* affichage du résultat *)
   afficher "il y a ",compteur,"occurrence(s) de ",lettre,"dans ",mot
FIN
```

Algorithme – exercice 19

Nous désirons écrire un algorithme qui calcule la position d'une lettre dans un mot. Si la lettre est absente, nous donnons 0 comme valeur. (Nous reprogrammons en fait la fonction RANG.)

Nous présentons deux approches possibles :

– nous disposons de la fonction LONGUEUR.

– nous ne disposons pas de la fonction LONGUEUR.

MÉTHODE

Première approche

Un mot est considéré comme une suite finie de caractères. Nous analysons le mot caractère par caractère et nous les comparons au caractère que l'on recherche. Le problème consiste à ne pas dépasser la longueur du mot. Pour analyser le mot caractère par caractère, nous utilisons la fonction SSCHAINE et un compteur POS qui nous permet de nous déplacer de 1 en 1 sur le mot :

```
SSCHAINE(mot,POS,1).
```

Nous arrêtons le traitement :
– lorsque nous avons trouvé le caractère
– lorsque nous avons atteint la longueur du mot.

Le traitement consiste alors à incrémenter le compteur POS. La structure utilisée est une itérative, car le caractère recherché peut être le premier du mot, l'entrée dans la boucle n'est pas nécessaire dans ce cas, le programme est fini puisque nous avons trouvé la position de la lettre. La condition pour poursuivre la boucle est :

```
SSCHAINE(mot,pos,1) <> car et longueur(mot) > pos
```

Il convient de faire très attention à la condition exprimée après l'itérative. Si le caractère n'est pas dans le mot, au dernier passage pos vaut longueur(mot) et la boucle doit être arrêtée car la fonction SSCHAINE ne peut pas être évaluée pour pos égal à longueur(mot)+1.

Enfin, lorsque nous quittons l'itérative, il faut savoir pour laquelle des deux raisons nous l'avons fait de manière à poursuivre le traitement adéquat.

```
programme aveclongueur
var mot, car : caractères
    pos, long : entiers
DEBUT
   (* saisie du mot et du caractère *)
   afficher "entrez un mot et un caractère"
   saisir mot, car
   (* initialisation du compteur *)
   pos ← 1
   (* calcul de la longueur du mot *)
   long ← LONGUEUR(mot)
   (* itérative de recherche du caractère *)
   TANT QUE SSCHAINE(mot, pos, 1) <> car et long > pos FAIRE
```

```
        (* incrémentation du compteur *)
        pos ← pos + 1
      FTQ
      (* test sur raison d'arrêt de la boucle *)
      SI SSCHAINE(mot, pos, 1) <> car
      ALORS pos ← 0
      FSI
      (* affichage du résultat *)
      afficher "lettre", car, "est à la position",pos,"dans le mot",mot
FIN
```

Deuxième approche

La méthode ne diffère pas beaucoup. Nous ne repérons pas la fin du mot avec la fonction LONGUEUR. Nous délimitons le mot par un espace à l'aide de la fonction de concaténation. Le traitement s'arrête :
– lorsque nous avons trouvé le caractère
– lorsque le caractère sur lequel nous sommes positionné est l'espace.

Une possibilité consiste à utiliser une variable booléenne, affectée du résultat de la comparaison de SSCHAINE(mot,pos,1) et car :

```
TROUVE ← SSCHAINE(mot,pos,1) = car
```

si l'égalité est vérifiée, la proposition à droite de l'affectation est vraie, et trouve est affecté de .vrai., sinon la proposition est fausse et trouve est affecté de .faux.

```
programme sanslongueur
(* la présence de la variable supplémentaire carcou, caractère
courant, évite d'utiliser deux fois la fonction sschaîne pour extraire
le même caractère (lors du test avec l'espace et lors de l'affectation
du booléen) *)
var carcou, mot, car : caractères
    pos : entier
    trouve : booléen
DEBUT
   (* saisie du mot et du caractère *)
   afficher "entrez un mot et un caractère"
   saisir mot, car
   (* ajout d'un espace pour délimiter la fin du mot)
   mot ← mot // " "
   (* initialisation du compteur *)
   pos ← 1
   (* initialisation du prochain caractère à traiter, caractère
   courant carcou *)
   carcou ← sschaîne(mot,pos,1)
   (* initialisation du booléen *)
   trouve ← carcou = car
   (* itérative de recherche du caractère *)
   TANT QUE non trouve et carcou <> " " FAIRE
     (* incrémentation du compteur *)
     pos ← pos + 1
```

```
    (* affectation de carcou et trouve *)
    carcou ← SSCHAINE(mot,pos,1)
    trouve ← carcou = car
  FTQ
  (* test sur raison d'arrêt de la boucle *)
  SI    carcou = " "
  ALORS pos ← 0
  FSI
  (* édition du résultat *)
  afficher "lettre" ,car, "est à la position ",pos, "dans le mot ",mot
FIN
```

2

Les tableaux

I – Les tableaux à une dimension

1 - APPROCHE DÉDUCTIVE

■ Algorithme – exercice 1

Ecrire un algorithme permettant de :
– saisir les noms de quatre élèves;
– calculer leurs moyennes respectives sachant que chaque élève a exactement 10 notes;
– éditer un état récapitulatif sous la forme :

nom 1er élève	nom 2ème	nom 3ème	nom 4ème
moyenne 1er	moyenne 2ème	moyenne 3ème	moyenne 4ème

MÉTHODE

Nous savons calculer la moyenne d'un élève à partir de la saisie de ses notes. Il suffit de saisir une à une les notes, de les cumuler puis d'effectuer la division par le nombre de notes.

Nous distinguons trois parties à l'algorithme :

• Initialisations des noms des élèves : nous utilisons quatre variables de type chaîne de caractères (nom1, nom2, nom3, nom4). Ces variables contiennent le même type d'information (un nom) mais différent par leur valeur.

• Calcul des moyennes : nous utilisons quatre variables de type réel (moy1, moy2, moy3, moy4). De même, ces variables diffèrent uniquement par leurs contenus. Nous utilisons également trois variables intermédiaires NBNOTE, NOTE, SOMMENOTE, afin de permettre le calcul des différentes moyennes.

• Edition de l'état récapitulatif.

Les deux premières parties doivent être exécutées quatre fois :

• Initialisation du nom du 1er élève
2ème élève
3ème élève
4ème élève

• Calcul de la moyenne du 1er élève
2ème élève
3ème élève
4ème élève

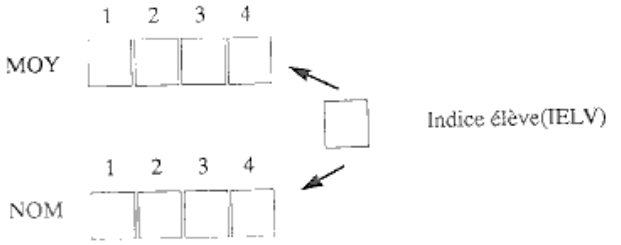
Le travail est à chaque fois identique, seules les variables de stockage changent. Nous utilisons un indicateur IELV qui nous permettra de sélectionner les informations de l'élève traité.

```
programme editmoyenne-v1
var ielv, nbnote, note, sommenote : entiers
   nom1, nom2, nom3, nom4 : caractères
   moy1, moy2, moy3, moy4 : réels
début
   (* initialisation des noms *)
   POUR ielv ← 1 JQA 4
        afficher "entrez le nom du ",ielv,"élève"
        SUIVANT ielv FAIRE
                1 : entrer nom1
                2 : entrer nom2
                3 : entrer nom3
                4 : entrer nom4
        FINSUIVANT
   FPOUR
   (* calcul des moyennes *)
   POUR ielv ← 1 JQA 4
        afficher "entrer les notes de "
        SUIVANT ielv FAIRE
                1 : afficher nom1
                2 : afficher nom2
                3 : afficher nom3
                4 : afficher nom4
        FINSUIVANT
        (* calcul de la moyenne *)
        sommenote← 0
        POUR nbnote ← 1 JQA 10
                afficher "entrez la note n° ", nbnote,
                entrer note
                sommenote ← sommenote + note
        FPOUR
        (* affectation de cette moyenne *)
        SUIVANT ielv FAIRE
                1 : moy1 ← sommenote / 10
```

```
                2  :  moy2 ← sommenote / 10
                3  :  moy3 ← sommenote / 10
                4  :  moy4 ← sommenote / 10
          FINSUIVANT
      FPOUR
      (* édition de l'état *)
      afficher nom1, nom2, nom3, nom4
      afficher moy1, moy2, moy3, moy4
  fin
```

INCONVÉNIENTS

Nous constatons qu'une classe est rarement composée de 4 élèves. Aussi, l'algorithme s'alourdit considérablement lorsque l'on augmente le nombre d'élèves, les quatre moyennes et les quatre noms n'ayant pas de lien physique commun. Il serait souhaitable de disposer d'un objet plus complexe, permettant de stocker ces moyennes (ou ces noms), auquel on accéderait à l'aide d'un compteur (tel que ielv). C'est-à-dire :



DÉFINITION

Un tableau est une structure de donnée linéaire qui permet de stocker des données de même type. Chacune des valeurs est repérée par un indice indiquant la position de la donnée dans le tableau.

FORMAT GENERAL

Un tableau est déclaré comme un type particulier de donnée. Nous lui donnons un nom, une valeur d'indice minimale et une valeur d'indice maximale correspondant au nombre maximal de cases le composant. Nous déclarons également l'indice qui permet d'adresser les différentes cases. L'indice doit obligatoirement être du type entier.

```
<nom-du-tableau> (<indice-min> : <indice-max>) : TABLEAU DE <type-donnée>
```

Exemple 1 :

Les moyennes des dix élèves d'une classe sont stockées dans un tableau linéaire suivant :

	1	2	3	4	5	6	7	8	9	10
MOY	5	12	14	7,5	10	9,5	13	8	6	10

le premier élève a comme moyenne : MOY(1) = 5
le deuxième élève a comme moyenne : MOY(2) = 12
le troisième élève a comme moyenne : MOY(3) = 14
le quatrième élève a comme moyenne : MOY(4) = 7,5
le cinquième élève a comme moyenne : MOY(5) = 10
le sixième élève a comme moyenne : MOY(6) = 9,5
le septième élève a comme moyenne : MOY(7) = 13
le huitième élève a comme moyenne : MOY(8) = 8
le neuvième élève a comme moyenne : MOY(9) = 6
le dixième élève a comme moyenne : MOY(10) = 10

Nous déclarons ce tableau de la manière suivante :
moy(1:10) : TABLEAU de réels

❑ Exemple d'application 1

Soit le tableau de valeurs suivant :

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
TAB	11	12	11	10	15	12	13	16	17	18	20	13	15	12	17

1) Quelle est la valeur de l'indice minimal ?
2) Quelle est la valeur de l'indice maximal ?
3) A quel indice se trouve la valeur 18 ?
4) A quels indices se trouve la valeur 12 ?
5) Quelle valeur se trouve à l'indice 12 ?
6) Quelle est la moyenne des 3 valeurs situées aux indices 7, 9 et 13 ?
7) A quel indice se trouve la plus petite valeur ?
8) A quel indice se trouve la plus grande valeur ?

RÉPONSES :

1) Indice minimal : 1
2) Indice maximal : 15
3) La valeur 18 se trouve à l'indice 10 : tab(10) = 18
4) La valeur 12 se trouve aux indices 2, 6 et 14 : tab(2) = tab(6) = tab(14) = 12
5) A l'indice 12, nous trouvons la valeur 13 : tab(12) = 13
6) A l'indice 7, nous trouvons la valeur 13 tab(7) = 13
   A l'indice 9, nous trouvons la valeur 17 tab(9) = 17
   A l'indice 13, nous trouvons la valeur 15 tab(13) = 15
   d'où la somme 45

et la moyenne

7) La plus petite valeur est 10 et se trouve à l'indice 4 : tab(10) = 4
8) La plus grande valeur est 20 et se trouve à l'indice 11 : tab(11) = 20

REMARQUE

Le nom d'un tableau n'est jamais utilisé seul, dans toutes les instructions (saisie, affichage, calcul, test), il est toujours suivi d'un indice, entouré de parenthèses.

2 – CRÉATION D'UN TABLEAU

La création d'un tableau consiste en un remplissage des différentes cases qui le constituent. Cette opération peut se faire de deux manières différentes :
– en renseignant les cases une à une à partir de la première;
– en adressant les cases directement, et ce dans un ordre quelconque.

Exemple 2

Une série de 10 nombres est saisie au clavier. Nous voulons les stocker dans un tableau pour pouvoir rechercher le plus petit et le plus grand d'entre eux.

L'ordre de saisie des nombres n'ayant aucune incidence pour la suite du traitement, nous stockons le premier nombre saisi dans la première case,
le deuxième nombre saisi dans la deuxième
...
le dixième nombre saisi dans la dixième

```
programme saisitableau
var valsai(1:10) : tableau d'entiers
   n : entier
début
   (* boucle de traitement *)
   POUR n ← 1 JQA 10
     (* saisie du nombre *)
     afficher "entrez un nombre "
     saisir valsai(n)
   FPOUR
fin
```

REMARQUE

Il est inutile d'utiliser une variable simple en intermédiaire de saisie

```
saisir nombre
valsai(n) ← nombre
```
équivalent à `saisir valsai(n)`

Si l'utilisateur veut saisir moins de 10 nombres, il est contraint de continuer.

Exemple 3 :

Nous désirons stocker dans un tableau les noms des élèves saisi de façon aléatoire. Nous mettons le nom d'un élève dans la case du tableau correspondant à son numéro : le nom de l'élève ayant le numéro 12 est stocké dans la case numéro 12 du tableau. Nous supposons que la classe contient exactement 35 élèves.

