Développement Digital 1ère Année

ISTA NTIC

Algorithmique

Pr Bendahmane

Etapes de réalisation d'un programme

Enoncé du problème

↓ *Spécification*

Cahier des charges

↓ *Analyse*

Algorithme

↓ *Traduction en langage*

Programme source

↓ *Compilation*

Programme exécutable

↓ *Tests et modifications*

Version finale et résultats

La réalisation de programmes passe par l'écriture d'algorithmes
⇒ D'où l'intérêt de l'**Algorithmique**

Algorithmique

- Le terme **algorithme** vient du nom du mathématicien arabe **Al-Khawarizmi** (820 après J.C.)
- Un algorithme est une description complète et détaillée des actions à effectuer et de leur séquencement pour arriver à un résultat donné
  - Intérêt: séparation analyse/codage (pas de préoccupation de syntaxe)
  - Qualités: **exact** (fournit le résultat souhaité), **efficace** (temps d'exécution, mémoire occupée), **clair** (compréhensible), **général** (traite le plus grand nombre de cas possibles), …
- **L'algorithmique** désigne aussi la discipline qui étudie les algorithmes et leurs applications en Informatique
- Une bonne connaissance de l'algorithmique permet d'écrire des algorithmes exacts et efficaces

3

Représentation d'un algorithme

Historiquement, deux façons pour représenter un algorithme:

- **L'Organigramme:** représentation graphique avec des symboles (carrés, losanges, etc.)
  - offre une vue d'ensemble de l'algorithme
  - représentation quasiment abandonnée aujourd'hui
- **Le pseudo-code:** représentation textuelle avec une série de conventions ressemblant à un langage de programmation (sans les problèmes de syntaxe)
  - plus pratique pour écrire un algorithme
  - représentation largement utilisée

4

Notions et instructions de base

Exemple D'un Algorithme

Variables A, B, C: **Entier**
Début
A ← 3
B ← 7
A ← B
B ← A+5
C ← A + B
C ← B – A
Fin

Notion de variable

- Dans les langages de programmation une **variable** sert à stocker la valeur d'une donnée
- Une variable désigne en fait un emplacement mémoire dont le contenu peut changer au cours d'un programme (d'où le nom variable)
- Règle : Les variables doivent être **déclarées** avant d'être utilisées, elle doivent être caractérisées par :
  - un nom (**Identificateur**)
  - un **type** (entier, réel, caractère, chaîne de caractères, …)

7

Choix des identificateurs (1)

Le choix des noms de variables est soumis à quelques règles qui varient selon le langage, mais en général:

- Un nom doit commencer par une lettre alphabétique
  exemple valide: A1 **exemple invalide: 1A**
- doit être constitué uniquement de lettres, de chiffres et du soulignement _ (Eviter les caractères de ponctuation et les espaces) **valides: SMIP2007, SMP_2007** **invalides: SMP 2005,SMI-2007,SMP;2007**
- doit être différent des mots réservés du langage (par exemple en Java: **int, float, else, switch, case, default, for, main, return**, …)
- La longueur du nom doit être inférieure à la taille maximale spécifiée par le langage utilisé

8

Choix des identificateurs (2)

Conseil: pour la lisibilité du code choisir des noms significatifs qui décrivent les données manipulées

exemples: TotalVentes2004, Prix_TTC, Prix_HT

Remarque: en pseudo-code algorithmique, on va respecter les règles citées, même si on est libre dans la syntaxe

9

Types des variables

Le type d'une variable détermine l'ensemble des valeurs qu'elle peut prendre, les types offerts par la plus part des langages sont:

- Type numérique
  - **Entier**
  - **Réel**
- Type logique ou booléen: deux valeurs VRAI ou FAUX
- Type caractère: lettres majuscules, minuscules, chiffres, symboles, ...

  exemples: 'A', 'a', '1', '?', ...
- Type chaîne de caractère: toute suite de caractères,

  exemples: " Nom, Prénom", "code postale: 1000", ...

