COURS ALGORITHMIQUE

FRÉDÉRIC GIROD
ELORRI

elorri
Formation
COMPÉTHANCE !

INTRODUCTION

PROGRAMME

- Un programme correspond à une méthode de résolution pour un problème donné.

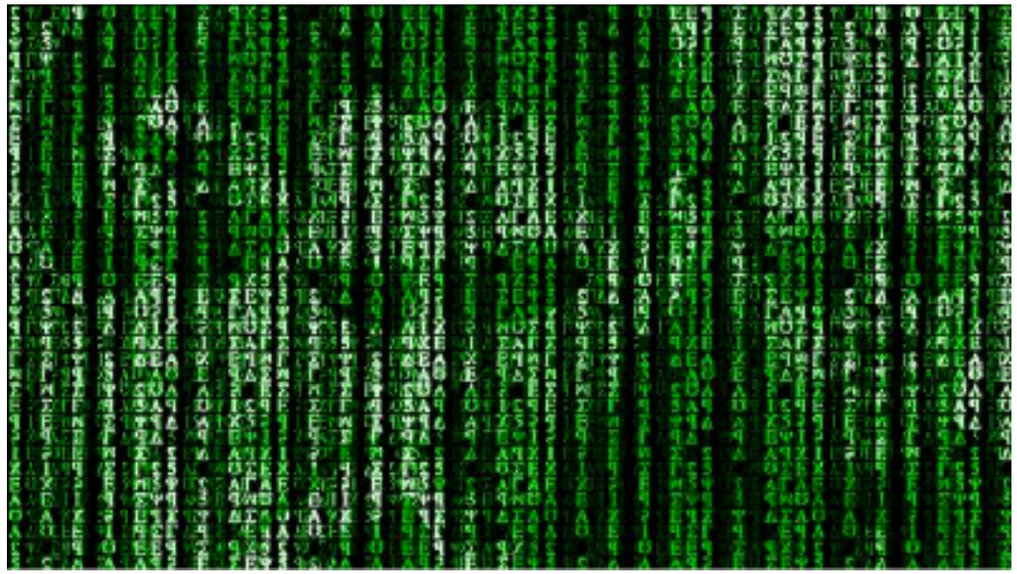
- Cette méthode de résolution est effectuée par une suite d'instructions d'un langage de programmation.

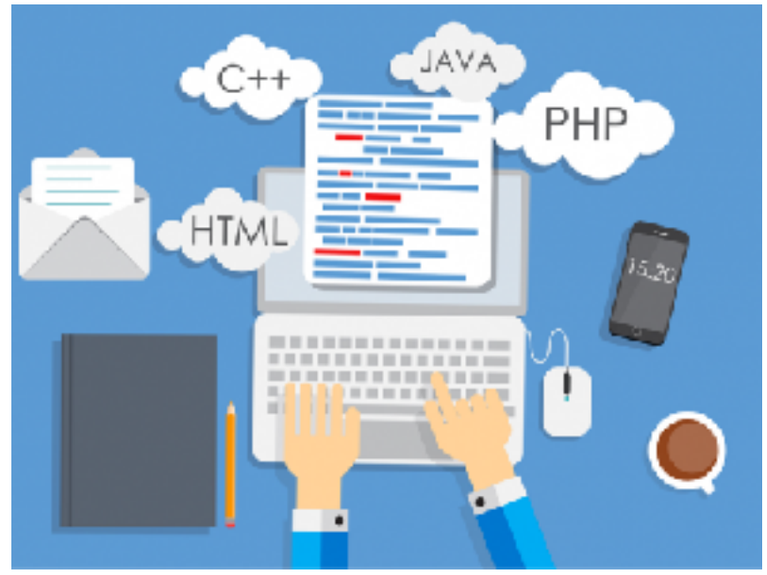
- Ces instructions permettent de traiter et de transformer les données (entrées) du problème à résoudre pour aboutir à des résultats (sorties).

- Un programme n'est pas une solution en soi mais une méthode à suivre pour trouver des solutions.