```
programme saisinom
var nom(1:35) : tableau de caractères
   numéro : entier
début
   (* boucle de traitement *)
   POUR i ← 1 JQA 35
     (* saisie du nom et du numéro de l'élève *)
     afficher "entrez son numéro "
     saisir numéro
     afficher "entrez le nom de l'élève "
     saisir nom(numéro)
   FPOUR
fin
```

■ Algorithme – exercice 2

Reprenons l'exemple 2 (saisie de 10 nombres), en ne contraignant pas l'utilisateur à saisir exactement 10 valeurs.

MÉTHODE

L'utilisateur peut saisir au maximum 10 nombres. Il nous indique qu'il veut s'arrêter en donnant la valeur zéro. La structure POUR ne peut donc plus être utilisée. Le traitement ne s'effectue plus un nombre précis de fois.

Les conditions d'arrêt du traitement sont :
– 10 nombres ont été saisis
– ou la valeur saisie est zéro.

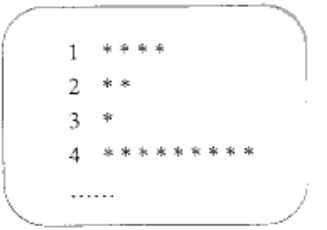
```
programme saisirépétitif
(* version répétitive du programme *)
var valsai(1:10) : tableau d'entiers
   taille : entier
début
   (* initialisation de l'indice *)
   taille ← 0
   REPETER
     taille ← taille + 1
     afficher "entrez un nombre (ou 0 pour arrêt) "
     saisir valsai(taille)
   JUSQU'A valsai(taille) = 0 ou taille = 10
   (* réajustement de la taille du tableau *)
   SI    valsai(taille) = 0
   ALORS taille ← taille - 1
   FSI
fin
```

REMARQUE :

Le choix de la valeur d'arrêt 0 ne nous autorise pas à saisir 0 comme nombre.

■ Algorithme – exercice 3

Soit une file de nombres compris entre 1 et 20, saisis au clavier. Nous voulons éditer un histogramme horizontal de la fréquence des nombres.



MÉTHODE

A chaque nouvelle saisie d'un nombre, il faut augmenter de 1 le nombre d'occurence lui correspondant. Comme chacun de ces nombres est obligatoirement compris entre 1 et 20, nous utilisons un tableau de 20 cases pour compter les occurences de chacun des nombres :

La première case contient le nombre de	1	saisis
la deuxième case contient le nombre de	2	saisis
.....	.....	
la dix-neuvième case contient le nombre de	19	saisis
la vingtième case contient le nombre de	20	saisis

Nous initialisons au préalable, à zero chacune des cases du tableau et nous controlons, à chaque nouvelle saisie que le nombre introduit est bien compris entre 0 et 20.

Pour éditer un histogramme, il nous faudra lire le contenu du tableau.

```
programme histogramme
var occurre(1:20) : tableau d'entiers
   nombre : entier
début
   (* initialisation du tableau *)
   POUR nombre ← 1 à 20
     occurre(nombre) ← 0
   FPOUR
   (* saisie du premier nombre *)
   afficher "entrez un nombre compris entre 1 et 20 (ou 0 pour
      arrêt)"
   saisir nombre
   (* contrôle de la saisie *)
   TANT QUE nombre < = 0 ou nombre > 20
     afficher "votre nombre est incorrect, veuillez le ressaisir "
     saisir nombre
   FTQ
```

```
    (* boucle de traitement des occurrences *)
    TANT QUE nombre <> 0
      (* mise à jour du tableau *)
      occurre(nombre) ← occurre(nombre) + 1
      (* saisie des autres nombres *)
      afficher "entrez un nombre compris entre 1 et 20 (ou 0 pour
           arrêt)"
      saisir nombre
      (* contrôle de la saisie *)
      TANT QUE nombre < = 0 ou nombre > 20
          afficher "votre nombre est incorrect, veuillez le ressaisir "
          saisir nombre
      FTQ
    FTQ
    (* édition de l'histogramme *)
    POUR nombre ← 1 JQA 20
      afficher " nombre de " nombre;
      (* édition des * correspondantes au nombre sans retour à la ligne *)
      POUR j ← 1 JQA occurre(nombre)
          afficher "*";
      FPOUR
      (* nous provoquons un retour à la ligne *)
      afficher " "
    FPOUR
fin
```

REMARQUE

Le ";" après un ordre d'affichage évite le retour à la ligne.

L'utilisation d'une structure itérative et non répétitive permet d'obtenir un algorithme correct y compris lorsque le premier nombre saisi vaut zéro.

3 – ÉDITION D'UN TABLEAU

Nous parcourons les différentes cases du tableau en faisant varier l'indice et nous affichons leur contenu au fur et à mesure.

Algorithme – exercice 4

Ecrire l'algorithme permettant d'éditer le contenu du tableau suivant :

a	b	d	e	g	h	a	p	e	f	g	n	v	c	m

MÉTHODE

- Nous déclarons un objet de type tableau de 15 cases contenant chacune un caractère (nous l'appelons LETTRE).
- Nous saisissons le contenu du tableau.
- Nous éditons chaque case l'une après l'autre, en commençant par la première case :

```
afficher LETTRE(1)
afficher LETTRE(2)
afficher LETTRE(3)      <=>     afficher LETTRE(N)
  ...                           avec N variant de 1 en 1 jusqu'à 15
afficher LETTRE(15)
```

```
programme éditiontableau
var  lettre(1:15) : tableau de caractères
   n : entier
début
   (* saisie du tableau LETTRE *)
   (* cf algorithme exercice 2 *)
   (* boucle de traitement *)
   POUR n ← 1 JQA 15
     afficher lettre(n)
   FPOUR
fin
```

4. – DIFFÉRENTES OPÉRATIONS POSSIBLES SUR LES TABLEAUX

Sur la structure de données de type tableau, il est possible de faire plusieurs sortes de traitements, comme par exemple la recherche du minimum ou du maximum, la somme des cases, le produit, la moyenne… Nous allons représenter sous forme algorithmique les principaux. Pour l'ensemble de ces algorithmes, nous travaillons sur un tableau de 35 cases numériques. Nous l'appelons VALEURS.

■ Algorithme – exercice 5

Calculez la somme des nombres contenus dans VALEURS.

MÉTHODE

Nous ajoutons un à un le contenu des cases, depuis la première jusqu'à la trente-cinquième. Nous utilisons un objet de type entier, SOMME, qui reçoit le cumul intermédiaire.

```
programme valeursomme
var  valeurs(1:35) : tableau d'entiers
   somme, i : entiers
début
   (* saisie du tableau VALEURS *)
   (* cf algorithme exercice 2 *)
   (* initialisation de somme *)
   somme ← 0
   (* boucle de calcul des sommes intermédiaires *)
   POUR i ←  1 JQA 35
     somme ← somme + valeurs(i)
   FPOUR
   (* édition du résultat *)
   afficher "la somme de : "
```

```
    POUR i ← 1 JQA 34
      afficher valeurs(i)," + " ;
    FPOUR
    afficher valeurs(35)," = ", somme
fin
```

■ Algorithme – exercice 6

Calculez le produit et la moyenne des nombres stockés dans VALEURS.

MÉTHODE

Nous reprenons la même méthode que pour l'algorithme précédent, en effectuant cette fois le produit et la somme que nous diviserons par 35 pour obtenir la moyenne.

```
programme valeurproduit
var  valeurs(1:35) : tableau d'entiers
   produit, somme, i : entiers
début
   (* saisie du tableau VALEURS *)
   (* cf algorithme exercice 2 *)
   (* initialisation du produit et de la somme *)
   produit ← 1
   somme ← 0
   (* boucle de calcul des produits intermédiaires *)
   POUR i ← 1 JQA 35
     produit ← produit * valeurs(i)
     somme ← somme + valeurs(i)
   FPOUR
   (* affichage du résultat *)
   afficher "le produit de : "
   POUR  i ← 1 JQA 34
     afficher valeurs(i)," * " ;
   FPOUR
   afficher valeurs(35)," = ", produit
   afficher "la moyenne de ces nombres est ", somme / 35
fin
```

■ Algorithme – exercice 7

Recherchez le plus petit élément du tableau VALEURS

MÉTHODE

Nous réutilisons le principe de l'algorithme - exercice 11 de la première partie :
– Supposons que la première case contienne le minimum relatif.
– Comparons le contenu de la deuxième case avec le minimum relatif. Si celui-ci est inférieur, il devient le minimum relatif.
– Renouvelons la même opération avec les 33 cases suivantes.

```
programme minivaleurs
var  valeurs(1:35):tableau d'entiers
     i, min : entiers
début
   (* saisie du tableau VALEURS *)
   (* cf algorithme - exercice 2 *)
   (* initialisation du minimum relatif *)
   min ← valeurs(1)
   (* boucle de recherche du minimum absolu *)
   POUR  i ← 2 JQA 35
     SI  valeurs(i) < min
     ALORS (* changement du minimum relatif *)
           min ← valeurs(i)
     FSI
   FPOUR
   (* affichage du minimum relatif *)
   afficher "le minimum relatif des nombres suivants "
   POUR  i ← 1 JQA 35
   afficher valeurs(i);
   FPOUR
   afficher " est " ,min
fin
```

■ Algorithme – exercice 8

Recherchez le plus grand élément du tableau VALEURS

METHODE

Nous appliquons le même principe que précédemment, mais la comparaison porte sur la supériorité.

```
programme maxvaleur
var  valeurs [illegible] tableau d'entiers
     [illegible]
début
   (* saisie du tableau VALEURS *)
   (* cf algorithme - exercice 2 *)
   (* initialisation du maximum relatif *)
   max ← valeurs (1)
   (* boucle de recherche du maximum absolu *)
   POUR  i ← 2 JQA 35
     SI  valeurs(i) > max
     ALORS (* changement du maximum relatif *)
           max <- valeurs(i)
     FSI
   FPOUR
   (* affichage du maximum absolu *)
   afficher "le maximum des nombres suivants "
   POUR  i ← 1 JQA 35
```

```
        afficher valeurs(i);
    FPOUR
    afficher " est " ,max
fin
```

REMARQUE :

Supposons que l'on veuille rechercher simultanément le maximum et le minimum. Deux possibilités sont offertes :
– l'enchaînement des deux structures POUR, coûteux en temps
– l'utilisation d'une seule structure POUR :

```
min ← valeur(1)
max ← valeur(1)
POUR  i ← 2 JQA 35
      SI    min > valeur(i)
      ALORS min ← valeur(i)
      SINON SI max < valeur(i)
            ALORS max ← valeur(i)
            FSI
      FSI
FPOUR
```

■ Algorithme – exercice 9

Ecrire l'algorithme permettant d'effectuer la somme de deux tableaux de même dimension et de stocker le résultat dans un troisième tableau utilisé pour des calculs supplémentaires :

	1	12	-1	4	-4	6	7
+	4	-12	5	6	-7	9	8
	5	0	4	10	-11	15	15

TAB 1
TAB 2
TAB 3

MÉTHODE

```
tab3(1)<- tab1(1)  +  tab2(1)
tab3(2)<- tab1(2)  +  tab2(2)
tab3(3)<- tab1(3)  +  tab2(3)          tab3(n) <- tab1(n) + tab2(n)
tab3(4)<- tab1(4)  +  tab2(4)   <=>    avec n variant de 1 en 1
tab3(5)<- tab1(5)  +  tab2(5)          jusqu'à 7
tab3(6)<- tab1(6)  +  tab2(6)
tab3(7)<- tab1(7)  +  tab2(7)
```

```
programme [illegible]
var tab1[illegible],tab3(1:7) : tableau d'entiers
   n : [illegible]
début
   (* saisie des tableaux TAB1 et TAB2 *)
   (* cf algorithme - exercice 2 *)
   (* boucle : addition case par case *)
   POUR n←1 [illegible]
     tab3(n) ← tab1(n) + tab2(n)
   FPOUR
   (* édition [illegible] *)
   POUR n←1 [illegible]
     afficher tab1(n), " + ", tab2(n), " = " , tab3(n)
   FPOUR
fin
```

■ PROBLEME 1

Soit un tableau contenant les précipitations atmosphériques des 6 derniers mois (printemps-été) de chacun des départements, et un tableau contenant le nom des départements.

1 – On déclare sinistrés les départements dont la valeur est inférieure à un seuil. Ecrire l'algorithme permettant, à partir de la saisie de ce seuil, d'afficher les noms des départements sinistrés.

DEPT	AIN	AISNE	...	ARIEGE	AUBE	AUDE	AVEYRON	...
PRECIP	122	230	...	50	170	185	140	...

Par exemple, si seuil = 150 l'Ain, l'Ariège et l'Aveyron sont sinistrés.

2 – Soit un tableau complémentaire indiquant, pour chacun des départements, sa principale culture et donc son besoin en eau sous forme d'un nombre compris entre 1 et 5 (5 codes de cultures principales ayant été recensées). A partir de la saisie des 5 seuils de précipitations associés aux 5 codes de cultures, écrire l'algorithme permettant d'afficher le nom des départements sinistrés.

CULTURE	4	3	...	1	2	1	5	...

SEUIL	60	100	150	200	300

Avec ces valeurs de seuil, l'Ain, code culture 4, seuil 200, sinistré
l'Aisne, code culture 3, seuil 150, non sinistré
l'Ariège et l'Aveyron le sont, l'Aube et l'Aude non.

MÉTHODE

1 – Nous utilisons deux tableaux, DEPT, PRECIP, dont le contenu est saisi en début de programme.

Nous saisissons le seuil dans une variable SEUIL. Nous comparons SEUIL avec chacune des valeurs contenues dans les cases du tableau PRECIP. Nous utilisons un indice INDDEP que nous faisons varier de 1 en 1 jusqu'à 99. Pour chaque valeur de inddep, nous effectuons la comparaison entre seuil et precip(inddep) puis recherchons si besoin le nom du département sinistré dans le tableau dept à l'indice inddep

```
programme précipitation
var  dept(1:99) : tableau de caractères
     précip(1:99) : tableau d'entiers
     inddep,seuil : entiers
début
   (* saisie des noms et des précipitations départements *)
   POUR  inddep ← 1 JQA 99
     afficher "entrez le nom du département ayant le code ", inddep
     saisir dept(inddep)
     afficher "entrez le total de ses précipitations "
     saisir précip(inddep)
   FPOUR
   (* saisie de la valeur du seuil sinistre *)
   afficher "entrez le seuil en deçà duquel une région est
             sinistrée"
   saisir seuil
   (* comparaison avec les départements *)
   afficher "liste des départements sinistrés : "
   POUR  inddep ← 1 JQA 99
     SI  seuil > précip(inddep)
     ALORS afficher dept(inddep)
     FSI
   FPOUR
fin
```

2 – Nous devons saisir le tableau des cultures par département, ainsi que les valeurs des 5 seuils.

Nous remarquons que la valeur du seuil correspondant à la culture pratiquée dans une région donnée (exemple AIN: culture(1)=4) se trouve à l'indice correspondant à cette culture dans le tableau seuil (exemple : seuil(culture(1))=200).

Nous devons donc comparer : precip(inddep) et seuil(culture(inddep)) en faisant varier inddep de 1 en 1 jusqu'à 99

```
programme culture
var  dept(1:99) : tableau de caractères
     seuil(1:5), culture(1:99), précip(1:99) : tableau d'entiers
     inddep, indseuil : entiers
début
   (* saisie des noms, précipitations et culture des départements *)
   POUR  inddep ← 1 JQA 99
     afficher "entrez le nom du département ayant pour code ", inddep
```

```
        saisir dept(inddep)
        afficher "entrez ses précipitations"
        saisir précip(inddep)
        REPETER
            afficher "entrez son code de principale culture (entre 1 et 5)"
            saisir culture(inddep)
        JUSQU'A culture(inddep) > 0 et culture(inddep) < 6
      FPOUR
      (* saisie des seuils associés aux cultures *)
      POUR  indseuil ← 1 JQA 5
        afficher "entrez le seuil associé à la culture ", indseuil
        saisir seuil(indseuil)
      FPOUR
      (* édition de la liste *)
      POUR  inddep ← 1 JQA 99
        SI  seuil(culture(inddep)) > precip(inddep)
        ALORS afficher "le département ",dept(inddep), "est sinistré"
              afficher "type de culture ", culture(inddep)
        FSI
      FPOUR
fin
```

5 - RECHERCHE D'UN ÉLÉMENT DANS UN TABLEAU

Pour rechercher un élément dans un tableau, nous disposons de deux solutions :
– Soit nous connaissons l'indice de la case où il figure. Il suffit alors de vérifier que le contenu est bien celui que l'on recherche.
– Soit nous connaissons le contenu de la case mais pas son indice.

Il paraît évident que la première recherche est très simple, aussi nous nous attachons à présenter la deuxième à l'aide de trois algorithmes.

■ Algorithme – exercice 10

Recherchez dans un tableau contenant 20 noms de chanteurs, un nom saisi au clavier.

MÉTHODE

Nous saisissons le nom à rechercher dans la variable NOM. Nous appelons CHANTEURS le tableau des noms de chanteurs. Le traitement de recherche commence à la première case du tableau et s'arrête au plus tard à la vingtième.

Il y a deux conditions d'arrêt possibles du traitement :

– soit nous avons atteint la vingtième case;

– soit nous avons trouvé le chanteur.

Attention, le nom peut se trouver à la vingtième case !!!