10

Déclaration des variables

- Rappel: toute variable utilisée dans un programme doit avoir fait l'objet d'une déclaration préalable
- En pseudo-code, on va adopter la forme suivante pour la déclaration de variables

  Variables liste d'identificateurs : type
- Exemple:

```
Variables   i, j,k : entier
            x, y : réel
            OK: booléen
            ch1, ch2 : chaîne de caractères
```

- Remarque: pour le type numérique on va se limiter aux entiers et réels sans considérer les sous types

11

L'instruction d'affectation

- **l'affectation** consiste à attribuer une valeur à une variable (ça consiste en fait à remplir où à modifier le contenu d'une zone mémoire)
- En pseudo-code, l'affectation se note avec le signe ←

  Var← e: attribue la valeur de e à la variable Var
  - e peut être une valeur, une autre variable ou une expression
  - Var et e doivent être de même type ou de types compatibles
  - l'affectation ne modifie que ce qui est à gauche de la flèche
- **Ex valides:** i ←1 j ←i k ←i+j
  x ←10.3 OK ←FAUX ch1 ←"SMI"
  ch2 ←ch1 x ←4 x ←j

  (voir la déclaration des variables dans le transparent précédent)
- **non valides:** i ←10.3 OK ←"SMI" j ←x

12

Quelques remarques

- Beaucoup de langages de programmation (C/C++, Java, ...) utilisent le signe égal = pour l'affectation ←. Attention aux confusions:
  - l'affectation n'est pas commutative : A=B est différente de B=A
  - l'affectation est différente d'une équation mathématique :
    - A=A+1 a un sens en langages de programmation
    - A+1=2 n'est pas possible en langages de programmation et n'est pas équivalente à A=1
- Certains langages donnent des valeurs par défaut aux variables déclarées. Pour éviter tout problème il est préférable **d'initialiser les variables** déclarées

13

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 :
  - des opérateurs arithmétiques: **+, -, *, /, % (modulo), ^ (puissance)**
  - des opérateurs logiques: **NON, OU, ET**
  - des opérateurs relationnels: **= (égalité) ,≠ , <, >, <=, >=**
  - des opérateurs sur les chaînes: **&** (concaténation)
- Une expression est évaluée de gauche à droite mais en tenant compte de **priorités**

14

Priorité des opérateurs

- Pour les opérateurs arithmétiques donnés ci-dessus, l'ordre de priorité est le suivant (du plus prioritaire au moins prioritaire) :
  - ^ : (élévation à la puissance)
  - * , / (multiplication, division)
  - % (modulo)
  - + , - (addition, soustraction)

  exemple: **2 + 3 * 7 vaut 23**
- En cas de besoin (ou de doute), on utilise les parenthèses pour indiquer les opérations à effectuer en priorité

  exemple: **(2 + 3) * 7 vaut 35**

15

Exercice 1

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

Variables A, B : **Entier**
Début
A ← 1
B ← A+2
A ← 3
Fin

16

Exercice 2

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

Variables A, B, C: **Entier**
Début
A ← 3
B ← 7
A ← B
B ← A+5
C ← A + B
C ← B – A
Fin

17

Exercice 3

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

Variables A, B, C: **Entier**
Début
A ← 3
B ← A + 4
A ← A + 1
B ← A - 4
Fin

18

Exercice 4

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

Variables A, B : **Entier**
Début
A ← 1
B ← 2
A ← B
B ← A
Fin

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

19

Exercice 5

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

20

Exercice 6

Que produit l'algorithme suivant ?

Variables A, B,C : **Chaine de caractères**
Début
A ← ''10 ''
B ← ''2''
C ← B + A
Fin

L' instruction d'Ecriture

- **L'écriture** permet d'afficher des résultats à l'écran (ou de les écrire dans un fichier)
  - En pseudo-code, on note: **écrire x**
    la machine affiche le contenu de la zone mémoire x
  - Conseil: Avant de lire une variable, il est fortement conseillé d'écrire des messages à l'écran, afin de prévenir l'utilisateur de ce qu'il doit frapper

L' instruction de Lecture

- Les instructions de lecture et d'écriture permettent à la machine de communiquer avec l'utilisateur
- La **lecture** permet d'entrer des donnés à partir du clavier
  - En pseudo-code, on note: **lire x**

    la machine met la valeur entrée au clavier dans la zone mémoire nommée x
  - Remarque: Le programme s'arrête lorsqu'il rencontre une instruction Lire et ne se poursuit qu'après la frappe d'une valeur au clavier et de la touche Entrée

Exemple (lecture et écriture)

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

```
Algorithme Calcul_double
  variables A, B : entier
  Début
      écrire("entrer le nombre ")
      lire  A
      B ← 2*A
      écrire("le double de ", A, "est :", B)
  Fin
```

Exercice 7

Ecrire un algorithme qui permet de calculer la moyenne générale d'un étudiant sachant qu'il a passé un examen dans deux matières à coefficient différent.

Exercice 8

Ecrire un algorithme qui demande à l'utilisateur le prix hors taxe d'un article et le taux de TVA , puis calcule et affiche le prix TTC.