PROGRAMMATION

- Ensemble des activités orientées vers la conception, la réalisation, le test et la maintenance de programmes





LANGAGES INFORMATIQUES

- Le langage informatique est l'intermédiaire entre le programmeur et la machine.

- Il permet d'écrire des programmes destinés à effectuer une tache donnée.

LANGAGES DE PROGRAMMATION

- Deux types de langages

  ➡ Langages procéduraux : PHP 4, Perl, Cobol, ...

  ➡ Langages orientés objets : C++, Java, C#, PHP 5+, ...

- Le choix d'un langage de programmation n'est pas facile, chacun a ses spécificités et correspond mieux à certains types d'utilisations.

ALGORITHME

- Un algorithme est une suite d'instructions ayant pour but de résoudre un problème donné

- Un algorithme peut se comparer à une recette de cuisine

- Exemples :

  ➡ Préparer une recette de cuisine

  ➡ Montrer le chemin à un touriste

  ➡ …

ALGORITHME

EXEMPLE

29 min (26,9 km)

via A25

Le plus rapide, malgré un trafic ralenti

Prendre Rue du Printemps en direction de Rue Denis Papin

2 min (750 m)

Rejoindre Rue Jean Jaurès/D14

130 m

Au rond-point, prendre la 5e sortie sur Rue du Printemps

450 m

Prendre à droite sur Rue Papin

110 m

Tourner à droite au 1er croisement et continuer sur Rue Denis Papin

Voie à accès limité

Votre destination se trouvera sur la gauche.

78 m

Urbilog (SAS)

31 Rue Denis Papin, 59650 Villeneuve-d'Ascq

ALGORITHME ET PROGRAMMATION

- L'élaboration d'un algorithme précède l'étape de programmation

  ➡ Un programme est un algorithme

  ➡ Un langage de programmation est un langage compris par l'ordinateur

➡ La rédaction d'un algorithme est un exercice de réflexion

ALGORITHME ET PROGRAMMATION

- La rédaction d'un algorithme est un exercice de réflexion

  ➡ L'algorithme est indépendant du langage de programmation

  ➡ Par exemple, on utilisera le même algorithme pour une implantation en Java, en C# ou en PHP…

  ➡ L'algorithme est la résolution brute d'un problème informatique

ALGORITHMIQUE

- L'algorithmique désigne la discipline qui étudie les algorithmes et leurs applications en Informatique.

- Une bonne connaissance de l'algorithmique permet d'écrire des algorithmes efficaces.

- Permet le développement de l'esprit logique.

MÉMO

ALGORITHME = MÉTHODE DE RÉSOLUTION

REPRÉSENTATION D'UN ALGORITHME

- Deux façons de représenter un algorithme :

➡ L'organigramme : représentation graphique avec des symboles (carrés, losanges, etc…)

➡ Le pseudo-code : représentation textuelle avec une série de conventions ressemblant à un langage de programmation

NOTIONS ET INSTRUCTIONS INCONTOURNABLES EN ALGORITHMIQUE

INSTRUCTIONS DE BASE

- Un programme informatique est formé de quatre types d'instructions considérées comme la « base » de la programmation :

➡ l'affectation de variables

➡ la lecture et/ou l'écriture

➡ les conditions

➡ les boucles

NOTION DE VARIABLE

- Une variable sert à stocker la valeur d'une donnée dans un langage de programmation

- Une variable désigne un emplacement dont le contenu peut changer au cours d'un programme (d'où le nom de variable)

NOTION DE VARIABLE

- Une variable se caractérise par :

  ➡ un nom (ou identifiant)

  ➡ un type (indique le type de valeur que peut prendre la variable, exemple : un nombre ou une chaine de caractères)

  ➡ une valeur

- <u>Attention :</u> la variable doit être déclarée avant d'être utilisée.

NOTION DE VARIABLE

- Le choix du nom d'une variable est soumis à quelques règles qui varient selon le langage, mais en général :

➡ doit commencer par une lettre alphabétique

➡ doit être constitué uniquement de lettres, de chiffres et du « tiret du bas » (_), pas d'espaces et de caractères de ponctuation

NOTION DE VARIABLE

- Conseil : pour la lisibilité et la compréhension du code, il est recommandé de choisir des noms significatifs qui décrivent les données manipulées

  ➡ Exemples : NomRecette, Prix_TTC, etc…

TYPES DES VARIABLES

- Le type d'une variable détermine l'ensemble des valeurs qu'elle peut prendre.

