Conception de base de l'algorithme

I - Notions générales

A - Définitions

Un algorithme peut être défini comme la séquence d'action permettant d'arriver à un résultat déterminé à partir d'une situation donnée. Il doit respecter les règles suivantes :

1. il est défini sans ambiguïté
2. il se termine après un nombre fini d'opérations
3. toutes les opérations en doivent pouvoir être effectuées par un homme utilisant des moyens manuels
4. il manipule des objets qui doivent être définis de façon très précise.

B - Les objets

Les objets d'entrée : données fournies à l'algorithme
Les objets de sortie : résultats produits par l'algorithme
Les objets internes : ce sont les objets servant aux manipulations internes de l'algorithme

Le traitement d'un objet concerne la valeur de cet objet.
Un objet dont la valeur ne peut pas être modifiée dans un algorithme est une constante.
Un objet dont la valeur peut être modifiée dans un algorithme est une variable.

Un objet est parfaitement défini si nous connaissons ces trois caractéristiques :

- son identificateur (nom de l'objet) : il est représenté par une vie quelconque de caractères alphanumériques commençant obligatoirement par une lettre.
- sa valeur (constantes ou variables)
- son type : le type d'un objet est défini par un ensemble de valeurs constantes et par l'ensemble des opérations que nous pouvons appliquer à ces constantes :

- Réel ;
- Entier :
  - DIV : division entière (partie entière du quotient)
    5 DIV 2 = 2
  - / : division réelle
    5 / 2 = 2,5
  - MOD : le reste de la division entière
    5 MOD 2 = 1
- Booléen
  - opérateurs : et / ou / non
- Caractères :
  - représenté en code ASCII
  - permet d'avoir une relation d'ordre
    Exemples : « A » < « a » *ASCII : 65 < 97*
    « A » < « Z »ASCII : 65 <90
- chaîne de caractères : on considère comme prédéfini certaines fonctions
  - opérateurs :
    - LONGUEUR (chaîne) : nombre de caractères de la chaîne
      Exemple : « BON » & « JOUR » = BONJOUR

Tous les objets utilisés dans un algorithme doivent être déclarés pour cela nous déterminons quels sont les objets de valeurs constantes et variables.

Les constantes :
CONST nom_objet = valeur
Les variables :
VAR nom_objet : type

Remarque : - si plusieurs variables sont de même type nous pouvons les regrouper sur une même ligne ;
- les constants sont déclarés en premier ;
- les termes réservés CONST et VAR ne figure qu'une seule fois dans l'algorithme ;

Exemples : CONST max. : 50
min. : 10
VAR jour, mois, année : chaîne de caractères

C - Formalisme d'un algorithme

Un algorithme sera défini par :
- un nom ;
- déclarations des variables et des constantes ;
- les actions constituant le traitement a exécuté seront délimitées par les termes : DEBUT et FIN ;

Remarque : afin de permettre une plus grande visibilité, il faudra utiliser des commentaires délimités par les sigles /*commentaires*/.

Programme nom_pg

```
/*Déclarations des constantes*/
CONST       nom_const = valeur
VAR         nom_var : type

Début
  | Action 1
  | Action 2
  | etc.
Fin
```

D - Comment trouver un algorithme ?

Question à me poser :
- quelles sont les données dont on dispose ?
- quelles sont les résultats que l'on doit obtenir ?

Comment obtenir ces résultats :
- appliquer les règles de gestion ;
- traité à plusieurs exemples significatifs (attention aux cas particuliers) ;
- écrire les actions nécessaires pour le traitement.

E - La validité d'un algorithme

Pour être valable un algorithme doit répondre aux critères suivants :
-- le résultat donné doit être le résultat attendu, c'est-à-dire que le programme doit donner le bon résultat dans tous les cas de figures, il faut donc procéder à des tests par l'intermédiaire de différents essais ;
-- l'algorithme doit s'arrêter une fois sa tâche accomplie ;
-- le résultat doit être donné dans un temps acceptable ;
-- l'algorithme doit gérer au mieux la mémoire de l'ordinateur.

II - Instructions élémentaires

A – Affectation

nom_var ← valeur (permet d'affecter une valeur à une variable)

La valeur peut être :
-- une variable du même que nom_var ;
-- une constante du dit nom_var ;
-- une expression dont l'évaluation produit un résultat du type nom_var

Exemples :

```
VAR     R : entier
R ← 10
VAR     X, Y : entier
R ← Y           R prend la valeur de X
R ← X + Y       X ← R           X prend la valeur de R
                R ← R +1        on évalue R + 1 avec l'ancienne valeur de R et on range le résultat dans R.
```

Exemple : écrire un algorithme permettant de calculer le nombre de caractères du mot DUPONT.

Programme Compte v.1

```
CONST       nom : « Dupont »
VAR         compte : entier

Début
   |  compte ← longueur (nom)
Fin
```

Deux inconvénients :
-- pas d'affichage du résultat ;
-- travaille avec le même mot ;

B - Instructions d'entrée / sortie

Nous disposerons d'une instruction de saisie qui permet de récupérer une valeur entrée au clavier et d'une instruction d'affichage qui permet l'édition d'une valeur à l'écran.

1 -- Instructions d'entrée

Saisir (nom_var)
nom_var est une variable déclarée

Exemple : VAR R : entier que
Saisir (R)

2 -- Instructions de sortie

Afficher (valeur)

La valeur peut être :
-- une variable ;
-- une expression ;
-- une constante.