```
programme recherchechanteur
var  chanteur(1:20) : tableau de caractères
     i : entier
     nom : caractères
début
   (* saisie du tableau CHANTEUR *)
   (* cf algorithme- exercice 2 *)
   (* saisie du nom à rechercher *)
   afficher "entrez un nom de chanteur "
   saisir nom
   (* initialisation de l'indice *)
   i ← 0
   (* recherche de ce chanteur *)
   REPETER
    i ← i + 1
   JUSQU'A chanteur(i) = nom ou i = 20
   (* affichage du résultat *)
   SI    chanteur(i) = nom
   ALORS afficher"bravo, nous avons les mêmes goûts en matière
                  de musique"
   SINON afficher"je n'apprécie pas ce genre de musique"
   FSI
fin
```

■ Algorithme – exercice 11

Rechercher le nombre d'occurences d'un prénom saisi au clavier à l'intérieur d'un tableau de 50 prénoms.

MÉTHODE

Contrairement à l'exercice précédent, nous ne nous arrêtons pas lorsque nous rencontrons pour la première fois le prénom dans le tableau. En effet, il s'agit de compter le nombre de fois où le prénom est présent dans le tableau, nous utilisons un compteur appelé COMPTEUR, que nous incrémentons de 1 à chaque nouvelle occurrence.

Nous saisissons un prénom à rechercher dans la variable PRENOMCHER. Nous contrôlons que cette variable a bien été valorisée.

```
programme occurrence
var  prénom(1:50) : tableau de caractères
   compteur : entier
   prénomcher : caractères
début
   (* saisie du tableau PRÉNOM *)
   (* cf algorithme- exercice 2 *)
   (* initialisation du compteur *)
   compteur ← 0
   (* saisie du prénom recherché *)
   afficher "entrez un prénom "
```

```
    saisir prénomcher
    TANT QUE   prénomcher <> " "
      afficher "attention, on vous demande un prénom "
      afficher "entrez un prénom"
      saisir prénomcher
    FTQ
    (* recherche du prénom dans le tableau *)
    POUR  i ← 1 JQA 50
      (* occurrence du prénom *)
      SI  prénom(i) = prénomcher
      ALORS     compteur ← compteur + 1
      FSI
    FPOUR
    (* affichage du résultat *)
    afficher "le prénom ",prénomcher," est présent ", compteur,"fois"
fin
```

Algorithme – exercice 12

L'ordinateur a en mémoire un mot de 5 lettres stocké dans un tableau. Il s'agit de le trouver. Pour cela, nous saisissons des mots de 5 lettres. et l'algorithme indique le nombre de lettres communes et bien placées.

L'utilisateur a la possibilité de s'arrêter en demandant la solution.

MÉTHODE

Nous arrêtons le traitement, soit :
- si le mot a été trouvé
- si l'utilisateur a demandé la solution.

Pour demander la solution, l'utilisateur donne le mot "SOL". Nous vérifions que le mot donné possède bien 5 lettres, ou est égal à SOL, sinon nous renouvelons la saisie du mot.

Pour compter le nombre de lettres communes au mot saisi et à celui de l'ordinateur, nous les comparons caractère par caractère et nous incrémentons un compteur à chaque égalité.

Le mot de l'ordinateur étant stocké dans un tableau, nous obtenons le contenu de chaque case à l'aide de l'indice qui varie de 1 en 1 jusqu'à 5. Nous le comparons avec le mot proposé par l'utilisateur et découpé en caractères à l'aide de la fonction sschaîne :

```
motàtrouver(1) = sschaîne(mot,1,1)
motàtrouver(2) = sschaîne(mot,2,1)          motàtrouver(i) = sschaîne(mot,i,1)
motàtrouver(3) = sschaîne(mot,3,1)  <=>     avec i variant de 1 en 1
motàtrouver(4) = sschaîne(mot,4,1)          jusqu'à 5
motàtrouver(5) = sschaîne(mot,5,1)
```

Pour faciliter l'expression de la condition d'arrêt du traitement, nous reconstituons le mot de l'ordinateur dans une variable de type chaîne de caractères MOTORDI. Lorsque les contenus de motordi et mot sont identiques, nous arrêtons le traitement.

```
programme jeu
var motàtrouver(1:5) : tableau de caractères
    mot, motordi : caractères
    compte, i : entiers
début
   (* saisie du tableau MOTÀTROUVER *)
   (* cf algorithme- exercice 2 *)
   (* pour conserver à cet algorithme son rôle de jeu, la
    saisie doit être faite par une autre personne que le
    joueur *)
   (* reconstitution du mot de l'ordinateur *)
   motordi ← " "
   POUR i ← 1 JQA 5
    motordi ← motordi // motàtrouver(i)
   FPOUR
   (* saisie du premier mot *)
   afficher "entrez un mot de 5 lettres (ou SOL pour arrêt) "
   saisir mot
   (* contrôle de la saisie *)
   TANT QUE longueur(mot) <> 5 et mot <> "SOL"
    afficher "votre mot n'a pas un nombre de lettres correct "
    afficher "entrez un mot de 5 lettres (ou SOL pour arrêt) "
    saisir mot
   FTQ
   (* boucle de recherche du mot de l'ordinateur *)
   TANT QUE mot <> motordi et mot <> "SOL"
    (* recherche des lettres communes *)
    (* initialisation du compteur de lettres communes et bien placées *)
    compte ← 0
    POUR i ← 1 JQA 5
        SI motàtrouver(i)=sschaîne(mot,i,1)
        ALORS compte ← compte + 1
        FSI
    FPOUR
    (* affichage du nombre de lettres communes *)
    afficher "vous avez ", compte, "lettres communes et bien placées"
    (* saisie des autres mots *)
    afficher "entrez un mot de 5 lettres (ou SOL pour arrêt)"
    saisir mot
    TANT QUE longueur(mot) <> 5 et mot <> "SOL"
        afficher "votre mot n'a pas un nombre de lettres correct"
        afficher "entrez un mot de 5 lettres (ou SOL pour arrêt)"
        saisir mot
    FTQ
   FTQ
   (* affichage des résultats *)
   SI    mot = "SOL"
   ALORS afficher "le mot à trouver était " motordi
   SINON afficher "BRAVO !!!"
   FSI
fin
```

6 - SUPPRESSION D'UN ÉLÉMENT DANS UN TABLEAU

La suppression d'un élément dans un tableau peut se faire de deux manières distinctes :
– en accédant directement à la case par le biais de l'indice
– en recherchant séquentiellement la case à supprimer à l'aide de son contenu.

De toute façon, nous devons décaler d'une case vers la gauche tous les éléments qui suivent la case à supprimer.

Exemple 3

Soit le tableau des enfants inscrits aux cours de tennis :

Nicolas	Mathilde	Arnaud	Juliette	Camille	Stan

Mettre à jour le tableau en tenant compte du fait qu'Arnaud a démissionné.

Nicolas	Mathilde	Juliette	Camille	Stan	

■ Algorithme – exercice 13

Ecrire l'algorithme correspondant à l'exemple 3 lorsque l'indice de la case à supprimer est connu.

MÉTHODE

Nous devons saisir l'indice de la case à supprimer en contrôlant que cette case existe.

Pour décaler les cases qui la suivent, il suffit de copier le contenu de la case à l'indice j+1 dans la case d'indice j, et de mettre à blanc la dernière qui a été décalée.

Attention, la case à supprimer est peut-être la dernière du tableau, auquel cas nous n'avons pas de décalage à effectuer.

```
programme décalage1
var enfant(1:6) : tableau de caractères
    i, indémission : entiers
début
   (* saisie du tableau enfant *)
   (* cf algorithme- exercice 2 *)
   (* saisie de l'indice du démissionnaire *)
   afficher "entrez l'indice du démissionnaire "
   saisir indémission
   (* contrôle de l'existence de cet indice *)
   TANT QUE indémission < 1 ou indémission > 6
     afficher " cette valeur est impossible "
     afficher "redonnez un numéro "
     saisir indémission
   FTQ
```

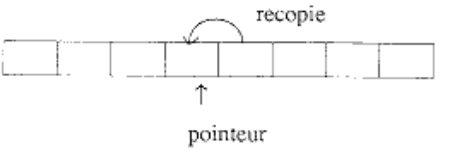
```
    (* initialisation du compteur *)
    i ← indémission
    (* décalage des valeurs supérieures *)
    TANT QUE i < 6
      enfant(i) ← enfant(i+1)
      i ← i + 1
    FTQ
      (* mise à blanc dernière case *)
    enfant(6) ← " "
fin
```

REMARQUE

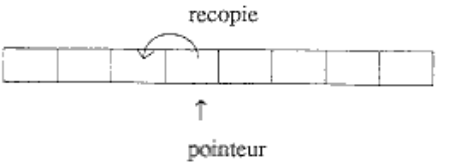
A force de supprimer des valeurs dans le tableau, celui-ci contient un certain nombre de cases vides, il n'est donc pas nécessaire de les décaler toutes. Un seul espace doit être transféré, il exécute la mise à blanc du dernier élément non vide du tableau. La condition d'arrêt du décalage devient alors :
– soit la case à effacer est vide (la dernière action itérative a donc transféré un espace);
– soit nous avons fini de traiter le tableau qui était plein.

```
(* début de l'algorithme identique jusqu'au décalage *)
i ← indémission
TANT QUE i < 6 et enfant(i) <> " "
      enfant(i) ← enfant(i+1)
      i ← i + 1
FTQ
(* cas particulier du tableau plein *)
SI i = 6
ALORS enfant(6) ← " "
FSI
```

Nous pouvons procéder d'une autre manière pour réaliser le décalage. En effet, dans cette première version, nous nous positionnons sur la case à supprimer et nous l'effaçons par le contenu de la case qui suit immédiatement.



Nous pouvons également nous positionner sur la case à déplacer et la recopier dans la case qui la précède.



```
(* début de l'algorithme identique jusqu'au décalage *)
i ← indémission
TANT QUE i < 6 et enfant(i) <> " "
     i ← i + 1
     enfant(i-1) ← enfant(i)
FTQ
SI i = 6
ALORS enfant(6) ← " "
FSI
```

Algorithme – exercice 14

Ecrire l'algorithme correspondant à l'exemple 3 lorsque le contenu de la case à supprimer est connu.

MÉTHODE

La seule différence avec l'exercice précédent est la recherche préalable de la case à supprimer grâce à son contenu.

```
programme décalagev2
var enfant(1:6) : tableau de caractères
    nomdémission : caractères
    i : entier
début
   (* saisie du tableau ENFANT *)
   (* cf algorithme - exercice 2 *)
   (* saisie du nom de l'enfant démissionnaire *)
   afficher "entrez le nom de l'enfant"
   saisir nomdémission
   (* recherche de l'indice de la case *)
   i ← 1
   TANT QUE i < 6 et enfant(i) <> nomdémission
     i ← i + 1
   FTQ
   SI    enfant(i) = nomdémission
   ALORS (* décalage des éléments d'indice supérieur *)
         TANT QUE i < 6 et enfant(i) <> " "
               enfant(i) ← enfant(i+1)
               i ← i + 1
         FTQ
         (* cas particulier du dernier *)
         SI i = 6
         ALORS enfant(i) ← " "
         FSI
   SINON afficher "ce nom ne figure pas dans la liste des inscrits"
   FSI
fin
```

II – Tri d'un tableau

Un tableau est ordonné lorsqu'il existe une relation d'ordre entre le contenu de ses différentes cases. Nous parlons de :
– tri croissant si le contenu de la case d'indice I est inférieur ou égal au contenu de la case d'indice I + 1.
– tri décroissant si le contenu de la case d'indice I est supérieur ou égal au contenu de la case d'indice I + 1.

Plusieurs méthodes de tri existent. Nous allons étudier les principales. Pour tous ces exercices, nous nous basons sur un tableau de 30 valeurs numériques VALNUM.

Algorithme – exercice 15

Tri croissant par recherche successive des minima.

MÉTHODE

Le principe est le suivant :
• recherche du minimum dans le tableau de 30 valeurs et échange du contenu des cases d'indice 1 et d'indice correspondant à la valeur du minimum.
• application du même principe sur 29 valeurs (30 – première), puis 28, puis 27, ..., jusqu'au traitement d'un tableau de deux cases.

Visualisation du traitement sur 4 valeurs :

tableau initial

8	1	7	5

après le premier passage : 4 valeurs : cases 1 à 4
minimum 1 indmin 2 échange des cases 1 et 2

1	8	7	5

après le second passage : 3 valeurs : cases 2 à 4
minimum 5 indmin 4 échange des cases 2 et 4

1	5	7	8

après le troisième passage : 2 valeurs : cases 3 à 4
minimum 7 indmin 3 non échange des cases 3 et 3

1	5	7	8

Nous pouvons réutiliser les principes énoncés dans l'algorithme – exercice 7.