- Les types proposés par la plupart des langages sont :

➡ Type numérique

➡ Type booléen

➡ Type caractère

➡ Type chaîne de caractère

DÉCLARATION DES VARIABLES

- En algorithmique, la déclaration de variables est effectuée par la forme suivante :

  ➡ Variables listes des noms des variables : type

- Exemple :

```
Variables
nombre1, nombre2 : entier
porteOuverte : booléen
chaine1, chaine2 : chaîne de caractères
```

AFFECTATION D'UNE VARIABLE

- L'affectation consiste à attribuer une valeur à une variable.
- En algorithmique, l'affectation est notée par le signe <-
- Exemple : maVariable <- 12
- L'affectation ne modifie que ce qui est à gauche de la flèche.

AFFECTATION D'UNE VARIABLE

- La plupart des langages de programmation utilisent le signe égal = pour l'affectation <-

- Lors d'une affectation, l'expression de droite est évaluée et la valeur trouvée est affectée à la variable de gauche

- Exemple : A <- B différent de B <- A

NOTION DE CONSTANTE

- Une constante est une variable dont la valeur ne change pas au cours de l'exécution du programme.

- En algorithmique, la déclaration d'une constante est effectuée par la forme suivante :

  ➡ Constante nom de la constante=valeur : type

- Exemple :
```
Constante
pi=3.14 : réel
nombreMax=128 : entier
```

SYNTAXE GÉNÉRALE D'UN ALGORITHME

```
Algo monPremierAlgo
//commentaire sur une ligne
/* Commentaire sur
plusieurs lignes */
Constantes
const1<-20 : entier
const2<-"bonjour" : chaîne
Variables
var1, var2 : entier
var3 : chaîne
Début //corps de l'algorithme

/*
INSTRUCTIONS
*/

Fin
```

AFFECTATION : EXERCICES

Donnez les valeurs des variables A, B et C après exécution des instructions suivantes ?

```
Variables
A, B, C : entier
Début
A <— 7
B <— 17
A <— B
B <— A+5
C <— A+B
C <— B-A
Fin
```

AFFECTATION : EXERCICES

Donnez les valeurs des variables A et B après exécution des instructions suivantes ?

```
Variables
A, B : entier
Début
A <— 6
B <— 2
A <— B
B <— A
Fin
```

Les deux dernières instructions permettent-elles d'échanger les valeurs de A et B ?

AFFECTATION : ÉCHANGES

Ecrire une algorithme permettant d'échanger les valeurs de deux variables A et B ?

EXPRESSIONS ET OPÉRATEURS

- Une expression peut être une valeur, une variable ou une opération constituée de variables reliées par des opérateurs

- Exemples : 1, b, a*2, a+3*b-c, ...

- L'évaluation de l'expression fournit une valeur unique qui est le résultat de l'opération
  Les opérateurs dépendent du type de l'opération, ils peuvent être :

- Une expression est évaluée de gauche à droite mais en tenant compte des priorités des opérateurs.

EXPRESSIONS ET OPÉRATEURS

- Les opérateurs dépendent du type de l'opération, ils peuvent être :

➡ des opérateurs arithmétiques: +, -, *, /, % (modulo)

➡ des opérateurs logiques: NON(!), OU(| |), ET (&&)

➡ des opérateurs relationnels: =, <, >, <=, >=

➡ des opérateurs sur les chaînes: & (concaténation)

PRIORITÉ DES OPÉRATEURS

- Pour les opérateurs arithmétiques, l'ordre de priorité est le suivant :

  ➡ () : les parenthèses

  ➡ ^ : (élévation à la puissance)

  ➡ * , / (multiplication, division)

  ➡ + , - (addition, soustraction)

- Exemple : 9 + 3 * 4 = ?