Exercice 9

Ecrire un algorithme qui demande à l'utilisateur le prix hors taxe d'un article, le taux de TVA et le taux de remise , puis calcule et affiche le prix TTC.

Les Structures conditionnelles

Les instructions conditionnelles (1)

- Les instructions conditionnelles servent à n'exécuter une instruction ou une séquence d'instructions que si une condition est vérifiée
- On utilisera la forme suivante: **Si** condition **alors**

  instruction ou suite d'instructions1

  Sinon

  instruction ou suite d'instructions2

  Finsi
  - la condition ne peut être que vraie ou fausse
  - si la condition est vraie, se sont les instructions1 qui seront exécutées
  - si la condition est fausse, se sont les instructions2 qui seront exécutées
  - la condition peut être une condition simple ou une condition composée de plusieurs conditions

29

Tests: instructions conditionnelles (2)

- La partie Sinon n'est pas obligatoire, quand elle n'existe pas et que la condition est fausse, aucun traitement n'est réalisé
  - On utilisera dans ce cas la forme simplifiée suivante:

    Si condition **alors**

    instruction ou suite d'instructions1

    Finsi

30

Exemple (Si…Alors…Sinon)

```
Algorithme AffichageValeurAbsolue (version1)
Variable x : réel
Début
    Ecrire (" Entrez un réel : “)
    Lire (x)
    Si (x < 0) alors
            Ecrire ("la valeur est négative ")
    Sinon
            Ecrire ("la valeur  est positive  ")
    Finsi
Fin
```


Conditions composées

- Une condition composée est une condition formée de plusieurs conditions simples reliées par des opérateurs logiques:
  ET, OU, OU exclusif (XOR) et NON
- Exemples :
  - x compris entre 2 et 6 : (x > 2) ET (x < 6)
  - n supérieur à 3 ou inférieur à 0 : (n>3) OU (n<0)
- L'évaluation d'une condition composée se fait selon des règles présentées généralement dans ce qu'on appelle tables de vérité

Tables de vérité

C1	C2	**C1 ET C2**
VRAI	VRAI	**VRAI**
VRAI	FAUX	**FAUX**
FAUX	VRAI	**FAUX**
FAUX	FAUX	**FAUX**

C1	C2	**C1 OU C2**
VRAI	VRAI	**VRAI**
VRAI	FAUX	**VRAI**
FAUX	VRAI	**VRAI**
FAUX	FAUX	**FAUX**

C1	C2	**C1 XOR C2**
VRAI	VRAI	**FAUX**
VRAI	FAUX	**VRAI**
FAUX	VRAI	**VRAI**
FAUX	FAUX	**FAUX**

C1	**NON C1**
VRAI	**FAUX**
FAUX	**VRAI**

33

Tests imbriqués

- Les tests peuvent avoir un degré quelconque d'imbrications

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

34

Tests imbriqués: exemple (version 1)

```
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
```


Tests imbriqués: exemple (version 2)

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

Remarque : dans la version 2 on fait trois tests systématiquement alors que dans la version 1, si le nombre est négatif on ne fait qu'un seul test

Conseil : utiliser les tests imbriqués pour limiter le nombre de tests et placer d'abord les conditions les plus probables

La structure *Selon*

- Sa syntaxe est la suivante :

```
Selon variable
   cas  valeur1 :  Action 1
   cas  Valeur2 :  Action2
                .
                .
                .
    cas Valeur n  : Action n
   Sinon  :        Action o
Fin Selon
```


Exercice 10

Ecrire un algorithme qui demande un nombre entier à l'utilisateur, puis qui affiche s'il est pair ou impair

Exercice 11

- Ecrire un algorithme qui, selon que la moyenne est :
  Entre 10 et 12, affiche Passable ;
  Entre 12 et 14, affiche Assez bien ;
  Entre 14 et 16, affiche Bien ;
  Supérieure ou égale à 16, affiche T. Bien.

Exercice 12

- Ecrire un algorithme qui permet de résoudre l'équation à premier degré :
  Ax+B =0

Exercice 13

- Ecrire un algorithme qui permet de résoudre l'équation : $Ax^2+Bx+C=0$

Exercice 14

Le prix de photocopies dans une reprographie varie selon le nombre demandé: 0,5 DH la copie pour un nombre de copies inférieur à 10, 0,4DH pour un nombre compris entre 10 et 20 et 0,3DH au-delà.