```
programme trimin
var  valnum(1:30) : tableau d'entiers
     indrech, indval, indmin, inter : entiers
début
   (* saisie du tableau VALNUM *)
   (* application de l'algorithme - exercice 2 *)
   (* boucle de traitement *)
   POUR  indval ← 1 jqa 29
   (* recherche du minimum dans un tableau de 30- indval + 1 éléments *)
         indmin ← indval
         POUR  indrech ← indval + 1 JQA 30
               SI valnum(indrech) < valnum(indmin)
               ALORS indmin ← indrech
               FSI
         FPOUR
```

```
            (* échange des cases d'indices indval et indmin *)
            SI indval <> indmin
            ALORS inter ← valnum(indval)
                  valnum(indval) ← valnum(indmin)
                  valnum(indmin) ← inter
            FSI
      FPOUR
fin
```

REMARQUE

La connaissance de l'indice de la valeur minimum (indmin) est suffisante. Nous pouvons très facilement retrouver le contenu de la case du tableau à partir de son indice ce qui n'est pas réciproque.

Pour echanger le contenu de deux variables nous devons utiliser une variable intermédiaire. En effet, pour echanger le contenu de A et de B, l'affectation de A dans B ou de B dans A nous fait perdre respectivement la valeur de B ou de A.

■ Algorithme – exercice 16

Tri croissant par recherche des maxima successifs du tableau VALNUM.

MÉTHODE

Le principe sera le même que précédemment, mais en recherchant le maximum, que nous mettrons cette fois-ci à la dernière position du tableau.

```
programme trimax
var valnum(1:30) : tableau d'entiers
    indval, indrech, indmax, inter : entiers
début
   (* saisie du tableau VALNUM *)
   (* application de l'algorithme - exercice 2 *)
   (* boucle de traitement *)
   POUR indval ← 30 JQA 2 PAS -1
     (* recherche du maximum sur un tableau de indval éléments *)
     indmax ← indval
     POUR indrech ← 1 JQA indval-1
          SI valnum(indrech) > valnum(indmax)
          ALORS indmax ← indrech
          FSI
     FPOUR
     SI indval <> indmax
     ALORS inter ← valnum (indval)
          valnum(indval) ← valmun(indmax)
          valnum(indmax) ← inter
     FSI
   FPOUR
fin
```

■ Algorithme – exercice 17

Tri bulle : nous parcourons le tableau en comparant les éléments consécutifs. S'ils sont mal ordonnés, nous les échangeons. Nous recommençons jusqu'à ce qu'il n'y ait plus d'échange.

Exemple :

Soit à trier	5	18	14	4	26
1er parcours	5	14	18	4	26
	5	14	4	18	26
2e parcours	5	4	14	18	26
3e parcours	4	5	14	18	26
4e parcours	4	5	14	18	26

Comme nous n'avons procédé à aucun échange, nous avons terminé.

MÉTHODE

Nous comparons les éléments deux à deux, et nous affectons une variable booléenne à vraie si nous procédons à un échange. La condition d'arrêt du traitement est que la variable booléenne soit restée à faux.

```
programme bulle
var valnum(1:30) : tableau d'entiers
    i, inter : entier
    échange : booléen
début
   (* saisie du tableau nom *)
   (* application de l'algorithme exercice 2 *)
   (* boucle de traitement *)
   REPETER
     (* initialisation du booléen *)
     échange ← .faux.
     (* boucle de recherche des échanges *)
     POUR i ← 1 JQA 29
         SI valnum(i) > valnum(i+1)
         ALORS échange ← .vrai.
               inter ← valnum(i)
               valnum(i) ← valnum(i+1)
               valnum(i+1) ← inter
         FSI
     FPOUR
   JUSQU'A non échange
fin
```

III – Opérations sur un tableau trié

1 - RECHERCHE D'UN ÉLÉMENT

■ Algorithme – exercice 18

Recherche d'un nom saisi au clavier, à l'intérieur d'un tableau de 20 noms ordonnés.

MÉTHODE

Reprenons les principes énoncés dans l'algorithme - exercice 10.

Cette fois, nous arrêtons le traitement lorsque le nom à chercher devient inférieur au nom contenu dans la case traitée du tableau ou lorsque le tableau a été parcouru dans sa totalité.

```
programme recherchetri
var  nom(1:20) : tableau de caractères
     nomàchercher : caractères
     i : entier
début
   (* saisie du tableau NOM *)
   (* cf algorithme exercice 2 *)
   (* saisie du nom à chercher *)
   afficher "entrez un nom "
   saisir nomàchercher
   (* initialisation de l'indice *)
   i ← 1
   (* recherche du mot *)
   TANT QUE nomàchercher > nom(i) et i < 20
     i ← i + 1
   FTQ
   (* affichage du résultat *)
   SI nomàchercher = nom(i)
   ALORS afficher "Le nom figure dans la liste à l'indice ", i
   SINON afficher "Le nom ne figure pas dans cette liste "
   FSI
fin
```

■ Algorithme – exercice 19

Sur un tableau délimité par les indices borneinf et bornesup, ordonné de manière croissante, le principe de recherche dichotomique est le suivant.

• Nous recherchons l'élément situé à l'indice médian du tableau (indice médian représente le milieu du tableau : (borneinf + bornesup)/2).

• Nous effectuons une comparaison entre l'élément recherché et l'élément médian. Trois cas peuvent se présenter :

– Elément à chercher = élément médian : arrêt du traitement.

– Elément à chercher < élément médian : l'élément à chercher est obligatoirement (s'il existe) dans la partie gauche du tableau. Nous réappliquons le même principe en modifiant la borne supérieure de recherche dans le tableau (bornesup ← indice médian – 1).

– Elément à chercher > élément médian : l'élément à chercher est obligatoirement (s'il existe) dans la partie droite du tableau. Nous réappliquons le même principe en modifiant la borne inférieure de recherche dans le tableau (borneinf ← indice médian + 1)

MÉTHODE

Nous utilisons quatre variables :
– Le tableau des valeurs ordonnées de manière croissante
– BORNINF, BORNSUP, INDMED pour délimiter les recherches.
Nous utilisons le principe énoncé ci-dessus. Les conditions d'arrêt du traitement sont :
* Egalité entre la valeur recherchée et l'élément médian.
* La borne supérieure est devenue inférieure à la borne inférieure, car si la valeur n'existe pas, le tableau a été réduit à 0 case.

```
programme dichotomique
var  t(1:100)  : tableau d'entiers
     elecher, bornsup, borninf, indmed : entiers
début
   (* reprendre la saisie d'un tableau T cf algorithme exercice 2 *)
   (* reprendre l'algorithme de tri algorithme exercice 15 à 18 *)
   (* saisie de l'élément à rechercher *)
   afficher "entrez l'élément à rechercher"
   saisir elecher
   borninf ← 1
   bornsup ← 100
   indmed ← (borninf + bornsup) div 2
   TANT QUE t(indmed) <> elecher et borninf < bornsup
     SI  elecher > t(indmed)
     ALORS borninf ← indmed + 1
     SINON bornsup ← indmed - 1
     FSI
     indmed ← (borninf + bornsup) div 2
   FTQ
   SI t(indmed) = elecher
   ALORS afficher "l'élément figure à l'indice ", indmed, "du tableau"
   SINON afficher "l'élément ne figure pas dans le tableau"
   FSI
fin
```

REMARQUE

Une autre méthode consiste à réduire systématiquement le tableau à une case (c-à-d borneinf = bornesup). Deux solutions se présentent :
– Soit l'élément recherché figure dans le tableau, auquel cas il se trouve forcément dans cette dernière case.
– Soit l'élément recherché ne figure pas dans le tableau.

```
programme dichotomieversion2
     (* reprise du début de l'algorithme *)
   REPETER
     indmed ← (borninf + bornsup) div 2
     SI  elecher > t(indmed)
     ALORS borninf ← indmed + 1
     SINON (* attention elecher peut être égal à t(indmed) *)
           bornsup ← indmed
     FSI
```

```
JUSQU'A bornsup = borninf
SI   t(bornsup) = elecher
ALORS afficher "l'élément recherché figure à l'indice ", bornsup
SINON afficher "l'élément recherché ne figure pas dans le tableau"
FSI
```

2 - AJOUT D'UN ÉLÉMENT

Lors de l'ajout d'un élément dans un tableau trié, l'ordre doit y être respecté et le tableau ne doit pas être plein. L'élément doit être inseré à sa place et les éléments plus grands doivent être décalés vers la droite dans la limite supérieure du tableau.

■ Algorithme – exercice 20

Soit une liste de termes informatiques, contenus dans un tableau ordonné suivant l'ordre alphabétique. Ce tableau contient 35 cases au total. Nous voudrions ajouter un nouveau mot dans le répertoire.

MÉTHODE

Nous appelons REPERTOIRE le tableau de caractères.

Nous testons si la 35e case est déjà renseignée, auquel cas le tableau est plein.

Nous recherchons la place à laquelle l'élément doit être inséré et nous décalons vers la droite tous les termes "supérieurs" s'ils existent. Attention à commencer le décalage par l'élément le plus à droite du tableau pour que les contenus soient deplacés avant d'être ecrasés.

```
programme ajouttri
var  répertoire(1:35) : tableau de caractères
     inddec, indins, fin : entiers
     mot : caractères
début
   (*  saisie du tableau REPERTOIRE *)
   (* cf algorithme - exercice 2 *)
   (* saisie du mot à insérer *)
   afficher "entrez le mot à insérer dans le répertoire "
   saisir mot
   SI    répertoire(35) <> " "
   ALORS afficher "nous sommes désolés, le répertoire est complet "
   SINON (* recherche de la place où l'on doit insérer *)
         indins ← 1
         TANT QUE répertoire(indins) < mot et indins < 35
               indins ← indins + 1
         FTQ
         (* recherche de l'indice du dernier élément *)
         fin ← indins
         TANT QUE fin < 35 et répertoire(fin) <> " "
               fin ← fin + 1
         FTQ
         (décalage des mots plus "grand" *)
```

```
            POUR  inddec ← fin - 1 JQA indins PAS -1
                    répertoire(inddec + 1) ← répertoire(inddec)
            FPOUR
            (* mise en place de l'élément à insérer *)
            répertoire(indins) ← mot
        FSI
    fin
```

3 - SUPPRESSION D'UN ÉLÉMENT

Nous procédons d'une manière similaire au traitement d'un tableau non ordonné (algorithme 13 et 14). Seule la recherche de l'élément à supprimer peut être optimisée comme dans l'algorithme 19.

4 - FUSION DE DEUX TABLEAUX ORDONNÉS

L'opération de fusion de deux tableaux ordonnés consiste à interclasser les éléments en provenance des deux tableaux en les ordonnant dans un troisième tableau.

Exemple 4

Soient les deux tableaux de prénoms suivant :

HELENE	MARIE	SOPHIE	SYLVIE
ANDREE	CHRISTINE	JOELLE	RENEE

La fusion de ces deux tableaux ordonnés produit un tableau ordonné de 8 cases.

ANDREE	CHRISTINE	HELENE	JOELLE	MARIE	SOPHIE	SYLVIE	RENEE

❑ Exemple d'application 2

Soit les deux tableaux ordonnés suivants :

TABLE 1	12	14	16	21	34	56	78	90	100
TABLE 2	05	10	12	13	25	67	100		

1 – Donnez le contenu du tableau FUSION, résultat de la fusion de TABLE1 et TABLE2 en conservant les doublons (valeurs communes).

2 – Donnez le contenu du tableau FUSION2, résultat de la fusion de TABLE1 et TABLE2 sans conserver les doublons.

Réponses

1 – FUSION

05	10	12	12	13	14	16	21	25	34	56	67	78	90	100	100

2 – FUSION2

05	10	12	13	14	16	21	25	34	56	67	78	90	100

■ Algorithme – exercice 21

Ecrire l'algorithme permettant d'obtenir le tableau FUSION de l'exemple d'application 2.

MÉTHODE

Nous initialisons à 1 les indices de chacun des trois tableaux.

Nous effectuons un traitement séquentiel de chaque tableau.

Nous comparons deux à deux les contenus de cases provenant de l'un et l'autre des tableaux TABLE1 et TABLE2.

SI le contenu de la case du tableau TABLE1 est supérieur au contenu de la case du tableau TABLE2 :

ALORS nous recopions le contenu de la case du tableau TABLE2 dans la case disponible de FUSION et nous incrémentons de 1 les indices de TABLE2 et FUSION,

SINON nous recopions le contenu de la case du tableau TABLE1 dans la case disponible de FUSION et nous incrémentons de 1 les indices de FUSION et TABLE1.

Nous arrêtons le traitement lorsque l'indice de TABLE1 ou de TABLE2 a atteint sa valeur maximale.

```
programme fusionversion1
var   table1(1:9), table2(1:7), fusion(1:16) : tableau d'entiers
      indtab1, indtab2, indfus : entiers
début
   (* initialisation des indices *)
   indfus ← 1
   indtab1 ← 1
   indtab2 ← 1
   TANT QUE indtab1 <= 9 et indtab2 <= 7
      SI    table1(indtab1) > table2(indtab2)
      ALORS fusion(indfus) ← table2(indtab2)
            indtab2 ← indtab2 + 1
      SINON fusion(inddfus) ← table1(indtab1)
            indtab1 ← indtab1 + 1
      FSI
      indfus ← indfus + 1
   FTQ
```

```
    (* seule une des deux structures POUR qui suivent est exécutée *)
    (* le tableau table1 contient des valeurs supérieures *)
    POUR j ← indtab1 JQA 9
       fusion(indfus) ← table1(j)
       indfus ← indfus + 1
    FPOUR
    (* le tableau table2 contient des valeurs supérieures *)
    POUR j ← indtab2 JQA 7
       fusion(indfus) ← table2(j)
       indfus ← indfus + 1
    FPOUR
    (* affichage du tableau résultat de la fusion *)
    POUR j ← 1 JQA 16
       afficher "la valeur à l'indice ",j,"est",fusion(j)
    FPOUR
fin
```

REMARQUE

La structure POUR ne peut en aucun cas être utilisée à la place de la structure itérative principale. En effet, les deux indices indtab1 et indtab2 ne progressent pas simultanément.

■ Algorithme – exercice 22

Ecrire l'algorithme permettant d'obtenir le tableau FUSION2 de l'exemple d'application 2.

MÉTHODE

Celle-ci ne diffère guère de la précédente. En effet, il suffit d'ajouter la comparaison d'égalité pour éviter de recopier deux fois la même valeur et d'incrémenter les deux indices indtab1 et indtab2 respectivement de 1.