Exemples : VAR R, X, Y : entier

Afficher (« Bonjour »)	Bonjour
Afficher (R)	Valeur de R
Afficher (3 + 2)	5
Afficher (« 3 + 2 = 5 »)	3 + 2 = 5
Afficher (« 3 + 2 = 3 + 2 »)	3 + 2 =, 3 + 2
Afficher (« 3 + 2 = », 3 + 2)	3 + 2 =, 5
Afficher (X + Y)	Valeur de X + Y
Afficher (« X + Y »)	X + Y
Afficher (« La somme est de : », X + Y	La somme est de, X + Y

Remarque : dans le cas d'un dialogue via le clavier, il convient d'afficher des messages facilitant la communication.

Exemple : écrire un algorithme permettant de calculer et d'afficher le nombre de caractères d'un mot quelconque.

Programme Compte v 1.1

```
Var     compte : entier
        nom : chaîne de caractères

Début
  | Afficher (« Quel est le nom ? »)
  | Saisir (nom)
  | compte ← longueur (nom)
  | Afficher (« Le nombre de caractères et de : », compte)
Fin
```

Exemple : écrire un algorithme permettant d'afficher la table des carrés de 1 à 5 ;
Résultats à afficher : 1^2 = 1, 2^2 = 4, 3^2 = 9, 4^2 = 16, 5^2 = 25 ;

```
Début
  | Afficher (« 1^2 = », 1^2)
  | Afficher (« 2^2 = », 2^2)
  | Afficher (« 3^2 = », 3^2)
  | Afficher (« 4^2 = », 4^2)
  | Afficher (« 5^2 = », 5^2)
Fin
```

```
Début
  | Afficher (« 1^2 = 2 »)
  | Afficher (« 2^2 = 4 »)
  | Afficher (« 3^2 = 9 »)
  | Afficher (« 4^2 = 16 »)
  | Afficher (« 5^2 = 25 »)
Fin
```

III -- Les structures conditionnelles

A -- La structure alternative

```
Si      condition alors
  |     action 1
Sinon   action 2
Fsi
```

Action 1 et action 2 peuvent être :
-- une instruction ;
-- un ensemble d'instructions ;
-- un algorithme.

Exemple : écrire un algorithme permettant de calculer et d'afficher la valeur absolue et la différence entre le nombre.

Programme Calcul v 1

```
Var     x, y, dif, abs : réel

 Début
    |    Afficher (« Veuillez saisir le premier nombre »)
    |    Saisir (x)
    |    Afficher (« Veuillez saisir le deuxième nombre »)
    |    Saisir (y)
    |    Si     dif < 0 alors
    |     |     abs ← dif * - 1
    |     |     sinon abs ← dif
    |    Fsi
    |    Afficher (« La valeur absolue est de », abs)
  Fin
```

Remarque : il n'est pas obligatoire de spécifier une partie sinon.

B -- La structure conditionnelle

```
Si   condition alors
 |   action
Fsi
```

Action peut être :

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

Programme Calcul v 1.1

```
Var     x, y, dif, abs : réel

 Début
    |    Afficher (« Veuillez saisir le premier nombre »)
    |    Saisir (x)
    |    Afficher (« Veuillez saisir le deuxième nombre »)
    |    Saisir (y)
    |    dif ← x - y
    |    Si     dif < 0 alors
    |     |     abs ← dif * - 1
    |    Fsi
    |    Afficher (« La valeur absolue est de », dif)
  Fin
```

C -- Imbrication de Si

Il se peut un dans certains cas que l'expression d'un si ne suffise pas à exprimer tout les cas de figure.

```
Si     condition 1 alors
 |       Si   condition 2 alors
 |        |   action 1
 |       Fsi
Sinon
 |       Si   condition 3 alors
 |        |   action 2
 |       Sinon
 |         |   Si  condition 4 alors
 |         |    |  action 3
 |         |   Fsi
 |       Fsi
Fsi
```

Exemple : écrire un algorithme permettant d'afficher la tarification d'une lettre en fonction de son poids

- *poids <= 20g : 0,50 €*
- *20 < poids <= 50g = 0,70 €*
- *Poids > 50g : 1,10 €*

Programme Tarification colis

(Sans constantes)

```
VAR             poids, prix : réel

 Début
   |    Afficher (« Veuillez saisir le poids »)
   |    Saisir (poids)
   |    Si      poids <= 20 alors
   |     |      prix ← 0,50 €
   |    Sinon
   |      |      Si  poids <= 50 alors
   |      |       |  prix ← 0,70 €
   |      |       |  sinon
   |      |       |       prix ← 1,10 €
   |      |      Fsi
   |    Fsi
   |    Afficher (« Le prix du colis est de », prix, « . »)
  Fin
```

(Avec constantes)

```
CONST           poids1 = 20
                poids2 = 50
                prix1 = 0,50 €
                prix2 = 0,70 €
                prix3 = 1,10 €

VAR             poids, prix : réel

 Début
   |    Afficher (« Veuillez saisir le poids »)
   |    Saisir (poids)
   |    Si      poids <= poids1 alors
   |     |      prix ← prix1
   |    Sinon
   |      |      Si  poids <= poids2 alors
   |      |       |  prix ← prix2
   |      |       |  sinon
   |      |       |       prix ← prix3
   |      |      Fsi
   |    Fsi
   |    Afficher (« Le prix du colis est de », prix, « . »)
  Fin
```

Exemple : écrire une algorithme permettant en nous vous de demander de demander le numéro du mois en cours et d'afficher le nom du mois correspondant.