Ecrivez un algorithme qui demande à l'utilisateur le nombre de photocopies effectuées, qui calcule et affiche le prix à payer

Exercice 15

- Ecrire un algorithme qui demande à l'utilisateur d'entrer un nombre entre 1 et 7 et qui affiche le jour correspondant.

Les Structures Répétitives

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

(Dans ce cours, on va s'intéresser essentiellement aux boucles *Tant que* et boucles *Pour* qui sont plus utilisées et qui sont définies en Maple)

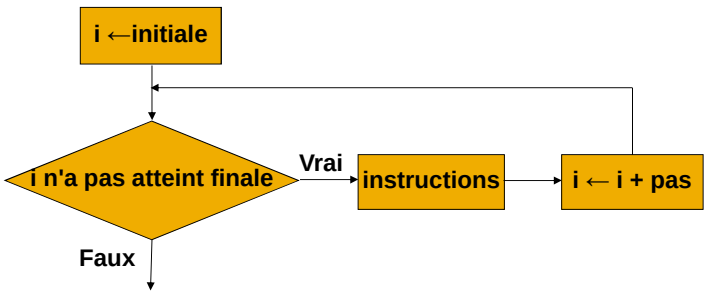
45

Les boucles Pour

Pour compteur **allant de** initiale **à** finale **pas** valeur du pas

instructions

FinPour



46

Les boucles Pour

- Remarque : le nombre d'itérations dans une boucle Pour est connu avant le début de la boucle
- **Compteur** est une variable de type entier (ou caractère). Elle doit être déclarée
- **Pas** est un entier qui peut être positif ou négatif. **Pas** peut ne pas être mentionné, car par défaut sa valeur est égal à 1. Dans ce cas, le nombre d'itérations est égal à finale - initiale+ 1
- **Initiale et finale** peuvent être des valeurs, des variables définies avant le début de la boucle ou des expressions de même type que compteur

Boucle Pour : exemple

```
Variables  i : entier
Debut

    Pour i allant de 1 à 10
            Ecrire (" La valeur de i est :  ", i)
    FinPour

Fin
```

Boucle Pour : remarque

- Il faut éviter de modifier la valeur du compteur (et de finale) à l'intérieur de la boucle. En effet, une telle action :
  - perturbe le nombre d'itérations prévu par la boucle Pour
  - rend difficile la lecture de l'algorithme
  - présente le risque d'aboutir à une boucle infinie

Exemple : **Pour** i **allant de** 1 **à** 5
i ← i -1
écrire(" i = ", i)
Finpour

Exercice 16

- Affichez les nombres pairs entre 0 et 50.
- Affichez la table de multiplication d'un nombre entré par l'utilisateur

Exercice 17

- Ecrire un algorithme qui affiche la moyenne générale d'une classe de 30 étudiants, sachant que chaque étudiant a passé un examen dans 2 matières différentes.

Exercice 18

- Calculer et afficher les sommes suivantes :

- S= 1+2+3……+100
- S=1+2+4+8+16+….+ 128
- S=1+ ½ + 3 + ¼ + 1/5 + 1/6 + 1/7+ ….. + 1/20
- S=(1*2)+(2*3)+(3*4)+…..+(19*20)
- S=(1*2)+(3*4)+(5*6)+…..+(19*20)

Exercice 19

- Calculer et afficher le factoriel d'un nombre entré par l'utilisateur.
- Calculer et Afficher les sommes suivantes :

$S = 1! + 2! + \ldots + n!$

$S = 1 + x/1! + x^2/2! + \ldots + x^n/n!$

$S = 1 + x^3/3! + x^5/5! + \ldots + x^{2n+1}/(2n+1)!$

$S = 1 - x + x^2 - x^3 + x^4 - x^5 \ldots + x^{2n}$

Exercice 20

- Ecrire un algorithme qui demande successivement 20 nombres à l'utilisateur et qui affiche :
  - La valeur maximale ainsi que sa position
  - Le nombre de valeurs positives
  - La moyenne des valeurs négatives

Exercice 21

- Ecrire un algorithme qui demande un nombre à l'utilisateur et qui indique si ce nombre est premier ou non.

Boucles imbriquées

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

- **Exemple:** Que produit l'algorithme suivant ?

```
Pour i allant de 1 à 4
        Pour j allant de 1 à 3
                écrire(i,j)
        FinPour
FinPour
```

Exécution

Exercice 22

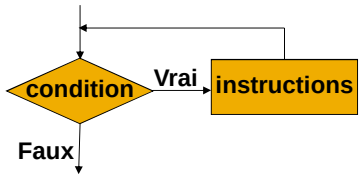
- Une classe de **N** élèves a passé un examen dans **P** matières différentes à coefficients différents.
- Calculer et afficher :
  - La moyenne de chaque élève.
  - La moyenne de la classe.