```
programme fusionversion2
   (* reprendre l'algorithme précédent jusqu'à la structure itérative *)
   TANT QUE indtab1 < = 9 et indtab2 < = 7
     SI table1(indtab1) > table2(indtab2)
     ALORS fusion(indfus) ← table2(indtab2)
           indtab2 ← indtab2 + 1
     SINON fusion(indfus) ← table1(indtab1)
           indtab1 ← indtab1 + 1
           SI table1(indtab1) = table2(indtab2)
           ALORS indtab2 ← indtab2 + 1
           FSI
     FSI
     indfus ← indfus + 1
   FTQ
(* reprise de la suite *)
```

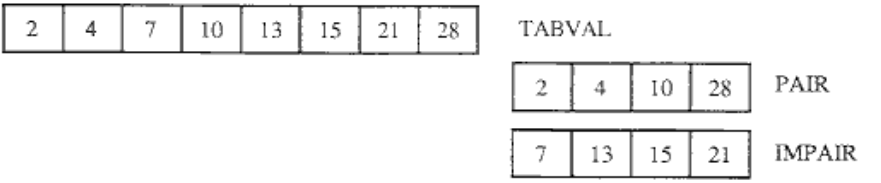
5 - ÉCLATEMENT D'UN TABLEAU ORDONNÉ

■ Algorithme – exercice 23

Soit un tableau de 100 valeurs ordonnées de manière croissante, écrire l'algorithme permettant de créer deux nouveaux tableaux ordonnés de manière croissante, contenant l'un les valeurs paires, l'autre les valeurs impaires.

MÉTHODE

Nous différençons une valeur paire d'une valeur impaire, car la première est divisible par 2. Aussi, nous analysons chacune des valeurs contenues dans le tableau initial, et nous les stockons dans le tableau correspondant.




```
programme pairimpair
var  tabval(1:100) , pair(1:100) , impair(1:100) : tableau d'entiers
     indpair, indimpair, indval : entiers
début
   (* saisie des valeurs stockées dans TABVAL *)
   (* application de l'algorithme exercice 2 *)
   (* initialisation des indices des tableaux pair et impair *)
   indpair ← 1
   indimpair ← 1
   (* éclatement du tableau tabval *)
   POUR indval ← 1 JQA 100
     SI  (tabval(indval) MOD 2) = 0
     ALORS     (* le nombre est pair *)
               pair(indpair) ← tabval(indval)
               indpair ← indpair + 1
     SINON     (* le nombre est impair *)
               impair(indimpair) ← tabval(indval)
               indimpair ← indimpair + 1
     FSI
   FPOUR
   (* édition des résultats *)
   afficher "liste des nombres pairs : "
   POUR indval ← 1 JQA indpair - 1
     afficher pair(indval)
   FPOUR
   afficher "liste des nombres impairs : "
   POUR indval ← 1 JQA indimpair - 1
     afficher impair(indval)
   FPOUR
fin
```

Réponses

1 – FUSION

05	10	12	12	13	14	16	21	25	34	56	67	78	90	100	100

2 – FUSION2

05	10	12	13	14	16	21	25	34	56	67	78	90	100

■ Algorithme – exercice 21

Ecrire l'algorithme permettant d'obtenir le tableau FUSION de l'exemple d'application 2.

MÉTHODE

Nous initialisons à 1 les indices de chacun des trois tableaux.

Nous effectuons un traitement séquentiel de chaque tableau.

Nous comparons deux à deux les contenus de cases provenant de l'un et l'autre des tableaux TABLE1 et TABLE2.

SI le contenu de la case du tableau TABLE1 est supérieur au contenu de la case du tableau TABLE2 :

ALORS nous recopions le contenu de la case du tableau TABLE2 dans la case disponible de FUSION et nous incrémentons de 1 les indices de TABLE2 et FUSION,

SINON nous recopions le contenu de la case du tableau TABLE1 dans la case disponible de FUSION et nous incrémentons de 1 les indices de FUSION et TABLE1.

Nous arrêtons le traitement lorsque l'indice de TABLE1 ou de TABLE2 a atteint sa valeur maximale.

```
programme fusionversion1
var   table1(1:9), table2(1:7), fusion(1:16) : tableau d'entiers
      indtab1, indtab2, indfus : entiers
début
   (* initialisation des indices *)
   indfus ← 1
   indtab1 ← 1
   indtab2 ← 1
   TANT QUE indtab1<= 9 et indtab2 <= 7
      SI    table1(indtab1) > table2(indtab2)
      ALORS fusion(indfus) ← table2(indtab2)
            indtab2 ← indtab2 + 1
      SINON fusion(inddfus) ← table1(indtab1)
            indtab1 ← indtab1 + 1
      FSI
      indfus ← indfus + 1
   FTQ
```

Nous comparons le contenu d'une case avec la valeur recherchée. S'il y a égalité, nous incrémentons le compteur de 1.

```
programme dim2recherche
var  valeurs(1:3,1:15) : tableau d'entiers
     nombre, col, lig, compteur : entiers
début
   (* saisie du tableau *)
   POUR lig ← 1 JQA 3
     afficher "Nombres de la ligne " ,lig
     POUR col ← 1 JQA 15
         afficher "Nombre numéro " ,col
         saisir valeur (lig,col)
     FPOUR
   FPOUR
   (* saisie du nombre à rechercher *)
   afficher "entrez un nombre "
   saisir nombre
   (* initialisation du compteur *)
   compteur ← 0
   (* recherche des occurrences *)
   POUR lig ← 1 JQA 3
     POUR col ← 1 JQA 15
          SI   valeur(lig,col) = nombre
          ALORS  compteur ← compteur + 1
          FSI
     FPOUR
   FPOUR
   (* affichage du résultat *)
   afficher "il y ", compteur, "occurrence(s) de ",nombre
fin
```

■ Algorithme – exercice 25

Considérons le tableau à deux dimensions, contenant pour chaque élève de la classe, ses moyennes dans chacune des 10 matières qui lui sont enseignées. Chaque élève reçoit obligatoirement un enseignement pour chacune des 10 matières. Les noms des élèves et ceux de chacune des 10 matières sont stockés dans des tableaux à une dimension.

Illustration :

	MAT	Analyse	Logiciel	Composant	Langages	...	
NOM							
ALBAN	MOYENNE	10	02	15	14	...	
DUPONT		08	05	02	17		
DURAND		12	11	17	15		
FLEUR		14	13	17	16		
...		...					

Nous vous demandons d'écrire l'algorithme permettant l'édition du récapitulatif des moyennes par matière, de la moyenne générale de chaque élève et de la classe.

MÉTHODE

Nous utilisons deux indices : indele, correspondant à l'indice ligne de l'élève, et indmat correspondant à l'indice colonne des moyennes par matière.

Pour effectuer la moyenne générale de l'élève, nous additionnons les moyennes une à une et nous divisons cette somme par 10, nous cumulons ces moyennes générales pour obtenir la moyenne de la classe.

```
programme relevé
var mat(1:10) ,nom(1 : 32): tableau de caractères
    moyenne(1:32,1:10): tableau de réels
    indele, indmat : entiers
    moygen, moycla : réels
début
   (* saisie des différents tableaux *)
   POUR indmat ← 1 JQA 10
    afficher "enter le mon de la matière numéro ",indmat
    saisir mat(indmat)
   FPOUR
   POUR indele ← 1 JQA 32
    afficher "entrer le nom de l'élève numéro ",indele
    saisir nom(indele)
    POUR indmat ← 1 JQA 10
         afficher "entrer sa moyenne obtenue en ",mat(indmat)
         saisir moyenne (indele,indmat)
    FPOUR
   FPOUR
   moycla ← 0
   (* répétitive principale *)
   POUR indele ← 1 JQA 32
    (* traitement d'un élève *)
    moygen ← 0
    afficher "nom de l'élève : ", nom(indele)
    (* traitement de chaque matière *)
    POUR indmat ← 1 JQA 10
         afficher "moyenne obtenue en ",mat(indmat) ,":"
         afficher moyenne(indele,indmat)
         moygen ← moygen + moyenne(indele,indmat)
    FPOUR
    afficher "moyenne générale de l'élève ", moygen/10
    moycla ← moycla + moygen/10
   FPOUR
   afficher "moyenne générale de la classe ", moycla/32
fin
```

■ Algorithme – exercice 26

Saisir des nombres entiers dans un tableau de 10 lignes et de 20 colonnes. Calculer les totaux par ligne et par colonne dans des tableaux TOTLIG et TOTCOL.

MÉTHODE

Le tableau de travail possède :

– 10 lignes donc TOTLIG contient 10 valeurs.

– 20 colonnes, donc TOTCOL contient 20 valeurs.

Trois étapes sont à respecter :

– Saisie du tableau de travail, à l'aide de deux boucles POUR.

– Calcul des totaux lignes : pour chaque valeur de l'indice ligne, il faut ajouter les éléments situés aux différentes colonnes de la ligne (faire varier l'indice colonne).

– Calcul des totaux colonnes : pour chaque valeur de l'indice colonne, il faut ajouter les éléments situés aux différentes lignes de la colonne (faire varier l'indice ligne).

Les deux étapes de cumuls peuvent être faites dans la même boucle.

```
programme totaux
var travail(1:10,1:20), totlig(1:10), totcol(1:20) : tableau
                                                     de réels
   indlig, indcol : entiers
début
   (* saisie du tableau travail *)
   POUR indlig ← 1 JQA 10
     afficher "ligne numéro" ,indlig
     POUR indcol ← 1 JQA 20
         afficher "entrez le " ,indcol," ième nombre"
         saisir travail(indlig,indcol)
     FPOUR
   FPOUR
   (* initialisation du tableau totlig *)
   POUR indlig ← 1 JQA 10
     totlig(indlig) ← 0
   FPOUR
   (* initialisation du tableau totcol *)
   POUR indcol ← 1 JQA 20
     totcol(indcol) ← 0
   FPOUR
   (* calcul des totaux colonnes et lignes *)
   POUR   indlig ← 1 JQA 10
     POUR indcol ← 1 JQA 20
          totlig(indlig) ← totlig(indlig) + travail(indlig,indcol)
          totcol(indcol) ← totcol(indcol) + travail(indlig,indcol)
     FPOUR
   FPOUR
fin
```

PROBLEME 3

Nous notons les 6 numéros du loto national pendant 52 semaines. Nous supposons qu'un seul tirage est effectué par semaine, et que seuls 6 numéros sont tirés au sort (pas de numéro complémentaire).

1 – Ecrire l'algorithme permettant de calculer la fréquence de sortie de chaque numéro (valeur de 1 à 49).

2 – Ecrire l'algorithme permettant de donner les 6 numéros les plus fréquents.

SOLUTION

1) MÉTHODE

Nous utilisons un tableau à 2 dimensions pour stocker les 6 numéros hebdomadaires, LOTO (1:52;1:6).

Comme il existe 49 valeurs différentes des numéros, nous stockons leur fréquence dans un tableau à une dimension FREQ(1:49).

Pour calculer ces fréquences, nous parcourons le tableau LOTO dans sa totalité. Pour chaque numéro lu, nous devons incrémenter de 1 le compteur correspondant. C'est-à-dire la valeur située à l'indice du tableau FREQ correspondant au numéro traité dans LOTO. Donc, la valeur contenue dans une case du tableau LOTO, correspond à l'indice de la case du tableau FREQ.

```
programme lotofreq
var loto(1:52,1:6),freq(1:49) : tableau d'entiers
    indlig, indcol : entiers
début
   (* saisie des différents tirages *)
   POUR indlig ← 1 JQA 52
     POUR      indcol ← 1 JQA 6
         REPETER
               afficher "entrez le ",indcol,"ième nombre de
                        la",indlig,"ième semaine"
               saisir loto(indlig,indcol)
         JUSQU'A loto(indlig,indcol) > 0 et loto(indlig,indcol) < 50
     FPOUR
   FPOUR
   (* initialisation du tableau des fréquences *)
   POUR indcol ← 1 JQA 49
     freq(indcol) ← 0
   FPOUR
   (* calcul des fréquences *)
   POUR indlig ← 1 JQA 52
     POUR indcol ← 1 JQA 6
          freq(loto(indlig,indcol)) ← freq(loto(indlig,indcol))+1
     FPOUR
   FPOUR
fin
```

2) MÉTHODE

Pour réaliser cette question, nous reprenons l'algorithme précédent et nous le complétons.

Il s'agit de réaliser une recherche successive des maximums sur la table des fréquences FREQ, en prenant soin de ne plus traiter par la suite le maximum déjà affiché. En effet, supposons que le loto ne comporte que 21 numéros, et que la table des fréquences soit la suivante :

FREQ	12	13	56	78	89	54	67	88	95	87	67	56	45	34	43	65	76	87	67	56	28

Premier passage : max = 95 pour le numéro 9.

Si l'on ne modifie pas la table, chaque passage indiquera que 95 est le maximum.

Première hypothèse :

On effectue un tri de la table selon la méthode de la recherche du maximum, auquel cas la valeur 95 est repoussée en bout de table :

12	13	56	78	89	54	67	88	28	87	67	56	45	34	43	65	76	87	67	56	95

mais on modifie totalement le sens de la table. Le numéro le plus sorti devient le numéro 21 !!!

Deuxième hypothèse :

Dans la limite où l'on ne désire pas conserver les valeurs exactes des fréquences pour un traitement ultérieur, on remplacera la fréquence du maximum par une valeur impossible et petite : –1, par exemple.

12	13	56	78	89	54	67	88	–1	87	67	56	45	34	43	65	76	87	67	56	28

et au passage suivant on obtient : maximum 89 pour le numéro 5.

Nous recommençons ce travail 6 fois.