Programme mois v 1

```
Var     numéro : entier
        mois : chaîne de caractère

Début
  | Afficher (« Entrez le numéro du mois »)
  | Saisir (numéro)
  |   Si    (numéro < 1) ou (numéro > 12) alors
  |   |     Afficher (« Erreur de saisie »)
  | Sinon
  |   |   Si    numéro = 1 alors
  |   |   |     mois ← « Janvier »
  |   |  Sinon
  |   |   |   Si    numéro = 2 alors
  |   |   |   |     mois ← « Février »
  |   |   |  Sinon
  |   |   |   |   Si    numéro = 3 alors
  |   |   |   |   |     mois ← « Mars »
  |   |   |   |  Sinon
  |   |   |   |   |   Si    numéro = 4 alors
  |   |   |   |   |   |     mois ← « Avril »
  |   |   |   |   |  Sinon
  |   |   |   |   |   |   Si    numéro = 5 alors
  |   |   |   |   |   |   |     mois ← « Mai »
  |   |   |   |   |   |  Sinon
  |   |   |   |   |   |   |   Si    numéro = 6 alors
  |   |   |   |   |   |   |   |     mois ← « Juin »
  |   |   |   |   |   |   |  Sinon
  |   |   |   |   |   |   |   |   Si    numéro = 7 alors
  |   |   |   |   |   |   |   |   |     mois ← « Juillet »
  |   |   |   |   |   |   |   |  Sinon
  |   |   |   |   |   |   |   |   |   Si    numéro = 8 alors
  |   |   |   |   |   |   |   |   |   |     mois ← « Août »
  |   |   |   |   |   |   |   |   |  Sinon
  |   |   |   |   |   |   |   |   |   |   Si    numéro = 9 alors
  |   |   |   |   |   |   |   |   |   |   |     mois ← « Septembre »
  |   |   |   |   |   |   |   |   |   |  Sinon
  |   |   |   |   |   |   |   |   |   |   |   Si    numéro = 10 alors
  |   |   |   |   |   |   |   |   |   |   |   |     mois ← « Octobre »
  |   |   |   |   |   |   |   |   |   |   |  Sinon
  |   |   |   |   |   |   |   |   |   |   |   |   Si    numéro = 11 alors
  |   |   |   |   |   |   |   |   |   |   |   |   |     mois ← « Novembre »
  |   |   |   |   |   |   |   |   |   |   |   |  Sinon
  |   |   |   |   |   |   |   |   |   |   |   |   |   Si    numéro = 12 alors
  |   |   |   |   |   |   |   |   |   |   |   |   |   |     mois ← « Decembre »
  |   |   |   |   |   |   |   |   |   |   |   |   |   Fsi
  |   |   |   |   |   |   |   |   |   |   |   |   Fsi
  |   |   |   |   |   |   |   |   |   |   |   Fsi
  |   |   |   |   |   |   |   |   |   |   Fsi
  |   |   |   |   |   |   |   |   |   Fsi
  |   |   |   |   |   |   |   |   Fsi
  |   |   |   |   |   |   |   Fsi
  |   |   |   |   |   |   Fsi
  |   |   |   |   |   Fsi
  |   |   |   |   Fsi
  |   |   |   Fsi
  |   |   Fsi
  |   Fsi
  |   Afficher (« Le mois est  », mois, «.»)
Fin
```

D -- Structures de choix

```
Suivant variables faire
    |   valeur 1 : action 1
    |   valeur 2 : action 2
    |   ect.
    |   Valeur n : action n
[sinon action]
Finsuivant
```

Action 1, action 2, … peut être :
-- une instruction ;
-- un ensemble d'instructions ;
-- un algorithme ;

Remarque : plusieurs valeurs peuvent entraîner un même traitement.

Exemple :

```
Suivant nombre faire
    |   0 : afficher (« Nul »)
    |   1, 3 ,5 : afficher (« Impair »)
    |   2, 4, 6 : afficher (« Pair »)
Finsuivant
```

Exemple : écrire une algorithme permettant en nous vous de demander de demander le numéro du mois en cours et d'afficher le nom du mois correspondant.

Programme mois v 1.1

```
Var     numéro : entier
        mois : chaîne de caractère

 Début
   |  Afficher (« Entrez le numéro »)
   |  Saisir (numéro)
   |  Suivant numéro faire
   |     |  1 : mois ← « janvier »
   |     |  2 : mois ← « février »
   |     |  3 : mois ← « mars »
   |     |  4 : mois ← « avril »
   |     |  5 : mois ← « mai »
   |     |  6 : mois ← « juin »
   |     |  7 : mois ← « juillet »
   |     |  8 : mois ← « août »
   |     |  9 : mois ← « septembre »
   |     |  10 : mois ← « octobre »
   |     |  11 : mois ← « novembre »
   |     |  12 : mois ← « décembre »
   |  Sinon
   |     |  Afficher (« erreur programme »)
   |  Finsuivant
   |  Afficher (« mois »)
  Fin
```

Exemple : un robot conduit une voiture, ils peuvent exécuter trois actions, « s'arrêter », « ralentir », « passer » en fonction de la couleur des feux qui sera saisi.