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 : remarques

- Le nombre d'itérations dans une boucle TantQue n'est pas connu au moment d'entrée dans la boucle. Il dépend de l'évolution de la valeur de condition
- Une des instructions du corps de la boucle doit absolument changer la valeur de condition de vrai à faux (après un certain nombre d'itérations), sinon le programme tourne indéfiniment

  ⟹ **Attention aux boucles infinies**
- Exemple de boucle infinie :

```
i ← 2
TantQue  (i > 0)
i ← i+1     (attention aux erreurs de frappe : + au lieu de -)
FinTantQue
```


Boucle Tant que : exemple

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

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

Lien entre Pour et TantQue

La boucle Pour est un cas particulier de Tant Que (cas où le nombre d'itérations est connu et fixé) . Tout ce qu'on peut écrire avec Pour peut être remplacé avec TantQue (la réciproque est fausse)

Pour compteur **allant de** initiale **à** finale **pas** valeur du pas

instructions

FinPour

peut être remplacé par :

(cas d'un pas positif)

compteur ← initiale
TantQue compteur <= finale
instructions
compteur ← compteur+pas
FinTantQue

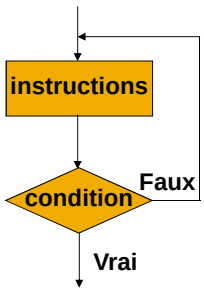
61

La boucle Répéter … jusqu'à …

Répéter

instructions

Jusqu'à condition



- 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 condition soit vrai (tant qu'elle est fausse)

62

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*
- Pour le choix entre *TantQue* et *jusqu'à :*
  - Si on doit tester la condition de contrôle avant de commencer les instructions de la boucle, on utilisera *TantQue*
  - Si la valeur de la condition de contrôle dépend d'une première exécution des instructions de la boucle, on utilisera *répéter jusqu'à*

Exercice 23

- Ecrire un algorithme qui demande à l'utilisateur un nombre compris entre 10 et 20 jusqu'à ce que la réponse convienne.
- En Cas de réponse supérieur à 20 on fera apparaître le message « Il faut un nombre plus petit » , et in versement « il faut un nombre plus grand » si le nombre est inférieur à 10

Exercice 24

- Un poissonnier sert un client qui a demandé 1Kg de poisson. Il pèse successivement différents poissons et s'arrête dès que le poids total égale ou dépasse 1Kg. Donner le nombre de poissons servis.

Exercice 25

- Ecrire un algorithme qui demande des nombres à l'utilisateur et qui calcule et affiche la moyenne des nombres impairs. La saisie s' arrête lorsque l'utilisateur entre le caractère « N » ou « n ».

Exercice 26

- Lire un code bancaire d'une carte guichet. Contrôlez ce code sachant qu'au bout de 3 tentatives non réussies la carte est retirer.

Les Tableaux

Exemple introductif

- Supposons qu'on veut conserver les notes d'une classe de 30 étudiants pour extraire quelques informations. Par exemple : calcul du nombre d'étudiants ayant une note supérieure à 10
- Le seul moyen dont nous disposons actuellement consiste à déclarer 30 variables, par exemple **N1, …, N30**. Après 30 instructions lire, on doit écrire 30 instructions Si pour faire le calcul

  nbre ← 0
  Si (N1 >10) alors nbre ←nbre+1 FinSi
  ….
  Si (N30>10) alors nbre ←nbre+1 FinSi

  c'est lourd à écrire
- Heureusement, les langages de programmation offrent la possibilité de rassembler toutes ces variables dans **une seule structure de donnée** appelée **tableau**

Tableaux

- Un **tableau** est un ensemble d'éléments de même type désignés par un identificateur unique
- Une variable entière nommée **indice** permet d'indiquer la position d'un élément donné au sein du tableau et de déterminer sa valeur
- La **déclaration** d'un tableau s'effectue en précisant le **type** de ses éléments et sa **dimension** (le nombre de ses éléments)
  - En pseudo code :

    Tableau identificateur**[dimension] : type**
  - Exemple :

    Tableau notes**[30] : réel**
- On peut définir des tableaux de tous types : tableaux d'entiers, de réels, de caractères, de booléens, de chaînes de caractères, …