```
programme lotoproba
(* reprendre les déclarations précédentes en ajoutant *)
var  indmax, nbfois : entiers
début
   (* reprendre la totalité de l'algorithme précédent *)
   POUR nbfois ← 1 JQA 6
     indmax ← 1
     POUR indlig ← 2 JQA 49
         SI    freq(indlig) > freq(indmax)
         ALORS indmax ← indlig
         FSI
     FPOUR
```

```
        afficher "classement ",nbfois, "pour",indmax,"sorti";
        afficher freq(indmax), "fois"
        freq(indmax) ← -1
      FPOUR
    fin
```

■ PROBLEME 4

Un professeur veut informatiser sa gestion de notes.

Il enseigne à une seule classe les 4 matières suivantes : la technologie, l'analyse, la programmation et les travaux pratiques sur machine.

Il possède une seule note dans chaque matière par trimestre.

Il veut utiliser quatre structures de tableau :

- MATIERE, à une dimension, pour gérer les noms des matières.
- NOM, à une dimension, pour gérer les noms de ses élèves.
- PRENOM, à une dimension, pour gérer les prénoms de ses élèves.
- NOTE, à deux dimensions, pour gérer les notes de chaque élève pour chaque matière.

Nous trouvons à la première case de NOM et PRENOM et à la première ligne de NOTE les renseignements du premier élève dans l'ordre alphabétique.

Tous les trimestres, il veut éditer l'état suivant :

MATIERE :	TECHNO	ANAL.	PROG.	T.P.	
MOY. CLASSE :	...	...	...	...	
NOTE MINI :	...	...	...	...	
NOTE MAXI :	...	...	...	...	
NOM / PRENOM	notes				moyenne
...	...	...	...	...	
...	...	...	...	...	...

NB : l'édition des noms d'élèves, des notes et des moyennes doit se faire dans l'ordre décroissant des moyennes.

Le professeur désire conserver les moyennes de chaque trimestre, afin de les comparer ultérieurement aux moyennes des autres trimestres.

1) Définir très clairement les variables simples et les tableaux dont vous avez besoin.

2) Ecrire l'algorithme de saisie des différentes notes et d'édition de l'état récapitulatif.

SOLUTIONS

1) – Les trois tableaux supplémentaires utilisés sont :

- Le tableau des moyennes par matière, MOYMAT, à une dimension (4 cases).
- Le tableau des minima et maxima par matières, à deux dimensions, MINMAX (4 lignes, 2 colonnes).
- Le tableau des moyennes élèves, MOYELE. Quelle dimension lui donner ?

Première hypothèse :

Ce tableau est à une dimension (une case par élève). Or, lors de l'édition qui doit se faire dans l'ordre décroissant de moyennes, il faudra pratiquer de la manière précédente (remplacer les valeurs successivement par –1), auquel cas nous ne pouvons pas réutiliser les moyennes et ceci est contraire aux exigences de l'énoncé.

Deuxième hypothèse :

Ce tableau est à une dimension, mais au moment où l'on effecture le tri, à chaque valeur déplacée, on décale dans les trois tableaux NOM, PRENOM, NOTE, les valeurs associées à cet élève. (Très coûteux en temps d'exécution.)

Troisième hypothèse :

Ce tableau est à deux dimensions : autant de lignes que d'élèves. Pour chaque ligne, deux renseignements, la moyenne et le numéro (dans l'ordre alphabétique) de l'élève. Ainsi, au moment du tri, nous n'aurons que deux valeurs à déplacer.

– Les variables simples utilisées sont les suivantes :

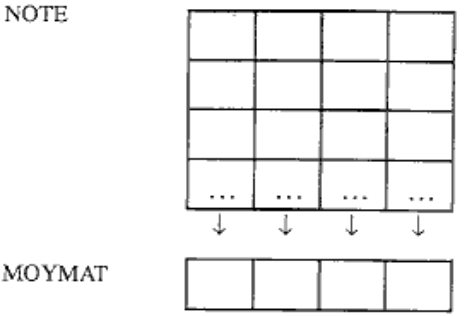
- INDELE, de type entier, variant de 1 à 24, servant d'indice aux tableaux NOM et PRENOM, et d'indice ligne aux tableaux NOTE et MOYELE.
- INDMAT, de type entier, variant de 1 à 4, servant d'indice au tableau MATIERE, et d'indice colonne aux tableaux NOTE.
- INDMAX et INDMIN, de type entier, qui servent d'indice repère des moyennes maximale et minimale.
- INTER, de type réel pour assurer les échanges, et INDTRAV de type entier, indice de travail sur les tableaux.

2) – MÉTHODE

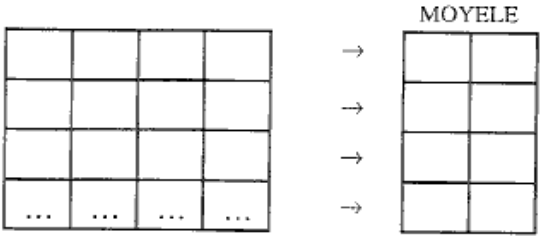
Quatre étapes fondamentales à ce traitement :

– Saisie des trois tableaux NOM, PRENOM, NOTE, avec contrôle de vraisemblance de la note.

– Calcul de la moyenne par matière : parcours en colonne du tableau NOTE et stockage de la moyenne des colonnes dans le tableau MOYMAT.



• Calcul des moyennes par élèves : parcours en ligne du tableau NOTE et mise en place de la moyenne dans la ligne correspondant à l'élève dans le tableau MOYELE.



Ces deux calculs peuvent être effectués dans la même boucle.
- Recherche du minimum et du maximum par matière, dans le tableau NOTE.
- Tri par recherche du maximum du tableau MOYELE.
- Edition du relevé.

```
programme bulletin
var  note(1:24;1:4),moyele(1:24,1:2),moymat(1:4) : tableaux de réels
   minmax(1:2,1:4) : tableau de réels
   nom(1:24),prénom(1:24),matière(1:4) : tableaux de caractères
   indtrav, inter, indele, indmat, indmin, indmax : entiers
début
   (* affectation des matières *)
   matière(1) ← "techno"
   matière(2) ← "analyse"
   matière(3) ← "program."
   matière(4) ← "T.P."
   (* saisie des noms, prénoms et notes *)
   POUR indele ← 1 JQA 24
     afficher "entrer les nom et prénom de l'élève n° ",indele
     saisir nom(indele)
     saisir prénom(indele)
     POUR indmat ← 1 JQA 4
         REPETER
               afficher "entrez la note obtenue en ", matière(indmat)
               saisir note(indele,indmat)
         JUSQU'A note(indele,indmat) > = 0 et note(indele,indmat) <= 20
     FPOUR
```

```
FPOUR
(* initialisation des moyennes matières *)
POUR  indmat ← 1 JQA 4
  moymat(indmat) ← 0
FPOUR
(* calcul des moyennes élèves *)
POUR  indele ← 1 JQA 24
  (* initialisation du repère de l'élève *)
  moyele(indele,2) ← indele
  (* initialisation de la moyenne *)
  moyele(indele,1) ← 0
  POUR indmat ← 1 JQA 4
       moymat(indmat) ← moymat(indmat) + note(indele,indmat)/24
      moyele(indele,1) ← moyele(indele,1) + note(indele,indmat)/4
  FPOUR
FPOUR
(* recherche des minima et maxima par matière *)
POUR  indmat ← 1 JQA 4
  indmin ← 1
  indmax ← 1
  POUR indele ← 2 JQA 24
      SI note(indele,indmat) < note(indmin,indmat)
      ALORS (* nouveau minimum *)
            indmin ← indele
      SINON SI note(indele,indmat) > note(indmax,indmat)
            ALORS (* nouveau maximum *)
                  indmax ← indele
            FSI
      FSI
  FPOUR
  (* remplissage du tableau minmax *)
  minmax(1,indmat) ← note(indmin, indmat)
  minmax(2,indmat) ← note(indmax, indmat)
FPOUR
(* tri de moyele par recherche du maximum *)
POUR  indele ← 1 JQA 23
  (* recherche du maximum *)
  indmax ← indele
  POUR indtrav ← indele + 1 JQA 24
       SI moyele(indmax,1) < moyele(indtrav,1)
       ALORS indmax ← indtrav
       FSI
  FPOUR
  (* échange de position du maximum *)
  SI  indmax <> indele
  ALORS (* échange des moyennes *)
        inter ← moyele(indele,1)
        moyele(indele,1) ← moyele(indmax,1)
        moyele(indmax,1) ← inter
        (* échange des repères élèves *)
```

```
            inter ← moyelé(indele,2)
            moyele(indele,2) ← moyele(indmax,2)
            moyele(indmax,2) ← inter
        FSI
      FPOUR
      (* édition de l'état *)
      (* affichage des matières *)
      afficher "matière : ";
      POUR indmat ← 1 JQA 4
        afficher matière(indmat) ;
      FPOUR
      (* affichage des moyennes de la classe *)
      afficher "moyenne classe :"
      POUR indmat ← 1 JQA 4
        afficher moymat(indmat);
      FPOUR
      (* affichage minimum et maximum *)
      afficher "note mini : "
      POUR indmat ← 1 JQA 4
        afficher minmax(1,indmat);
      FPOUR
      afficher "note maxi : "
      POUR indmat ← 1 JQA 4
        afficher minmax(2,indmat);
      FPOUR
      (* affichage du corps par ordre décroissant des moyennes *)
      afficher "nom/prénom notes              moyenne"
      POUR indele ← 1 JQA 24
        (* indice de l'élève dans l'ordre décroissant des moyennes *)
        indtrav ← moyele(indele,2)
        afficher nom(indtrav),prénom(indtrav);
        POUR indmat ← 1 JQA 4
            afficher note(indtrav,indmat);
        FPOUR
        afficher moyele(indele,1)
      FPOUR
fin
```

3 Les sous-programmes

I – Les actions nommées

Nous avons annoncé, dans le premier chapitre, qu'un algorithme ne devait pas dépasser la longueur d'une page. Or, force est de constater que le dernier algorithme écrit ne respecte pas cette règle.

Nous reprenons la définition d'un algorithme : une suite d'actions ordonnées en séquence qui portent sur les objets d'un univers fini. Chacune de ces actions peut être elle-même un algorithme. Chacun de ces algorithmes est appelé une ACTION NOMMEE. Lorsque nous l'utilisons à l'intérieur d'un autre algorithme, nous soulignons son nom, afin de bien mettre en valeur le fait que nous desirons exécuter à cet instant un autre algorithme. Sa description doit être faite en-dehors des limites de l'algorithme qui l'utilise, en respectant les règles syntaxiques habituelles. Toutefois, lors de la description de l'action, nous remplaçons le mot PROGRAMME par le mot ACTION.

Nous parlons alors d'algorithme APPELANT et d'algorithme APPELE. L'appelant est celui qui contient l'appel à l'action nommée, qui est elle-même l'appelée.

L'algorithme appelé utilise les variables déclarées dans l'appelant. Nous parlons alors de variables GLOBALES. L'appelé peut également déclarer ses propres variables qu'il est le seul à utiliser. Nous parlons alors de variables LOCALES.

■ Algorithme – exercice 1

Ecrire l'algorithme permettant, à partir de la saisie de trois nombres, de rechercher le minimum ou le maximum, au choix de l'utilisateur.

Les étapes à respecter sont les suivantes :

- Saisie des trois nombres
- Affichage du menu
- Saisie du choix
- En fonction du choix :
  - Calcul du minimum
  - Calcul du maximum

1 – Que peut faire l'algorithme appelant ?

2 – Que peut (peuvent) faire l'(les) algorithme(s) appelé(s) ?

3 – Quelles sont alors les variables globales ?

4 – Quelles sont les variables locales ?

5 – Ecrire les algorithmes respectifs.

MÉTHODE

Nous reprenons les principes vus au premier chapitre dans l'algorithme – exercice 11 en utilisant cette fois des actions nommées.

1 – L'algorithme appelant va permettre la saisie des trois nombres, l'affichage du menu et la saisie du choix du travail à effectuer.

2 – L'un des algorithmes appeles va permettre la recherche du minimum, l'autre la recherche du maximum.

3 – Les trois variables NB1, NB2, NB3 qui reçoivent les nombres et la variable CHOIX, qui reçoit le travail à effectuer sont globales.

4 – La variable MIN pour la recherche du minimum, la variable MAX pour la recherche du maximum sont locales.

```
programme recherche
var  (* variables globales *)
     nb1, nb2, nb3 : réels
     choix : entier
début
   (* saisie des trois nombres *)
   afficher "entrez trois nombres"
   saisir nb1
   saisir nb2
   saisir nb3
   (* affichage du menu pour choix *)
   afficher "voulez-vous : "
   afficher " 1 - Rechercher le minimum "
   afficher " 2 - Rechercher le maximum "
   saisir choix
   (* réalisation du choix *)
   SUIVANT choix FAIRE
     1 : minimum
     2 : maximum
     SINON afficher "erreur dans votre choix "
   FINSUIVANT
fin


action minimum
var  (* variable locale *)
    min : réel
début
   (* recherche du minimum relatif *)
   min ← nb1
   SI    min > nb2
   ALORS min ← nb2
```

```
   FSI
   SI min > nb3
   ALORS min ← nb3
   FSI
   (* affichage du minimum absolu *)
   afficher "le minimum de ",nb1, nb2, nb3, "est ", min
fin


action maximum
var  (* variable locale *)
     max : réel
début
   (* recherche du maximum relatif *)
   max ← nb1
   SI max < nb2
   ALORS max ← nb2
   FSI
   SI max < nb3
   ALORS max ← nb3
   FSI
   (* affichage du maximum absolu *)
   afficher "le maximum de ",nb1, nb2, nb3, "est ", max
fin
```

REMARQUE

L'utilisation d'actions nommées permet une plus grande lisibilité de l'algorithme principal (l'appelant). Nous pouvons faire très aisément des mises à jour de ce programme.

II – Les actions paramétrées

1 - ÉNONCÉ DU PROBLEME

■ Algorithme – exercice 2

Nous voulons écrire l'algorithme permettant de calculer la différence, exprimée en nombre de jours, entre deux dates de la même année, saisies au clavier. Pour cela, nous utilisons le principe du "quantième" d'une date.

Nous rappelons que le quantième d'une date est l'attribution d'un numéro séquentiel à chaque jour de l'année, en partant de 1 au premier janvier de l'année :

1er JANVIER –> quantième 1
1er FEVRIER –> quantième 32
...
31 DECEMBRE –> quantième 365
366 (si l'année est bissextile)

MÉTHODE

En ramenant chacune des dates saisies au clavier à leurs quantièmes respectifs, nous obtenons par différence le nombre de jours séparant ces deux dates.

– Comment allons-nous calculer le quantième d'une date ?

Sachant si l'année est bissextile (divisible par 4, si on se limite au XX^{e} siècle), et en connaissant le quantième du dernier jour de chaque mois, nous pouvons calculer par une simple addition le quantième de la date considérée.

– Comment allons-nous procéder ?

Sachant que ce calcul est effectué deux fois (une fois pour chaque date), nous écrivons cet algorithme comme une action nommée. Nous initialisons correctement les variables globales utilisées avant de faire un appel à cette action :
– Initialisation de la variable jour
– Initialisation de la variable mois
– Initialisation de la variable an
– Appel à "quantième"

L'action "quantième" est considérée comme une boîte cohérente et indépendante par l'algorithme principal qui, à partir de données valorisées dans l'algorithme principal, rend les résultats escomptés.

– Quelles sont les variables globales ?
Jour1, mois1, an1 et quant1, qui reçoivent la première date et son quantième.
Jour2, mois2, an2 et quant2, qui reçoivent la deuxième date et son quantième.
Jour, mois, an et quant, qui sont utilisés dans l'action nommée pour le calcul d'un quantième, et sont affectés dans l'algorithme appelant.

– Quelles sont les variables locales ?
Une seule variable est à déclarer dans l'action : c'est le tableau qui contient les quantièmes des derniers jours de chaque mois d'une année non bissextile. En effet :

Le quantième du

1er janvier	c'est	1 c-à-d	0 + 1	(tmois(1) + jour)
1er février		32 c-à-d	31 + 1	(tmois(2) + jour)
29 février		60 c-à-d	31 + 29	(tmois(2) + jour)
3 mars d'une année non bissextile		62 c-à-d	59 + 3	(tmois(3) + jour)
3 mars d'une année bissextile		63 c-à-d	59 + 1 + 3	
			(tmois(3) + 1 + jour)	
			(+ 1 pour compenser le jour supplémentaire)	