Programme robot v 1

```
Var    couleur : chaîne de caractères

 Début
    |  Afficher (« saisir la couleur »)
    |  Saisir (couleur)
    |  Suivants  couleur    faire
    |     |  vers : afficher (« passer »)
    |     |  orange : afficher (« ralentir »)
    |     |  rouge : afficher (« s'arrêter »)
    |  Sinon
    |     |  Afficher (« veuillez saisir une couleur correcte »)
    |  Finsuivant
   Fin
```

IV-- Les structures itératives

A -- L'instruction TANT QUE

```
Tant que  condition  faire
   |  Action 1
   |  Action 2
   |  etc.
Fintantque
```

Action 1, action 2, … peut être :
-- une instruction ;
-- un ensemble d'instructions ;
-- un algorithme ;

Cette structure est utilisée lorsque l'action peut être exécutée de zéro en n fois
⇨ instructions permettant la sortie de la boucle

Exemple : écrire un algorithme permettant d'afficher la table des carrés de 1 à 5 ;
Résultats à afficher : 1^2 = 1, 2^2 = 4, 3^2 = 9, 4^2 = 16, 5^2 = 25 ;

Programme carré v 1.1

```
Var    nb, nbcarre : entier

 Début
    |  nb ← 1
    |  Tant que     nb <= 5       faire
    |     |  nbcarre ← nb * nb
    |     |  Afficher ( nb, «^2 = », nbcarre)
    |     |  nb ← nb + 1
    |  FTQ
   Fin
```

Exemple : écrire un algorithme permettant d'afficher la longueur de chaque mot saisi. Nous souhaitons également compter le nombre de mots saisis. L'utilisateur entre autant de mots qui le souhaitent

Programme Mot v 1

```
Var     mot, choix : chaîne de caractères
        nbmot, lgmot : entier

Début
  | nbmot ← 0
  | Tant que  choix <> « N » Faire
  |   | Afficher (« Veuillez saisir un mot. »)
  |   | Saisir (mot)
  |   | lgmot ← LONGUEUR (mot)
  |   | lgmot ← nbmot + 1
  |   | Afficher (« Le mot », mot, « se compose », lgmot, « caractères. »)
  |   | Afficher (« Vous ne vous continuez O/N)
  |   | Saisir (choix)
  | FTQ
  | Afficher (« vous avez saisi », nbmot, « mots. »)
Fin
```

B -- L'instruction répéter

```
Répéter
  | Action
Jusqu'à   condition
```

Action peut être :

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

La condition est testée après l'exécution de l'action définie. Sa structure peut être utilisée lorsque l'action peut être exécutée 1 à n fois.

Exemple : écrire un algorithme permettant d'afficher la table des carrés de 1 à 5 ;
Résultats à afficher : 1^2 = 1, 2^2 = 4, 3^2 = 9, 4^2 = 16, 5^2 = 25 ;

Programme carré v 1.2

```
Var   nb: entier

Début
  | nb ←  1
  | Répéter
  |   | Afficher ( nb, «^2 = », nb * nb)
  |   | nb ← nb + 1
  | Jusqu'à      nb = 6
Fin
```

Exemple : écrire un algorithme permettant de calculer l'âge d'une personne à partir de son année de naissance. Nous recommençons autant de fois que possible.

Programme age v 1

```
Const     anné_cours = 2005
Var       anné_naiss, âge : entier
          choix : chaîne de caractères

 Début
   | Répéter
   |   | Afficher (« Quel est l'année de naissance »)
   |   | Saisir (anné_naiss)
   |   | âge ← anné_naiss * anné_cours
   |   | Afficher (« La personne née en », anné_naiss, « à », âge, « ans. »)
   |   | Afficher (« Voulez vous continuez O/N)
   |   | Saisir (choix)
   | Jusqu'à   choix = « N » ou choix = « n »
  Fin
```

Gestion des erreurs de saisie :

Première méthode :

```
Répéter
  | Afficher («  »)
  | Saisir (identifiant)
Jusqu'à   condition
```

Deuxième méthode :

```
Afficher («  »)
Saisir (identifiant)
Tant que  condition    Faire
  | Afficher («  »)
  | Saisir (identifiant)
FTQ
```

Exemple : écrire un algorithme (2 versions) permettant de saisir un nombre compris entre un et 100.

Programme nombre v 1

```
Var       nbre : réel

 Début
   | Répéter
   |   | Afficher (« Veuillez saisir un nombre compris entre 1 et 100 : »)
   |   | Saisir (nbre)
   | Jusqu'à   nbre >= 1 et nbre <= 100
  Fin
```

Programme nombre v 2

```
Var         nbre : réel

 Début
    |  Tant que  nbre <= 1 ou nbre > 100    faire
    |     |  Afficher (« Entrez un nombre entre un et 100 : »)
    |     |  Saisir (nbre)
    |  FTQ
  Fin
```

C -- L'instruction pour

```
Pour    identifiant   De    valeur-initiale   A     valeur-finale    [PAS DE incrément] Faire
   |  Action
Finpour
```

Action peut être :
-- une instruction ;
-- un ensemble d'instructions ;
-- un algorithme ;

Cette structure permet de répéter une action un nombre connu de fois.
Identificateur identifiant est de type entier
Valeur-initiale et valeur-finale sont des constantes ou des variables.
Incrément : valeur d'augmentation ou de diminution progressive de l'identifiant
(entier positif ou négatif) par défaut 1

Exemple : écrire un algorithme permettant d'afficher la table des carrés de 1 à 5 ;
Résultats à afficher : 1^2 = 1, 2^2 = 4, 3^2 = 9, 4^2 = 16, 5^2 = 25 ;