- Exemple : (9 + 3) * 4 = ?

LES OPÉRATEURS BOOLÉENS

LA TABLE DE VÉRITÉ...

LES INSTRUCTIONS DE LECTURE

- La lecture permet d'entrer des donnés à partir du clavier
- En algo, on écrit : lire(maVariable)
- La machine met la valeur entrée au clavier dans la variable nommée maVariable

LES INSTRUCTIONS D'ÉCRITURE

- L'écriture permet d'afficher des résultats à l'écran (ou de les écrire dans un fichier)

- En algo, on écrit : écrire(liste d'expressions)

- La machine affiche les valeurs des expressions décrite dans la liste.

- Exemple : écrire(a, b+2, "Bonjour")

LECTURE ET ÉCRITURE : EXEMPLE

- Ecrire un algorithme qui demande un nombre entier à l'utilisateur, puis qui calcule et affiche le carré de ce nombre

```
Algorithme CalculCarre
Variables
A, B : entier
Début
   écrire("entrer la valeur de A")
   lire(A)
   B ← A*A
   écrire("le carre de ", A, "est :", B)
Fin
```

LECTURE ET ÉCRITURE : EXERCICE

- Écrire un algorithme qui permet d'effectuer la saisie d'un nom, d'un prénom et affiche ensuite le nom complet

INSTRUCTIONS CONDITIONNELLES

- Une condition est une expression écrite entre parenthèse à valeur booléenne.

- Les instructions conditionnelles servent à n'exécuter une instruction ou une séquence d'instructions que si une condition est vérifiée.

INSTRUCTIONS CONDITIONNELLES

- En algo, une condition s'écrit :

```
Si condition alors
instruction ou suite d'instructions1
Sinon
instruction ou suite d'instructions2
Finsi
```

INSTRUCTIONS CONDITIONNELLES
EXERCICE

Écrire un algorithme qui demande un nombre entier à l'utilisateur, puis qui teste et affiche s'il est supérieur à 12 ou non

INSTRUCTIONS CONDITIONNELLES IMBRIQUÉES

Les instructions conditionnelles peuvent avoir un degré quelconque d'imbrications

```
Si condition1 alors
  Si condition2 alors
    instructionsA
  Sinon
    instructionsB
  Finsi
Sinon
  Si condition3 alors
    instructionsC
  Finsi
Finsi
```

INSTRUCTIONS CONDITIONNELLES IMBRIQUÉES : EXEMPLE

```
Variable n : entier
Début
   Ecrire ("entrez un nombre : « )
   Lire (n)
   Si n < 0 alors
      Ecrire ("Ce nombre est négatif")
   Sinon
      Si n = 0 alors
         Ecrire ("Ce nombre est nul")
      Sinon
         Ecrire ("Ce nombre est positif")
      Finsi
   Finsi
Fin
```

INSTRUCTIONS CONDITIONNELLES IMBRIQUÉES : EXERCICE

Le prix de la clé USB dans un magasin spécialisé varie selon le volume acheté :

5€ l'unité si le nombre de clé USB à acheter est inférieur à 10.

4€ l'unité si le nombre de clé USB à acheter est compris entre 10 et 20.

3€ l'unité si le nombre de clé USB à acheter est au-delà de 20.

Écrivez un algorithme qui demande à l'utilisateur le nombre de clé USB qu'il souhaite acheter et qui calcule et affiche le prix à payer.

INSTRUCTIONS CONDITIONNELLES COMPOSÉES

- Une condition composée est une condition formée de plusieurs conditions simples reliées par des opérateurs logiques : ET, OU et NON
- Exemple : x compris entre 2 et 6 : (x >= 2) ET (x <=6)
- L'évaluation d'une condition composée se fait selon des règles présentées précédemment dans la table de vérité

INSTRUCTION CAS

- Lorsque l'on doit comparer une même variable avec plusieurs valeurs, on peut remplacer cette suite de si par l'instruction cas

```
cas où maVariable vaut
  valeur1 : action1
  valeur2 : action2
  ...
  valeurn : actionn
  autre : action autre
fincas
```