Tableaux : remarques

- L'accès à un élément du tableau se fait au moyen de l'indice. Par exemple, **notes[i]** donne la valeur de l'élément i du tableau notes
- Selon les langages, le premier indice du tableau est soit 0, soit 1. Le plus souvent c'est 0 (c'est ce qu'on va adopter en pseudo-code). Dans ce cas, **notes[i]** désigne l'élément i+1 du tableau notes
- Il est possible de déclarer un tableau sans préciser au départ sa dimension. Cette précision est faite ultérieurement
  - Par exemple, quand on déclare un tableau comme paramètre d'une procédure, on peut ne préciser sa dimension qu'au moment de l'appel
  - En tous cas, un tableau est inutilisable tant qu'on n'a pas précisé le nombre de ses éléments
- Un grand avantage des tableaux est qu'on peut traiter les données qui y sont stockées de façon simple en utilisant des boucles

71

Tableaux : exemples (1)

- Pour le calcul du nombre d'étudiants ayant une note supérieure à 10 avec les tableaux, on peut écrire :

```
Variables       i ,nbre : entier
Tableau         notes[30] : réel
Début
    nbre ← 0
    Pour i allant de 0 à 29
            Si (notes[i] >10) alors
                nbre ←nbre+1
            FinSi
    FinPour
    écrire ("le nombre de notes supérieures à 10 est : ", nbre)
Fin
```

72

Tableaux : saisie et affichage

- Algorithmes qui permettent de saisir et d'afficher les éléments d'un tableau :

```
Pour i allant de 0 à n-1
    écrire ("Saisie de l'élément ", i + 1)
    lire (T[i] )
FinPour
```

```
Pour i allant de 0 à n-1
    écrire ("T[",i, "] =", T[i])
FinPour
```


Exercice 27

- Ecrire un algorithme qui permet de lire 100 notes et qui détermine le nombre de celle supérieur à la moyenne.

Exercice 28

- Ecrire un algorithme qui demande à l'utilisateur de remplir un tableau de N nombres et qui affiche la valeur maximale du tableau ainsi que sa position.

Exercice 29

- Soit un Tableau T de N éléments. Ecrire un algorithme qui calcule le nombre d'occurrence d'une valeur X figurant sur le tableau.
  NB : X doit être entré par l'utilisateur

Exercice 30

- Soit un Tableau T de N éléments triés dans l'ordre croissant. Ecrire un algorithme qui permet d'insérer dans T une valeur X entrée par l'utilisateur.

Exercice 31

- Ecrire un algorithme qui permet de supprimer dans un tableau T une valeur X entrée par l'utilisateur.

Exercice 32 – Recherche séquentielle

- Ecrire un algorithme permettant de rechercher, dans un tableau T, une valeur X entrée par l'utilisateur.

Recherche dichotomique

- Dans le cas où le tableau est ordonné, on peut améliorer l'efficacité de la recherche en utilisant la méthode de recherche dichotomique
- **Principe :** diviser par 2 le nombre d'éléments dans lesquels on cherche la valeur x à chaque étape de la recherche. Pour cela on compare x avec T[milieu] :
  - Si x < T[milieu], il suffit de chercher x dans la 1ère moitié du tableau entre (T[0] et T[milieu-1])
  - Si x > T[milieu], il suffit de chercher x dans la 2ème moitié du tableau entre (T[milieu+1] et T[N-1])

Exemple d'exécution

- Considérons le tableau T :

4	6	10	15	17	18	24	27	30

- Si la valeur cherché est 20 alors les indices inf, sup et milieu vont évoluer comme suit :

inf	0	5	5	6
sup	8	8	5	5
milieu	4	6	5	

- Si la valeur cherché est 10 alors les indices inf, sup et milieu vont évoluer comme suit :

inf	0	0	2
sup	8	3	3
milieu	4	1	2

Exercice 33 - Recherche dichotomique

- Ecrire un algorithme permettant de rechercher, dans un tableau trié T , une valeur X entrée par l'utilisateur (En utilisant la recherche dichotomique)

Tri d'un tableau

- Le tri consiste à ordonner les éléments du tableau dans l'ordre croissant ou décroissant
- Il existe plusieurs algorithmes connus pour trier les éléments d'un tableau :
  - Le tri par sélection
  - Le tri par insertion
  - Le tri rapide
  - Tri à bulle
  - ...
- Nous verrons dans la suite l'algorithme de tri par sélection. Le tri sera effectué dans l'ordre croissant

Tri par sélection

- **Principe** : à l'étape i, on sélectionne le plus petit élément parmi les (n - i +1) éléments du tableau les plus à droite. On l'échange ensuite avec l'élément i du tableau
- **Exemple :**