Programme carré v 1.3

```
Var    nb, nbcarre: entier

 Début
    |  Pour     nbre     De     1    A    5    Faire
    |     |  nbcarre ⟵ nb * nb
    |     |  Afficher ( nb, «^2 = », nbcarre)
    |  Finpour
  Fin
```

Exemple : écrire un algorithme permettant d'afficher
*1 * 9 = 9 3 * 9 = 27 5 * 9 = 45 7 * 9 = 18 9 * 9 = 91*

Programme calcul v 1

```
Var    nbre, résultat : entier

 Début
    |  Pour     nbre     De     1    A    9    Pas de     2    Faire
    |     |  résultat ⟵ nbre * 9
    |     |  Afficher (nbre, «* 9 = », résultat)
    |  Finpour
```

Fin

Exemple : écrire un algorithme permettant d'afficher les tables de multiplication de 1 à 9 pour les 10 premiers nombres entiers

Programme Table v 1

```
Var    nbre, table : entier

Début
   | Pour    table   De   2   A   9   Faire
   |   | Pour  nb    De   1   A   10  Faire
   |   |   | afficher (nb, « * », table, « = », nb * table)
   |   | Finpour
   | Finpour
  Fin
```

D -- Comparaison entre les différentes itérations

- ⇨ on utilise la boucle « Pour » quand l'on connaît le nombre d'itérations à l'avance ;
- ⇨ on utilise « Répéter » quand l'on ne connaît pas le nombre d'itérations à l'avance et que l'on doit entrer au moins une fois dans la boucle ;
- ⇨ on utilise « Tant que » lorsqu'on ne connaît pas le nombre d'itérations à l'avance et que l'on n'est pas sur de rentrer au moins une fois dans la boucle.

Les tableaux

I – Les tableaux

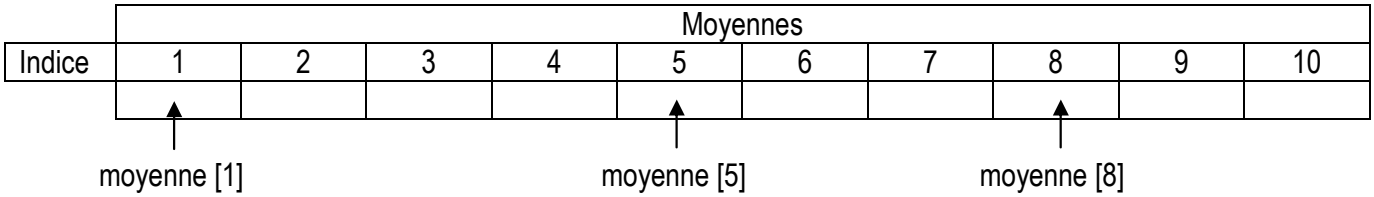
A - Définitions

C'est une structure de données linéaires qui permet de stocker des données de même type. Chacune des valeurs est repérée par un indice indiquant la position de la donner dans le tableau.

Var nom_tableau TABLEAU [indice_min..indice_max]De type donné

Exemple : déclaration d'un tableau permettant de stocker les moyennes de types réels.

VAR moyenne TABLEAU [1..10] De Réel



VAR i : entier
moyenne [i] contenu de la $i^{ème}$ case du tableau à condition que i est une valeur comprise entre 1 et 10.

B - Création d'un tableau

Elle consiste en un remplissage des différentes cases qui le constituent.

Exemple : écrire un algorithme permettant de saisir une saisie de 10 entiers dans un tableau.

Programme Tableau

```
Var     valeur TABLEAU [1..10] De Réels
        i : entier

 Début
    |  Pour     i de 1 à 10 faire
    |     |  Afficher (« Donner la valeur. »)
    |     |  Saisir (valeur [i])
    |  Fin pour
   Fin
```

C - Édition d'un tableau

Il s'agit d'afficher le contenu du tableau c'est-à-dire le contenu de ses différents cases.

Exemple : écrire un algorithme permettant d'afficher le contenu du tableau suivant

1	2	3	……	26
a	b	c	……	z

Nous non que le tableau a déjà été saisi.

Programme Editer

```
Var     valeur TABLEAU [1..26] De caractères
        i : entier

 Début
    |  Pour     i de 1 à 26 faire
    |     |  Afficher (« Valeur. », valeur [i])
    |  Fin pour
  Fin
```

D - Recherche d'un élément dans un tableau

Il arrive fréquemment qu'un utilisateur désire vérifier la présence d'un élément dans un tableau ou bien qu'il souhaite connaître la place d'un élément dans un tableau.
Dans tous les cas, il faut considérer que le tableau peut ne pas contenir tous les éléments recherchés.

Exemple : écrire un algorithme permettant de rechercher dans un tableau contenant les 20 reçus au BTS, le nom d'un étudiant saisie afin de déterminer s'il reçut ou recalés. Vous effectuerez la saisie du tableau préalable. On suppose qu'il n'y a pas d'homonyme.