```
programme appelant-versionactions
var jour, mois, an, quant, quant1, quant2 : entiers
    jour1, mois1, an1, jour2, mois2, an2 : entiers
début
   (* saisie de la première date *)
   afficher "donner successivement le jour, le mois, l'année de
             la première date"
   saisir jour1, mois1, an1
   (* saisie de la deuxième date *)
   afficher "donner successivement le jour, le mois, l'année de
```

```
    la deuxième date"
   saisir jour2, mois2, an2
   (* affectation des variables de travail de l'action *)
   jour ← jour1
   mois ← mois1
   an ← an1
   (* appel à l'action *)
   quantième
   (* affectation du résultat de l'action *)
   quant1 ← quant
   (* affectation des variables de travail de l'action *)
   jour ← jour2
   mois ← mois2
   an ← an2
   (* appel de l'action *)
   quantième
   (* affectation du résultat de l'action *)
   quant2 ← quant
   (* édition de la différence *)
   afficher "la différence en nombre de jours entre "
   afficher jour1, "/", mois1, "/", an1, "et "
   afficher jour2, "/", mois2, "/", an2, "est "
   afficher ABSOLU(quant1 - quant2)
fin


action quantième
const tmois(1:12) : tableau d'entiers
        c'est 0,31,59,90,120,151,181,212,243,273,304,334
début
   (* calcul du quantième *)
   quant ← tmois(mois) + jour
   (* réévaluation du quantième pour une année bissextile si *)
   (* le mois est au-delà du mois de février *)
   SI an mod 4 = 0 et mois > 2
     ALORS quant ← quant + 1
   FSI
fin
```

REMARQUES :

Si, dans la suite du programme principal, nous devons faire le même travail sur une date déclarée avec d'autres identificateurs, une affectation des nouvelles valeurs aux variables globales utilisées par l'action est nécessaire.

Si, dans un autre algorithme, nous avons besoin également de calculer le quantième d'une date, nous devons, soit :

– redécrire une action nommée identique dans sa logique, mais pas dans son contenu.

– réutiliser l'action nommée écrite en n'oubliant pas de déclarer les variables globales en conservant leurs identificateurs exacts.

INCONVÉNIENTS

Cette dernière solution suppose que nous nous souvenions du nom des identificateurs utilisés dans l'action, afin de pouvoir les déclarer et les affecter correctement.

2 - LES PROCÉDURES

L'idéal est de permettre à une action d'être appelée de façon universelle par tout algorithme. C'est-à-dire, pouvoir aussi bien écrire, dans l'algorithme précédent :

quantième(10,12,88)

ou quantième(JJ,MM,AA) JJ, MM et AA étant des variables contenant une date

ou quantième(A, B, C)

Pour cela, nous associons à cette action :

- Un nom comme tout algorithme.
- Une liste formelle :
  - des informations que doit lui fournir l'algorithme appelant pour qu'elle puisse fonctionner;
  - des informations qu'elle restitue.

Cette LISTE FORMELLE se traduit par une suite de variables. Dans l'exemple précédent (l'action quantième), JOUR, MOIS, AN et QUANT sont des identificateurs formels.

Pour utiliser cette action, nous devons lui fournir une liste de VALEURS REELLES permettant son exécution (rôles joués dans les exemples par 10, 12 et 88, ou JJ, MM et AA, ou A, B et C). Cette liste peut aussi bien être une suite de nombre, ou une suite d'identificateurs, ou encore les deux.

DÉFINITION

Une action à laquelle sont associés un nom et une liste de paramètres formels est dite action paramétrée ou procédure. Pour utiliser une telle action, il faut lui fournir une liste de paramètres réels.

Un paramètre réel est un identificateur ou une valeur pour lequel nous effectuons l'action.

Un paramètre formel est un identificateur présent dans la liste formelle des informations. C'est lui qui reçoit la valeur du paramètre réel fourni à l'appel. Nous attribuons à chacun des paramètres formels des noms différents et nous précisons une information supplémentaire qui est son statut. En effet, il existe trois statuts possibles :

- DONNEE(S) : début procédure :
  la valeur du paramètre réel est connue à l'appel de l'action
  fin procédure :
  le paramètre réel qui a fourni la valeur n'est en aucun cas modifié
- RESULTAT(S) : début procédure :
  la valeur du paramètre réel est ignorée à l'appel de l'action
  fin procédure :

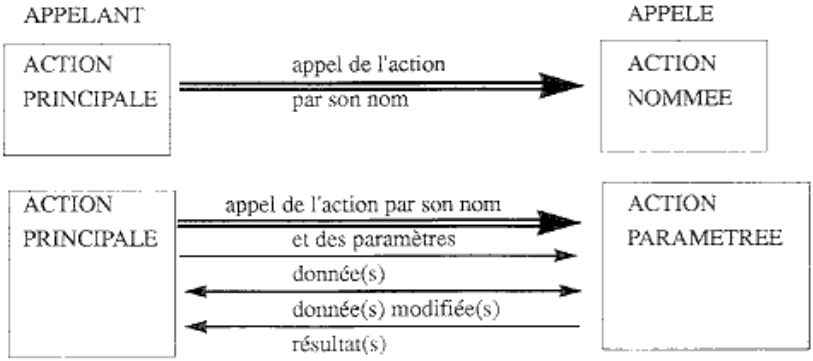
le paramètre réel est affecté par la valeur du paramètre formel à l'issue de l'action

– DONNEE(S) MODIFIEE(S) : début procédure

la valeur du paramètre réel est connue à l'appel de l'action

fin procédure :

le paramètre réel est affecté par la valeur du paramètre formel associé, à l'issue de l'action



❑ Exemple d'application 1

Reprendre l'exercice sur l'action quantième.

1 – Indiquer quels sont les paramètres formels et leurs statuts.
2 – Quelle est la caractéristique du tableau TMOIS ?

RÉPONSES

1 – Il existe quatre paramètres formels :

JOUR, MOIS et AN, qui doivent être fournis à l'action, afin de calculer un quantième. Ils sont donc du type DONNEES.

QUANT, qui est restitué par l'action, est du type RESULTAT.

2 – Le tableau TMOIS est une variable de travail, dite variable locale, à l'action quantième.

❑ Exemple d'application 2

Soit une action paramétrée permettant la mise à jour du solde d'un compte. Quels sont les paramètres formels qu'il faut fournir à l'action pour permettre la mise à jour de n'importe quel compte (précisez pour chacun leur statut).

RÉPONSES

Sachant que la mise à jour d'un solde se fait à partir de la nature du solde, de son montant, de la nature de l'opération et de son montant, il faut déclarer quatre paramètres :
– NATOPER (nature de l'opération), MONTOPER (montant de l'opération) en DONNEES.
– NATSOLDE (nature du solde du compte à mettre à jour), SOLDCOMPTE (solde du compte à mettre à jour) en DONNEES MODIFIEES.

DÉCLARATION D'UNE ACTION PARAMÉTRÉE

```
PROCEDURE <nom de l'action>
   (DONNEE(S) : <liste des identificateurs formels ayant ce statut>
   DONNEE(S) MODIFIEE(S) : <liste des identificateurs formels ayant ce statut>
   RESULTAT(S) : <liste des identificateurs formels ayant ce statut>)
```

Derrière les mots : donnée(s), donnée(s) modifiée(s) ou résultat(s), nous déclarons les paramètres formels. Nous ne leur assoçions pas de type, puisque celui-ci est, par défaut, le type des paramètres réels.

Le corps de l'action a la même structure qu'un algorithme principal :

– Phase de déclaration de ses propres variables (variables locales).

– Phase de traitement.

APPEL D'UNE ACTION PARAMÉTRÉE

L'appel se situe au niveau de l'algorithme appelant et se fait de la manière suivante :

```
NOM-DE-L'ACTION (<liste des paramètres réels>)
```

Attention : il n'est pas nécessaire de connaître le nom des paramètres formels pour utiliser l'action paramétrée, PAR CONTRE, il faut connaître l'ORDRE dans lequel ils ont été déclarés afin de le respecter IMPERATIVEMENT.

❑ Exemple d'application 3

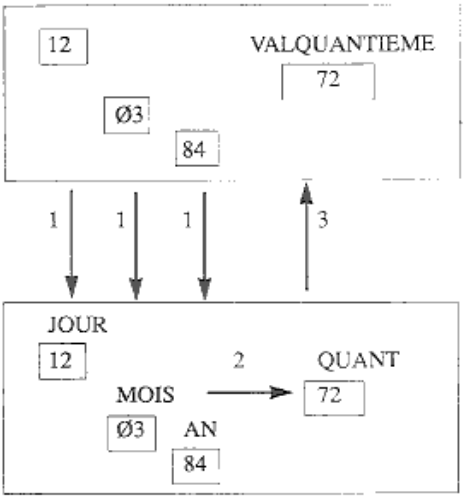
Traduire l'action nommée QUANTIEME sous forme d'action paramétrée.

RÉPONSE :

```
programme appelant-versionactions-paramètrées
var jour1, mois1, an1, jour2, mois2, an2, quant1, quant2 : entiers
début
   (* saisie de la première date *)
   afficher "donner successivement le jour, le mois, l'année de
             la première date"
   saisir jour1, mois1, an1
   (* saisie de la deuxième date *)
   afficher "donner successivement le jour, le mois, l'année de
             la deuxième date"
   saisir jour2, mois2, an2
   (* appel de l'action paramètrée *)
   quantième (jour1, mois1, an1, quant1)
   (* appel de l'action paramètrée *)
   quantième (jour2, mois2, an2, quant2)
   (* édition de la différence *)
   afficher "la différence en nombre de jours entre "
   afficher jour1, "/", mois1, "/", an1, "et "
   afficher jour2, "/", mois2, "/", an2, "est "
   afficher ABSOLU(quant1 - quant2)
fin
```

```
procédure quantième (données : JOUR, MOIS, AN
                     résultat : QUANT)
const tmois(1:12) : tableau d'entiers
            c'est 0,31,59,90,120,151,181,212,243,273,304,334
début
   (* reprendre l'algorithme de l'action quantième de l'exercice 2 *)
fin
```

En supposant qu'un algorithme principal réalise l'appel suivant :
"quantième(12, 03, 84, VALQUANTIEME)", nous nous proposons d'analyser le déroulement :



PROGRAMME APPELANT

étape 1 : transmission des paramètres réels

étape 2 : traitement des paramètres formels

étape 3 : transmission du paramètre résultat.

PROGRAMME APPELE

■ Algorithme – exercice 3

Nous nous proposons, à partir de la saisie de trois nombres a, b, c, soit :
- de calculer la factorielle de a,
- de calculer la factorielle de b,
- de calculer la factorielle de c,
- de résoudre l'équation $ax + b = 0$,
- de résoudre l'équation $bx + c = 0$,
- de résoudre l'équation $cx + a = 0$,
- de résoudre l'équation $ax^2 + bx + c = 0$,
- de résoudre l'équation $bx^2 + cx + a = 0$,
- de résoudre l'équation $cx^2 + ax + b = 0$.

L'utilisateur doit faire son choix à partir d'un menu.

MÉTHODE

Nous remarquons que nous allons détailler trois fois le calcul de la factorielle, la résolution d'une équation du premier degré, la résolution d'une équation du second degré.

Aussi, nous utilisons des procédures pour chacun de ces calculs. Nous laissons l'affichage des résultats au niveau du programme appellant afin de respecter l'indépendance des procédures par rapport à leur cadre d'utilisation (les résultats de ces procédures peuvent être exploités dans d'autres calculs). L'algorithme principal affiche, quant à lui, le menu, et permet à l'utilisateur de donner les trois valeurs et de faire son choix d'opération.

La procédure FACTORIELLE aura un paramètre formel en donnée et un en résultat. Nous rappelons que la factorielle de a est égale à : a * (a–1) * (a–2) * ... * 2 * 1 et que factorielle 0 vaut 1.

La procédure DEGRE1 (résolution d'une équation du premier degré) utilise quant à elle trois paramètres formels :

- deux en données, qui reçoivent les coefficients;
- un en résultat, qui contient la solution de l'équation (pour la résolution d'une équation du premier degré, voir algorithme-exercice 8 du premier chapitre).

La procédure DEGRE2 (résolution d'une équation du second degré) utilise cinq paramètres formels :

- trois en données qui reçoivent les coefficients;
- deux en résultats, qui contiennent les solutions.

La résolution d'une équation du second degré, du type $mx^2 + nx + p = 0$ est la suivante :

- si m = 0 : nous sommes ramenés à la résolution d'une équation du premier degré;
- si m <> 0

  alors nous calculons $n^2 - 4mp$ que nous appelons delta
  si delta < 0
  alors il n'existe aucune solution réelle à l'équation
  sinon si delta = 0
  alors il existe une seule solution à l'équation (–n/2m)
  sinon il existe deux solutions à l'équation
  (– n – racine carrée(delta))/(2m)
  (– n + racine carrée(delta))/(2m)

```
programme choixmenu
var  fact, choix, a, b, c : entiers
     sol, sol1, sol2 : caractères
début
   afficher "entrez trois nombres "
   saisir a, b, c
   REPETER
     afficher "voulez-vous : "
     afficher "1 - calcul factorielle de ",a
     afficher "2 - calcul factorielle de ",b
     afficher "3 - calcul factorielle de ",c
     afficher "4 - résoudre l'équation ",a,"x + ",b," = 0"
     afficher "5 - résoudre l'équation ",b,"x + ",c," = 0"
     afficher "6 - résoudre l'équation ",c,"x + ",a," = 0"
     afficher "7 - résoudre l'équation ",a,"x² + ",b,"x + ",c," = 0"
     afficher "8 - résoudre l'équation ",b,"x² + ",c,"x + ",a," = 0"
     afficher "9 - résoudre l'équation ",c,"x² + ",a,"x + ",b," = 0"
     afficher "10 - fin de travail"
```

```
        afficher "entrez votre choix :"
        saisir choix
        (* appel des différentes procédures *)
        SUIVANT choix FAIRE
            1 : factorielle (a,fact)
                afficher  "factorielle ",a," = ",fact
            2 : factorielle (b,fact)
                afficher  "factorielle ",b," = ",fact
            3 : factorielle (c,fact)
                afficher  "factorielle ",c," = ",fact
            4 : degré1 (a,b,sol)
                afficher  "solution de ",a,"x + ",b," = 0 : ",sol
            5 : degré1 (b,c,sol)
                afficher "solution de ",b,"x + ",c," = 0 : ",sol
            6 : degré1 (c,a,sol)
                afficher "solution de ",c,"x + ",a," = 0 : ",sol
            7 : degré2 (a,b,c,sol1,sol2)
                afficher "solution de ",a,"x² + ",b,"x + ",c," = 0 : ",
                         sol1,sol2
            8 : degré2 (b,c,a,sol1,sol2)
                afficher "solution de ",b,"x² + ",c,"x + ",a," = 0 : ",
                         sol1,sol2
            9 : degré2 (c,a,b,sol1,sol2)
                afficher "solution de ",c,"x² + ",a,"x + ",b," = 0 : ",
                         sol1,sol2
            10 : afficher "au revoir"
            sinon : afficher "erreur dans votre choix, recommencez"
        FINSUIVANT
    JUSQU'A choix = 10
fin


procedure factorielle (données : nombre
                       résultat : factor)
début
    factor ← 1
    TANT QUE nombre > 1
              factor ← factor * nombre
              nombre ← nombre - 1
    FTQ
fin


procedure degré1 (données : coef1, coef2
                  résultat : solution)
début
    SI coef1 = 0
    ALORS SI coef2 = 0
          ALORS solution ← "ℝ"
          SINON solution ← "vide"
          FSI
```