9	4	1	7	3

  - **Étape 1:** on cherche le plus petit parmi les 5 éléments du tableau. On l'identifie en troisième position, et on l'échange alors avec l'élément 1 :

1	4	9	7	3

  - **Étape 2:** on cherche le plus petit élément, mais cette fois à partir du deuxième élément. On le trouve en dernière position, on l'échange avec le deuxième:

1	3	9	7	4

  - **Étape 3:**

1	3	4	7	9

Exercice 35- Tri par sélection

- Soit un tableau T de taille N. Triez ce tableau en utilisant le tri par sélection

Tri à Bulle

- Le **tri à bulles** ou **tri par propagation** est un algorithme de tri qui consiste à faire remonter progressivement les plus grands éléments d'un tableau.

Tri à Bulle

- **Première étape:**
  (**5 1** 4 2 8) (**1 5** 4 2 8) Les éléments 5 et 1 sont comparés, et comme 5 > 1, l'algorithme les intervertit.
  (1 **5 4** 2 8) (1 **4 5** 2 8) Interversion car 5 > 4.
  (1 4 **5 2** 8) (1 4 **2 5** 8) Interversion car 5 > 2.
  (1 4 2 **5 8**) (1 4 2 **5 8**) Comme 5 < 8, les éléments ne sont pas échangés.
- **Deuxième étape:**
  (**1 4** 2 5 8) (**1 4** 2 5 8) Même principe qu'à l'étape 1.
  (1 **4 2** 5 8) (1 **2 4** 5 8)
  (1 2 **4 5** 8) (1 2 **4 5** 8)
  À ce stade, la liste est triée, mais pour le détecter, l'algorithme doit effectuer un dernier parcours.
- **Troisième étape:**
  (**1 2** 4 5 8) (**1 2** 4 5 8)
  (1 **2 4** 5 8) (1 **2 4** 5 8)
  Comme la liste est triée, aucune interversion n'a lieu à cette étape, ce qui provoque l'arrêt de l'algorithme.

Exercice 36- Tri par à bulle

- Soit un tableau T de taille N. Triez ce tableau en utilisant le tri à bulle.

Tableaux à deux dimensions (les matrices)

- Les langages de programmation permettent de déclarer des tableaux dans lesquels les valeurs sont repérées par **deux indices**. Ceci est utile par exemple pour représenter des matrices
- En pseudo code, un tableau à deux dimensions se **déclare** ainsi :

  tableau identificateur**[dimension1] [dimension2] : type**

  - Exemple : une matrice A de 3 lignes et 4 colonnes dont les éléments sont réels

    tableau A[3][4] **: réel**
- **A[i][j]** permet d'accéder à l'élément de la matrice qui se trouve à l'intersection de la ligne i et de la colonne j

Exemples : lecture d'une matrice

- Algorithme qui permet de saisir les éléments d'une matrice :

```
variables i,j : entier
Début
  Pour i allant de 0 à n-1
      écrire ("saisie de la ligne ", i + 1)
      Pour j allant de 0 à m-1
         écrire ("Entrez l'élément de la ligne ", i + 1, " et de la colonne ", j+1)
         lire (A[i][j])
      FinPour
  FinPour
Fin
```

Exemples : affichage d'une matrice

- Algorithme qui permet d'afficher les éléments d'une matrice :

```
    variables i,j : entier
Début
    Pour i allant de 0 à n-1
        Pour j allant de 0 à m-1
            écrire ("A[",i, "] [",j,"]=", A[i][j])
        FinPour
    FinPour
Fin
```


Exemples : somme de deux matrices

- Procédure qui calcule la somme de deux matrices :

```
Procédure SommeMatrices(n, m : entier par valeur,
        tableau A, B : réel par valeur , tableau C : réel par référence )
Début
 variables i,j : entier
  Pour i allant de 0 à n-1
      Pour j allant de 0 à m-1
          C[i][j] ← A[i][j]+B[i][j]
      FinPour
  FinPour
Fin Procédure
```

Exercice 37

- Ecrire un algorithme qui permet de calculer :
  - La moyenne de toutes les cases d'une matrice de L lignes et C colonnes.
  - La Somme de chaque ligne
  - La moyenne et le maximum de chaque colonne
  - La moyenne de la diagonale principale
  - La somme des éléments de la partie supérieure par rapport à la diagonale

Les chaines de caractères

Fonctions des chaines de caractères

- La grande majorité des langages proposent les fonctions suivantes de manipulation des chaînes de caractères, même si le nom et la syntaxe peuvent varier d'un langage à l'autre:
  - **Longueur(chaîne)** : renvoie le nombre de caractères d'une chaîne
  - **Extraire(chaîne, n1,n2)** : renvoie un extrait de la chaîne, commençant au caractère n1 et faisant n2 caractères de long.
  - **Position(chaîne, caractère)** : renvoie la position du caractère dans la chaine.
  - **Asc (caractère)** renvoie le code ASCII du caractère
  - **Chr (valeur)** renvoie le caractère dont code ASCII est égal à valeur.