Programme Recherche v 1.0

```
Var     étudiant TABLEAU [1..2O] De Chaîne de caractères
        i : entier

 Début
    |  Pour     i de 1 à 20 faire
    |     |  Afficher (« Saisir un nom »)
    |     |  saisir (étudiant [i])
    |  Fin pour
    |  Afficher (« Saisir un nom recherché : »)
    |  Saisir (nom)
    |  i ← 0
    |  Répéter
    |     |  i ← i + 1
    |  Jusqu'à (nom = étudiant [i]) ou (i = 20)
    |  Si nom = étudiant [i] Alors
    |     |  Afficher (« Reçues »)
    |  Sinon
    |     |  Afficher (« Recalés »)
    |  FSI
  Fin
```

Programme Recherche v 1.1

```
Var     étudiant TABLEAU [1..2O] De Chaîne de caractères
        i : entier

 Début
    |  Pour     i de 1 à 20 faire
    |    |  Afficher (« Saisir un nom »)
    |    |  saisir (étudiant [i])
    |  Fin pour
    |  Afficher (« Saisir un nom recherché : »)
    |  Saisir (nom)
    |  i ← 1
    |  Tant que (nom <> étudiant [i]) et (i < 20) Faire
    |    |  i ← i + 1
    |  FTQ
    |  Si nom = étudiant [i] Alors
    |    |  Afficher (« Reçues »)
    |  Sinon
    |    |  Afficher (« Recalés »)
    |  FSI
  Fin
```

Programme Recherche v 1.2

```
Var     étudiant TABLEAU [1..2O] De Chaîne de caractères
        i : entier
        trouve : booléen

 Début
    |  Pour     i de 1 à 20 faire
    |    |  Afficher (« Saisir un nom »)
    |    |  saisir (étudiant [i])
    |  Fin pour
    |  Afficher (« Saisir un nom recherché : »)
    |  Saisir (nom)
    |  i ← 1
    |  trouve ← Faux
    |  Répéter
    |    |   Si  nom = étudiant [i] alors
    |    |    |  trouve ← Vrai
    |    |  FSI
    |  Jusqu'à          (trouve)  ou (i = 20)
    |   SI trouve alors
    |    |  Afficher (« Reçus »)
    |  sinon
    |    |  Afficher (« Recalés »)
    |  FSI
  Fin
```

E - Suppression d'un élément dans un tableau

Elle peut se faire :
- en accédant directement à la case par le biais de l'indice que l'on connaît ;
- en recherchant la case supprimer à l'aide de son contenu.

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

Exemple : à partir du tableau des étudiants au BTS, écrire un algorithme permettant de supprimer l'étudiant qui est déplacé par erreur dans le tableau. On suppose que le tableau a été saisi. Le nom de l'étudiant à supprimer sera saisi, on suppose qu'il n'y a pas d'homonyme.

Programme Suppression v 1.0

```
Var     étudiant TABLEAU [1..2O] De Chaîne de caractères
        i, cpt : entier

 Début
    |  Afficher (« Saisir le nom à supprimer : »)
    |  Saisir (nom)
    |  i ← 0
    |  Répéter
    |     |  i ← i + 1
    |  Jusqu'à nom = étudiant [i]
    |  Pour cpt de i à 19 Faire
    |     |  étudiant [cpt] ← étudiant [cpt + 1]
    |  Fin pour
    |  étudiant [20] ← «  »
   Fin
```

II – Tri d'un tableau

À 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 décroissants si le contenu de la case d'indice i est supérieur ou égal au contenu de la case d'indice i + 1

A - Le tri à bulle

Principe : 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 à effectuer.

Exemple : Soit le tableau suivant à trier par ordre croissant

5	**18**	14	4	26
5	**18**	**14**	4	26
5	14	**18**	**4**	26
5	14	4	**18**	**26**
5	**14**	4	18	26
5	**14**	**4**	18	26
5	4	**14**	**18**	26
5	4	14	**18**	**26**
5	**4**	14	18	26
4	**5**	**14**	18	26
4	5	**14**	**18**	26
4	5	14	**18**	**26**
4	**5**	14	18	26
4	**5**	**14**	18	26
4	5	**14**	**18**	26
4	5	14	**18**	**26**

Il n'y a plus d'échange.

Programme TRI_BULLE

```
Const   n = 5
Var     valeur TABLEAU [1..n] D'Entier
        echange : booléen /*détermine si un échange de valeurs a été effectué*/
        inter : réel

Début
  | /* Saisie du tableau */
  | i ← 1
  | Pour i de 1 à n Faire
  |   | Afficher (« Donner un nombre »)
  |   | Saisir (valeur [i])
  | Fin pour
  | /* Tri du tableau */
  | Répéter
  |   | échange ← Faux
  |   | Pour i de 1 à (n – 1) Faire
  |   |   | si valeur [i] > valeur [i + 1] alors
  |   |   |   | échange ← Vrai
  |   |   |   | inter ← valeur [i]
  |   |   |   | valeur [i] ← valeur [i + 1]
  |   |   |   | valeur [i + 1] ← inter
  |   |   | FSI
  |   | Fin pour
  | Jusqu'à non échange
Fin
```

B - Tri croissant par recherches successives des minimums

Principe : soit un tableau de n valeurs

- Recherche du minimum dans le tableau et échange du contenu d'indice 1 et d'indice correspondant à la valeur du minimum.
- Applications du même principe sur (n - 1) valeur (n - premier) pour (n - 1), ... jusqu'à traitement de 2 cases.