INSTRUCTION CAS : EXEMPLE

```
Variables c : caractère
Début
   Ecrire("entrer un caractère")
   Lire (c)
   Si((c>='A') et (c<='Z')) alors
     Cas où c vaut
       'A','E', 'I', 'O', 'U', 'Y' :
       Ecrire("c'est une voyelle majuscule »)
       autre : Ecrire("c'est une consonne
       majuscule")
     Fincas
   Sinon
     Ecrire("ce n'est pas une lettre majuscule")
   Finsi
Fin
```

INSTRUCTIONS ITÉRATIVES : LES BOUCLES

- Les boucles servent à répéter l'exécution d'un groupe d'instructions un certain nombre de fois

- On distingue trois sortes de boucles en langages de programmation :

➡ Les boucles tant que : on y répète des instructions tant qu'une certaine condition est réalisée

➡ Les boucles jusqu'à : on y répète des instructions jusqu'à ce qu'une certaine condition soit réalisée

➡ Les boucles pour ou avec compteur : on y répète des instructions en faisant évoluer un compteur (variable particulière) entre une valeur initiale et une valeur finale

LA BOUCLE TANT QUE

```
TantQue (condition)
    instructions
FinTantQue
```

- La condition (dite condition de contrôle de la boucle) est évaluée avant chaque itération

➡ Si la condition est vraie, on exécute les instructions (corps de la boucle), puis, on retourne tester la condition. Si elle est encore vraie, on répète l'exécution.

➡ Si la condition est fausse, on sort de la boucle et on exécute l'instruction qui est après FinTantQue

LA BOUCLE TANT QUE

- Une des instructions du corps de la boucle doit absolument changer la valeur de la condition de vrai à faux (après un certain nombre d'itérations), sinon le programme va tourner indéfiniment

- C'est ce qu'on appelle une boucle infinie

```
i<—1
TantQue (i>0)
  i<—i+1
FinTantQue
```

LA BOUCLE TANT QUE : EXEMPLE

Contrôle de saisie d'une lettre alphabétique jusqu'à ce que le caractère entré soit valable

```
Variable C : caractère
Debut
  Écrire (" Entrez une lettre majuscule ")
  Lire (C)
  TantQue (C < 'A' ou C > 'Z')
    Ecrire ("Saisie erronée. Recommencez")
    Lire (C)
  FinTantQue
  Ecrire ("Saisie valable")
Fin
```

LA BOUCLE TANT QUE : EXERCICE

Ecrire un algorithme qui demande à l'utilisateur un nombre compris entre 1 et 3 jusqu'à ce que la réponse convienne.

LA BOUCLE RÉPÉTER ... JUSQU'À

```
Répéter
    instructions
Jusqu'à condition
```

- La condition est évaluée après chaque itération
- Les instructions entre Répéter et Jusqu'à sont exécutées au moins une fois et leur exécution est répétée jusqu'à ce que la condition soit vraie (tant qu'elle est fausse)

LA BOUCLE POUR

```
Pour compteur allant de début à fin
faire
  instructions
FinPour
```

- Compteur est une variable de type entier. (elle doit être déclarée)

- Début et fin peuvent être des valeurs, des variables définies avant le début de la boucle ou des expressions de même type que compteur

LA BOUCLE POUR : EXEMPLE

```
Pour i allant de 0 à 10 faire
  afficher(i)
FinPour
```

Cette boucle affichera successivement les nombres 0, 1, …, 10, on effectue donc 11 itérations.

LA BOUCLE POUR : EXERCICE

Ecrire l'algorithme permettant d'afficher la table de multiplication par 9.