```
    SINON solution ← cvchaîne(-b/a)
    FSI
fin


procédure degré2(données : nb1, nb2, nb3
        résultats : solut1, solut2)
var delta : réel
début
   solut2 ← " "
   SI    nb1 = 0
   ALORS degré1(nb2, nb3, solut1)
   SINON delta ← (nb2)^2 - 4*nb1*nb3
         SI delta < 0
         ALORS solut1 ← "vide"
         SINON SI delta = 0
               ALORS solut1 ← cvchaîne((-nb2/(2*nb1))
               SINON solut1 ← cvchaîne((-nb2 - racine(delta))/(2*nb1))
                     solut2 ← cvchaîne((-nb2 + racine(delta))/(2*nb1))
               FSI
         FSI
   FSI
fin
```

III – Les fonctions

DEFINITION

Une fonction est une procédure particulière, qui ne génère qu'un et un seul résultat. Par simplification, cet unique résultat peut être exploité directement dans une instruction.

Exemple :

Si nous reprenons l'exemple de la procédure quantième, qui possède une variable résultat QUANT, nous remarquons qu'il faut déclarer un paramètre réel QUANT1 pour recevoir la valeur du quantième. C'est cette seule variable que nous pouvons afficher :

```
(* programme principal *)
quantième(12,03,84,quant1)
afficher"le quantième du 12/03/84 est ",quant1
```

Nous pouvons traduire la procédure quantième en une fonction. Il n'est pas alors nécessaire de déclarer de variable QUANT1. Nous pouvons directement écrire :

```
(* programme principal *)
afficher "le quantième du 12/03/84 est ",quantième(12,03,84)
```

DÉCLARATION D'UNE FONCTION

```
FONCTION <nom-de-la-fonction>
(DONNEE(S) : <liste des identificateurs formels>) : <type du résultat>
```

La fonction étant elle-même le résultat, nous remplaçons ce qui aurait été la variable formelle résultat par le mot réservé VALRET (valeur de retour).

APPEL DE LA FONCTION

Cet appel doit être contenu dans une instruction (affectation, alternative, itérative...) :

- A ← nom-de-la-fonction(liste des paramètres réels)
- SI nom-de-la-fonction(liste des paramètres réels) > var ALORS action
- TANT QUE nom-de-la-fonction (liste des paramètres réels) = var action
- ...

Quelque soit l'utilisation de la fonction, le système vérifie la compatibilité des types de variables traitées et du type du résultat de la fonction (d'où l'importance de la déclaration de ce type).

❑ Exemple d'application 4

Traduire la procédure quantième sous forme de fonction

RÉPONSE :

```
programme appelant-de-fonctions
var jour1, mois1, an1, jour2, mois2, an2 : entiers
début
   (* saisie de la première date *)
   afficher "donner successivement le jour, le mois, l'année de
             la première date"
   saisir jour1, mois1, an1
   (* saisie de la deuxième date *)
   afficher "donner successivement le jour, le mois, l'année de
             la deuxième date"
   saisir jour2, mois2, an2
   (* édition de la différence *)
   afficher "la différence en nombre de jours entre "
   afficher jour1, "/", mois1, "/", an1, "et "
   afficher jour2, "/", mois2, "/", an2, "est "
   afficher ABSOLU(quantième (jour1, mois1, an1) - quantième
                   (jour2, mois2, an2))
fin


fonction quantième (données : jour, mois, an) : entier
const tmois(1:12) : tableau d'entiers
            c'est 0,31,59,90,120,151,181,212,243,273,304,334
var quant : entier
début
   (* calcul du quantième *)
   quant ← tmois(mois) + jour
   SI an mod 4 = 0 et mois > 2
   ALORS quant ← quant + 1
   FSI
   valret ← quant
fin
```

■ Algorithme – exercice 4

Reprendre l'exercice - algorithme 3 et rechercher, parmi les procédures qu'il contient, celle(s) qui aurai(en)t pu être traduite(s) sous forme de fonction. Ecrire l'(les) algorithme(s) correspondant(s).

MÉTHODE

Deux procédures peuvent être traduites en fonction : l'action "factorielle" et l'action "degré1". En effet, ce sont les seules qui produisent un résultat unique.

```
fonction factorielle (données : nombre) : entier
var fact : entier
début
   fact ← 1
   TANT QUE nombre > 1
     fact ← fact * nombre
     nombre ← nombre - 1
   FTQ
   valret ← fact
fin
```

```
fonction degré1 (données : coeff1, coeff2) : caractères
début
   SI coeff1 = 0
   ALORS  SI coeff2 = 0
          ALORS valret ← "ℝ"
          SINON valret ← "vide"
          FSI
   SINON valret ← cvchaîne (-b/a)
   FSI
fin
```

La structure de choix du programme appellant "choixmenu" est modifiée :

```
SUIVANT choix FAIRE
   1 : afficher "factorielle",a,"=",factorielle (a)
   2 : afficher "factorielle",b,"=",factorielle (b)
   3 : afficher "factorielle",c,"=",factorielle (c)
   4 : afficher "la solution de ",a, "x+",b, "= 0 est", degré1 (a,b)
   etc
```

IV – La récursivité

DÉFINITION :

Une procédure P, qui s'appelle elle-même ou qui appelle une autre procédure P' contenant un appel de P est une procédure récursive. Elle possède 2 propriétés :

- Il doit exister des critères pour lesquels les appels cessent.
- Chaque fois que la procédure s'appelle (directement ou indirectement), elle doit être plus proche de ses critères d'arrêt.

■ Algorithme – exercice 5

Ecrire le CALCUL DE FACTORIELLE sous forme récursive

définition :

$5! = 5 \times 4 \times 3 \times 2 \times 1$

$n! = n \times (n-1) \times (n-2) \times \ldots \times 1$

$n! = n \times (n-1)!$ et $0! = 1$

MÉTHODE

La procédure de calcul de factorielle avec le paramètre (n) appellera la procédure de calcul de factorielle avec le paramètre (n–1) et multipliera son résultat par n. La procédure de calcul avec le paramètre (n–1) fera de même avec (n–2) et ainsi de suite. Il existe un critère pour lequel les appels cessent, c'est n = 0, car on connaît le résultat de factorielle(0). De plus, le paramètre tend vers son critère d'arrêt, car étant décrémenté de 1 à chaque appel, il tend vers la valeur nulle.

```
programme calcul
var nombre, résultat : entiers
début
   afficher "calcul de factorielle : donner un nombre"
   saisir nombre
   fact (nombre,résultat)
   afficher "factorielle ",nombre," = ",résultat
fin


procédure fact   (donnée : n
                  résultat : res)
début
   si n = 0
   alors res ← 1
   sinon (* calcul de factorielle (n-1) *)
         fact  (n-1, res)
               (* multiplication de (n-1)! par n *)
         res ← n * res
   fsi
fin
```

exemple :

Visualisation du traitement pour factoriel (3)

```
1. afficher "calcul de factorielle : donner un nombre"
2. saisir ...    nombre = 3
3. fact (3, résultat)
                        n = 3   res = ?
        1. si n = 0
        3. sinon (* ...
        4. fact (2, res)
                                n = 2   res = ?
                1. si n = 0
                3. sinon (* ...
                4. fact (1, res)
                                        n = 1   res = ?
                        1. si  n = 0
                        3. sinon (* ...
                        4. fact (0, res)
                                                n = 0   res = ?
                                1. si  n = 0
                                2. alors        res <-- 1
                                7. fsi
                                8. fin
                        5. (* multipl.
                        6. res <-- 1 * 1
                        7. fsi
                        8. fin
                5. (* multipl.
                6. res <-- 2 * 1
                7. fsi
                8. fin
        5.      (* multipl.
        6.      res <-- 3 * 2
        7.      fsi
        8.      fin
4. affich ...
                                factoriel 3 = 6
5. fin
```

Cette procédure peut être écrite en fonction, le programme principal devient :

```
programme calcul
var n : entier
début
   afficher "donner un nombre"
   saisir nombre
   afficher "factorielle ",nombre," - ",fact (nombre)
fin

fonction fact (données : n) : entier
début
   si n = 0
   alors valret ← 1
   sinon valret ← n * fact (n-1)
   fsi
fin
```

Algorithme – exercice 6

Ecrire l'algorithme de la fonction récursive permettant de calculer un terme de la suite de Fibonacci.

SUITE FIBONACCI–> si n = 0 ou n = 1 alors Fn = n
si n > 1 alors Fn = Fn – 2 + Fn – 1

```
fonction fibo (donnée : n) : entier
début
   si    n = 0 ou n = 1
   alors valret ← n
   sinon valret ← fibo(n - 2) + fibo(n - 1)
   fsi
fin
```

Algorithme – exercice 7

Ecrire l'algorithme de la fonction récursive permettant d'effectuer une RECHERCHE DICHOTOMIQUE d'un élément dans un tableau tiré.

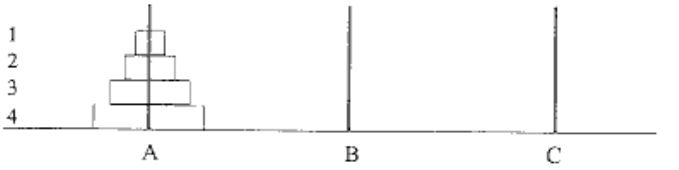
MÉTHODE

La recherche a déjà été faite (chapitre 2 algorithme - exercice 19) sans utiliser la récursivité. Pour la rendre récursive, il faut considérer qu'après la première sélection d'une moitié du tableau, le même procédé est réutilisé sur ce demi-tableau. Le sous-programme dicho doit rendre comme résultat l'indice du tableau dont la valeur est égale ou strictement supérieure à celle recherchée.

```
fonction dicho (données : élément, tableau, début, fin) : entier
var milieu : entier
début
   milieu ← ent ((début + fin) / 2)
   si début = fin
   alors valret ← début
   sinon si élément > tableau(milieu)
         alors valret ← dicho(élément,tableau,milieu+1,fin)
         sinon valret ← dicho(élément,tableau,début,milieu)
         fsi
   fsi
fin
```

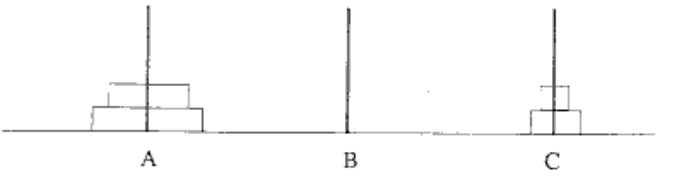
■ Algorithme – exercice 8

Un jeu ancien consiste à déplacer une pile de disques de diamètres décroissants d'un piquet sur un autre. La règle oblige à ne déplacer qu'un disque à la fois, à ne le placer que sur un disque de diamètre supérieur, ou un piquet libre, un troisième piquet sert d'intermédiaire.

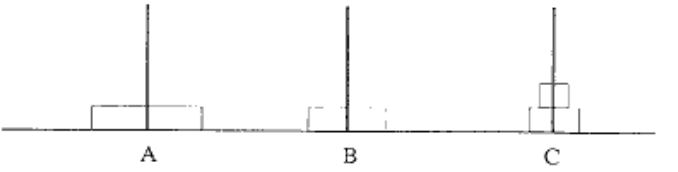


Solution : disque 1 de A vers B, disque 2 de A vers C, disque 1 de B vers C.

A cette étape, nous avons déplacé 2 disques ordonnés de A vers C.

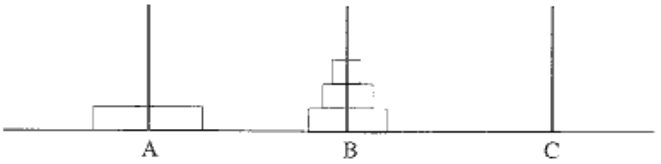


Disque 3 de A vers B.

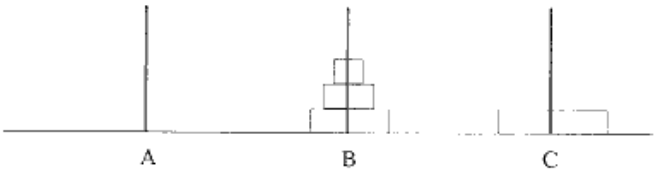


De nouveau, déplacement de 2 disques ordonnés de C vers B.
Disque 1 de C vers A, disque 2 de C vers B, disque 1 de A vers B.

A cette étape, nous avons déplacé 3 disques ordonnés de A vers B.



Disque 4 de A vers C.



Maintenant, il reste à déplacer 3 disques ordonnés de B vers C, nous avons su les déplacer de A vers B, nous savons donc les déplacer de B vers C.

MÉTHODE

Le procédé récursif consiste à exprimer un traitement en fonction d'un traitement plus élémentaire :

Pour déplacer 4 disques, apprenons à déplacer 3 disques.
Pour déplacer 3 disques, apprenons à déplacer 2 disques.
Pour déplacer 2 disques, apprenons à déplacer 1 disque.
Or, déplacer un disque, nous savons le faire !

Déplacer n disques de A vers C, (B piquet auxiliaire), c'est :

déplacer (n–1) disques de A vers B, (C piquet auxiliaire), puis
1 disque de A vers C, puis
(n–1) disques de B vers C, (A piquet auxiliaire)

La procédure appelée par l'instruction hanoï (n , A , C , B) sera :

```
procédure hanoï (données : nombre, piq-départ, piq-arriv, piq-aux)

début
   si    nombre = 1
   alors afficher "déplacer 1 disque de ",piq-départ," vers",
                  piq-arriv
   sinon hanoï (nombre -1, piq-départ, piq-aux, piq-arriv)
```

```
            afficher "déplacer 1 disque de ",piq-départ," vers ",
                     piq-arriv
            hanoï (nombre - 1, piq-aux, piq-arriv, piq-départ)
        fsi
    fin
```

REMARQUES

Tous ces exercices utilisent des piles (structure de données développées dans le chapitre 4), pour stocker les valeurs de paramètres et l'adresse de retour de chaque sous-programme, ainsi que l'adresse de retour dans le programme principal.

On peut toujours écrire un algorithme itératif sans utiliser la récursivité. Dans le cas de l'exercice sur la tour de hanoi, la solution récursive est bien plus courte que celle itérative.