Exemple :

8	1	7	5	4
1	**8**	7	5	4
1	**4**	7	5	**8**
1	4	**5**	**7**	8
1	4	5	7	8

Programme Tri_min

```
Const   n = 5
Var     valeur TABLEAU [1..n] D'Entier
        inter, j, indmin, i : réel

 Début
   |  i ← 1
   |  Pour i de 1 à n Faire
   |     |  Afficher (« Donner un nombre »)
   |     |  Saisir (valeur [i])
   |  Fin pour
   |  Pour i de 1 à (n – 1) Faire
   |     |  indmin ← i
   |     |  Pour i de 1 à (n + 1) Faire
   |     |     |    Si  valeur [j] < valeur [indmin] Alors
   |     |     |     |  indmin ← j
   |     |     |  FSI
   |     |  Fin pour
   |     |   Si  indmin <> i Alors
   |     |     |  inter ← valeur [i]
   |     |     |  valeur [i] ← valeur [indmin]
   |     |     |  valeur [indmin] ← inter
   |     |  FSI
   |  Fin pour
  Fin
```

III -- Opérateurs sur un tableau trié

A -- Recherche d'un élément par dichotomie dans un tableau trié

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

- Recherche de l'élément situé à l'indice médian du tableau (milieu du tableau)

- Effectuer une comparaison entre l'élément recherché et l'élément médian. Trois cas peuvent se présenter :
  - élément à chercher = élément médian : arrêt du traitement.
  - élément à chercher < élément médian : on modifie la borne supérieure de recherche dans le tableau (bornesup ← indice médian - 1) car l'élément à chercher est obligatoirement (s'il existe) dans la partie gauche du tableau.
  - élément à chercher > élément médian : on modifie la borne inférieure de recherche dans le tableau (borneinf ← indice médian + 1) car l'élément à chercher est obligatoirement (s'il existe) dans la partie droite du tableau.

Les conditions d'arrêt du traitement sont :
- égalité entre la valeur cherchée et 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.

Exemple 1 : Soit le tableau

1	4	5	7	8

et l'élément à chercher = 7

1	4	5	7	8
borneinf		median		bornesup

élément à chercher > élément médian (7 > 5)
borneinf ← indice médian + 1

1	4	5	7	8
			borneinf médian	bornesup

élément à chercher = élément médian (7 = 7)

Exemple 2 : Soit le tableau

1	4	5	7	8

et l'élément à chercher = 3

élément à chercher < élément médian (3 < 5)
bornesup ← indice médian - 1

1	4	5	7	8
Borneinf		médian		bornesup

1	4	5	7	8
borneinf médian	bornesup			

élément à chercher > élément médian (3 > 1)
borneinf ← indice médian + 1

1	4	5	7	8
	borneinf bornesup médian			

élément à chercher < élément médian (3 < 4)
bornesup ← indice médian -1

1	4	5	7	8
bornesup	borneinf			

3 n'est pas dans le tableau.

Programme Dichotomie

```
Const    n = vous ne vous 5
Var      valeur TABLEAU [1..n] D'Entier
         borneinf, bornesup, median, element : entier

 Début
    |  /* Saisie d'un élément a rechercher */
    |  Afficher (« aucune valeur »)
    |  Saisir (élément)
    |  /* Recherche de l'élément */
    |  Tantque (element <> valeur [median]) et (borneinf < = bornesup) Faire
    |     |        /* Comparaison */
    |     |    Si  (element < valeur[median]) alors
    |     |     |  bornesup ← (median – 1)
    |     |  sinon
    |     |     |  borneinf ← (median + 1)
    |     |    FSI
    |     |  median ← (borneinf + bornesup) DIV 2
    |   FTQ
    |        /* Résultat de la recherche */
    |     Si  (element = valeur[median]) alors
    |     |  Afficher ( element, « trouvé à l'indice médian », median, « . »)
    |   sinon
    |     |  Afficher ( « L'élément »,element, « n'est pas dans le tableau. »)
    |    FSI
  FIN
```

B -- Fusion de deux tableaux

L'opération de fusion de deux tableaux ordonnés consiste à classer les éléments de ces tableaux en les ordonnant dans un troisième tableau.

Table 1	12	14	16	21	34	56	78	90	100
Table 2	5	10	12	13	25	67	78		

Résultats de la fusion en conservant les doublons.

Résultat	5	10	12	12	13	14	16	21	25	34	56	78	78	90	100

Programme Dichotomie

```
Const    n1 =  9
         n2 = 7

Var      table1 TABLEAU [1..n] D'Entier
         table2 TABLEAU [1..n] D'Entier
         tabfusion TABLEAU [1..(n1 + n2)] D'Entier
         indtab1, indtab2, indfus, i = entier

 Début
    |  /* On suppose que table1 et table2 ont été saisies*/
    |  /* Initialisation des indices*/
    |  indtab1 ← 1
    |  indtab2 ← 1
```

```
    |  indfus ← 1
    |  /* Comparaison des valeurs dans les tableaux */
    |  Tantque  (indtab1 < = n1) et (indtab2 < = n2) Faire
    |       |   Si    table1[indtab1] > table2[indtab2] alors
    |       |    |    tabfusion[indtab2] ← table2[indtab2]
    |       |    |    indtab2 ← indtab2 + 1
    |       | sinon
    |       |    |   Si    (table2[indtab2] > table1[indtab1]) alors
    |       |    |    |    tabfusion[indtab2] ← table1[indtab1]
    |       |    |    |    indtab1 ← indtab1 + 1
    |       |    |   FSI
    |       |  FSI
    |       |    indfus ← indfus + 1
    |    FTQ
    |    Pour  i de indtab1 à n1 Faire
    |       |    tabfusion[indfus]  ← table1[i]
    |       |    indfus ← indfus + 1
    |  Fin pour
    |    Pour  i de indtab2 à n2 Faire
    |       |    tabfusion[indfus]  ← table2[i]
    |       |    indfus ← indfus + 1
    |  Fin pour
  FIN
```

IV – Les tableaux à deux dimensions

Chaque élément est repéré par deux indices indiquant la ligne et l'autre la colonne.

```
Var      nom_tableau   TABLEAU [ligne_min .. ligne_max, col_min .. col_max]De type donné
```

Exemple : soit tableau de 3 lignes 5 colonnes

	1	**2**	**3**	**4**	**5**
1	h	j	q	i	o
2	t	k	l	m	e
3	r	d	q	q	a

Écrire un algorithme permettant de saisir des valeurs dans ce tableau.