BOUCLES IMBRIQUÉES

Les instructions d'une boucle peuvent être des instructions itératives. Dans ce cas, on aboutit à des boucles imbriquées.

```
Pour i allant de 1 à 5
  Pour j allant de 1 à i
    écrire("O")
  FinPour
  écrire("K")
  écrire("\n")
FinPour
```

CHOIX D'UN TYPE DE BOUCLE

- Si on peut déterminer le nombre d'itérations avant l'exécution de la boucle, il est plus naturel d'utiliser la boucle Pour
- S'il n'est pas possible de connaître le nombre d'itérations avant l'exécution de la boucle, on fera appel à l'une des boucles TantQue ou répéter jusqu'à

LES FONCTIONS

SYNTAXE D'UNE FONCTION

```
Fonction Nom de la fonction (Liste des paramètres) : Type de résultat
Début
  Instructions de la fonction
  Retourner(résultat)
Fin
```

Les paramètres sont facultatifs, mais s'il n'y pas de paramètres, les parenthèses doivent rester présentes.

L'instruction Retourner renvoie au programme appelant le résultat de l'expression placée à la suite de ce mot clé.

FONCTION : EXEMPLE

Calcul du périmètre d'un rectangle

```
Fonction perimetreRectangle (largeur, longueur : entier) : entier
Variable perimetre : entier
Début
  perimetre<—(2*(largeur+longueur))
  Retourner(perimetre)
Fin
```

APPEL D'UNE FONCTION

Nom de la fonction (Liste des paramètres)

Pour exécuter une fonction, il suffit de faire appel à elle en écrivant son nom suivie des paramètres effectifs.

```
Début
  Variable perimetre : entier
  perimetre <— perimetreRectangle(10,15)
  Afficher("Le périmètre du rectangle
  est : ", perimetre)
Fin
```

FONCTION : EXERCICE

Définir une fonction qui renvoie le plus grand de deux nombres différents.

Ecrire un programme qui demande deux nombres à l'utilisateur et qui affiche le plus grand des deux nombres en appelant la fonction précédemment créée.

PORTÉE DES VARIABLES

- Une variable déclarée dans la partie déclaration de l'algorithme principale est appelée variable globale. Elle est accessible de n'importe où dans l'algorithme, même depuis les fonctions. Elle existe pendant toute la durée de vie du programme.

- Une variable déclarée à l'intérieur d'une fonction est dite locale. Elle n'est accessible que dans cette fonction, les autres fonctions n'y ont pas accès. La durée de vie d'une variable locale est limitée à la durée d'exécution de la fonction.

LES TABLEAUX

PROBLÉMATIQUE

```
Algorithme Note
Variables  N1, N2, N3, N4, N5 : Réel
Début
          Afficher("Entrer la valeur de la 1er note")
           Saisie(N1)
          Afficher("Entrer la valeur de la 2eme note")
          Saisie(N2)
          Afficher("Entrer la valeur de la 3eme note")
          Saisie(N3)
          Afficher("Entrer la valeur de la 4eme note")
          Saisie(N4)
          Afficher("Entrer la valeur de la 5eme note")
          Saisie(N5)
         Afficher("La note 1 multipliée par 3 est : ",N1 * 3)
         Afficher("La note 2 multipliée par 3 est : ",N2 * 3)
         Afficher("La note 3 multipliée par 3 est : ",N3 * 3)
         Afficher("La note 4 multipliée par 3 est : ",N4 * 3)
         Afficher("La note 5 multipliée par 3 est : ",N5 * 3)
Fin
```

DÉFINITION D'UN TABLEAU

- Un tableau est une suite d'éléments de même type.
- Il utilise plusieurs cases mémoire à l'aide d'un seul nom.
- Comme toutes les cases portent le même nom, elles se différencient par un numéro ou un indice.

SYNTAXE D'UN TABLEAU

```
Variable identificateur :
tableau[taille] de type
```

- La taille du tableau doit être un nombre entier.

- Les éléments d'un tableau sont des variables qui s'utilisent exactement comme n'importe quelles autres variables classiques.

UTILISATION D'UN TABLEAU

- Soit le tableau suivant : `Variable Note : tableau[30] de réels`

- L'utilisation de ces éléments se fait via le nom du tableau et son indice. Cet indice peut être soit une valeur (exemple : Note(3)), soit une variable (exemple : Note(i)) ou encore une expression (exemple : Note(i+1)).

- Le premier élément d'un tableau porte l'indice zéro.

UTILISATION D'UN TABLEAU : EXEMPLES

- Exemple 1 : L'instruction suivante affecte à la variable X la valeur du premier élément du tableau Note :

  ➡ X ← Note[0]

- Exemple 2 : L'instruction suivante affiche le contenu de l'élément en quatrième position du tableau Note :

  ➡ ecrire(Note[4])

- Exemple 3 : L'instruction suivante affecte une valeur introduite par l'utilisateur à la position trois du tableau Note :

  ➡ lire(Note[3])

UTILISATION D'UN TABLEAU : EXERCICE

Écrire un algorithme permettant de saisir 30 notes et de les afficher.

TABLEAUX À 2 DIMENSIONS

- Les tableaux à deux dimensions se représentent comme une matrice ayant un certain nombre de lignes (première dimension) et un certain nombre de colonnes (seconde dimension).

- Reprenons l'exemple des notes en considérant cette fois qu'un étudiant a plusieurs notes (une note pour chaque matière). Pour quatre étudiants, nous aurions le tableau de relevés des notes suivant :

	Etudiant 1	Etudiant 2	Etudiant 3	Etudiant 4
Informatique	12	13	9	10
Comptabilité	12.5	14	12	11
Mathématiques	15	12	10	13

SYNTAXE D'UN TABLEAU À 2 DIMENSIONS

```
Variable identificateur :
tableau[nb_lignes,nb_colonnes] de type
```

L'instruction suivante déclare un tableau Note de type réel à deux dimensions composé de 3 lignes et de 4 colonnes :

```
Variable Note : tableau[3,4] de réels
```

UTILISATION D'UN TABLEAU À 2 DIMENSIONS

- Pour accéder à un élément de la matrice (tableau à deux dimensions), il suffit de préciser, entre crochets, les indices de la case contenant cet élément.

- Exemple : L'instruction suivante affecte à la variable X la valeur de l'élément du tableau Note positionné sur la première ligne et la première colonne :

  ➡ X ← Note[0, 0]

PARCOURS D'UN TABLEAU À 2 DIMENSIONS

- Pour parcourir une matrice nous avons besoin de deux boucles, l'une au sein de l'autre, c'est ce qu'on appelle les boucles imbriquées.

- La première boucle est conçue pour parcourir les lignes.

- Tandis que la deuxième est utilisée pour parcourir les colonnes de la ligne précisée par la boucle principale (la première boucle).

TABLEAU À 2 DIMENSIONS : EXERCICE

Écrire un algorithme permettant la saisie des notes d'une classe de 15 étudiants pour 3 matières.

TABLEAU DYNAMIQUE

- Les tableaux définis jusqu'ici sont dits statiques, car il faut qu'au moment de l'écriture du programme le développeur décide de la taille maximale que pourra atteindre le tableau.

- Dans certains situations, on ne peut pas savoir la taille du tableau dans la phase de programmation.

- Les tableaux dynamiques sont des tableaux dont la taille n'est définie que lors de l'exécution. Pour créer un tableau dynamique, il suffit de lui affecter une taille vide.

SYNTAXE D'UN TABLEAU DYNAMIQUE

```
Variable identificateur : tableau[]
de type
```

Comme un tableau dynamique ne possède pas de taille prédéfinie, il convient de redimensionner le tableau avant de pouvoir s'en servir.

```
Redimensionner identificateur[taille]
```

TABLEAU DYNAMIQUE : EXEMPLE

```
Variables
nb en entier
Note[] de type entiers
Début
  Ecrire("Combien y a-t-il de notes à
  saisir ?")
  Lire(nb)
  Redimensionner Note[nb]
  …
Fin
```

EXERCICES COMPLÉMENTAIRES

PRIORITÉ DES OPÉRATEURS

Donner les valeurs des variables a, b et c

```
a <— 4 * 2 + 5
b <— 5 + 3 * 2 - 6
c <— (a > b) ET (b > 2)
d <— (a < b) OU (b > 2)
```

EXERCICES COMPLÉMENTAIRES

TRIANGLE D'ÉTOILES

Ecrire une algorithme qui affiche à l'écran le triangle d'étoiles suivant :

```
*
**
***
****
*****
```