Programme Saisie

```
Var      valeur TABLEAU [1..3 ,1..5] De caractères
         i, j = entier

 Début
    |    Pour  i de 1 à 3 Faire
    |       |    Pour  j de 1 à 5 Faire
    |       |       |    Afficher (« Entrer une valeur : »)
    |       |       |    Saisir (valeur[i, j])
    |       |  Finpour
    |  Finpour
 Fin en
```

Les enregistrements

Pb : Désire et afficher la moyenne de la note d'oral à et la note d'écrit obtenu au bac de français pour les élèves d'une classe de première. Pour cela, on souhaite également mémoriser le nom et le prénom de chaque élève. Comment mémoriser toutes ces informations afin de les manipuler ?

Solution 1 : utiliser plusieurs tableaux

```
const nb_eleve = 20
nom : tableau [1 .. nb_eleve] de chaîne de caractères
prenom : tableau [1 .. nb_eleve] de chaîne de caractères
oral : tableau [1 .. nb_eleve] de réel
écrit : tableau [1 .. nb_eleve] de réel
```

ou

```
tabnom : tableau [1 .. nb_eleve, 1 .. 2] de chaîne de caractères
tabnote : tableau [1 .. nb_eleve, 1 .. 2] de réel
```

Inconvénient : manipulation de beaucoup de tableaux

Solution 2 : utiliser un tableau

1	2	…	nb_eleve
nom	nom	…	
prenom	prenom	…	
oral	oral	…	
écrit	écrit	…	

I – Définition

Un enregistrement est un type de données structurées, il sert à rassembler dans une même rupture des données de types différents se rapportant à un même sujet.

II – Déclaration

```
TYPE   nom_enregistrement     : ENREGISTREMENT
       donnée 1 : type 1
       donnée 2 : type 2
       …
       donnée n : type n
FIN ENREGISTREMENT
```

type 1, type 2, type n peuvent être différent

Exemple :

```
TYPE   eleve_enr              : ENREGISTREMENT
       nom, prenom : chaîne de caractéres
       oral, écrit : réel
FIN ENREGISTREMENT
```

Remarque : eleve_enr n'est pas une variable mais un nouveau type au même titre que les types standard : réel, entier.

III – Utilisation

Pour utiliser un type enregistrement, il faut déclarer une variable de ce type.

```
VAR nom_var : nom_enregistrement
VAR nom_tab : TABLEAU [ind_min .. ind_max] de nom_enregistrement
```

Exemples :

1) VAR unEleve : eleve_enr
Accès au nom de l'élève : eleve_enr.nom
Accès à la note oral : eleve_enr.oral

2) VAR tabEleve : TABLEAU [1 .. nb_eleve] de eleve_enr
Accès au nom du troisième élève : tabEleve [3].nom
Accès au nom du cinquième élève : tabEleve [5].nom

3) ***Programme Élève***

```
CONST nb =20

TYPE   eleve_enr   : ENREGISTREMENT
         nom, prenom : chaîne de caractéres
         oral, écrit : réel
FIN ENREGISTREMENT

VAR  tabEleve : TABLEAU [1 .. nb] de eleve_enr
       i : entier
       moy : réel

 Début
   |  Pour i de 1 à nb Faire
   |    |  Afficher (« Donner le nom, le prénom, et les deux notes de l'élève  »)
   |    |  Saisir (tabeleve[i].nom
   |    |  Saisir (tabeleve[i].prenom)
   |    |  Saisir (tabeleve[i].oral)
   |    |  Saisir (tabeleve[i].écrit)
   |  Fin pour
   |  Pour i de 1 à nb Faire
   |    |  moy ← (tabeleve[i].oral + tabeleve[i].écrit) / 2
   |    |  Afficher (« La moyenne de l'élève », tabeleve[i].nom, « », tabeleve[i].prenom, « est de »,moy)
   |  Fin pour
  FIN
```

Remarque 1 : un type enr peut contenir un champ de type enregistrement. Supposons que dont le type eleve_enr, nous voulions sa date de naissance :

```
TYPE   date          : ENREGISTREMENT
         jour, mois, année : entier
FIN ENREGISTREMENT

TYPE   eleve_enr              : ENREGISTREMENT
         nom, prenom : chaîne de caractéres
         oral, écrit : réel
         datenaiss : date
FIN ENREGISTREMENT
```

Remarque 2 : type enr peut contenir un champ de type tableau. Supposons que le type eleve_enr est caractérisé par 6 notes :

```
TYPE   eleve_enr              : ENREGISTREMENT
         nom, prenom : chaîne de caractéres
         datenaiss : date
         note : tableau [1 .. 6] de réel
FIN ENREGISTREMENT
```

Remarque 3 : les données contenues dans les enregistrements et les tableaux sont perdus des que l'on quitte le programme dans lequel il est saisi. Solution : ou type de données, le **fichier**.