Exercice 38

- Ecrire un algorithme qui demande à l'utilisateur d'entrer une chaine de caractères et qui calcule et affiche le nombre de « A » se trouvant dans cette chaine.
- Ecrire un algorithme qui affiche le nombre de consonnes dans une chaîne entrée par l'utilisateur.
- Ecrire un algorithme qui permet de supprimer les voyelles d'une chaines entrée par l'utilisateur.
- Ecrire un algorithme qui permet de comparer deux chaines de caractères (caractère par caractère) .

Exercice 38 (suite)

- Ecrire un algorithme qui demande une chaine de caractères de longueur impaire et l'affiche sous la forme d'un triangle.
- Exp : bonjour
- onjou
- njo
- j

Fonctions et procédures

Fonctions et procédures

- Certains problèmes conduisent à des programmes longs, difficiles à écrire et à comprendre. On les découpe en des parties appelées **sous-programmes** ou **modules**
- Les **fonctions** et les **procédures** sont des modules (groupe d'instructions) indépendants désignés par un nom. Elles ont plusieurs **intérêts** :
  - permettent de **"factoriser" les programmes,** càd de mettre en commun les parties qui se répètent
  - permettent une **structuration** et une **meilleure lisibilité** des programmes
  - **facilitent la maintenance** du code (il suffit de modifier une seule fois)
  - ces procédures et fonctions peuvent éventuellement être **réutilisées** dans d'autres programmes

Fonctions

- Le **rôle** d'une fonction en programmation est similaire à celui d'une fonction en mathématique : elle **retourne un résultat à partir des valeurs des paramètres**
- Une fonction s'écrit en dehors du programme principal sous la forme :

```
Fonction nom_fonction (paramètres et leurs types) : type_retour
    Instructions constituant le corps de la fonction
    retourne ...
FinFonction
```

- Pour le choix d'un nom de fonction il faut respecter les mêmes règles que celles pour les noms de variables
- **type_retour** est le type du résultat retourné
- L'instruction **retourne** sert à retourner la valeur du résultat

Fonctions : exemple

- La fonction SommeCarre suivante calcule la somme des carrées de deux réels x et y :

```
Fonction SommeCarre (x : réel, y: réel ) : réel
        variable z : réel
        z ←x^2+y^2
        retourne (z)
FinFonction
```


Utilisation des fonctions

- L'utilisation d'une fonction se fera par simple écriture de son nom dans le programme principale. Le résultat étant une valeur, devra être affecté ou être utilisé dans une expression, une écriture, ...
- **Exepmle :**

```
Algorithme exepmleAppelFonction
  variables   z : réel
Début
   z ←5*SommeCarre(7,2)+1
   écrire("SommeCarre(3,5)= ", SommeCarre(3,5))
Fin
```

Procèdures

- Dans certains cas, on peut avoir besoin de répéter une tache dans plusieurs endroits du programme, mais que dans cette tache on ne calcule pas de résultats ou qu'on calcule plusieurs résultats à la fois
- Dans ces cas on ne peut pas utiliser une fonction, on utilise une **procédure**
- Une **procédure** est un sous-programme semblable à une fonction mais qui **ne retourne rien**
- Une procédure s'écrit en dehors du programme principal sous la forme :

  Procédure nom_procédure (paramètres et leurs types)

  Instructions constituant le corps de la procédure

  FinProcédure
- Remarque : une procédure peut ne pas avoir de paramètres

Appel d'une procédure

- L'appel d'une procédure, se fait dans le programme principale ou dans une autre procédure par une instruction indiquant le nom de la procédure :

```
Procédure exemple_proc (...)
        ...
FinProcédure

Algorithme exepmleAppelProcédure
Début
        exemple_proc (...)
        ...
Fin
```

- Remarque : contrairement à l'appel d'une fonction, on ne peut pas affecter la procédure appelée ou l'utiliser dans une expression. L'appel d'une procédure est une instruction autonome

Paramètres d'une procédure

- Les paramètres servent à échanger des données entre le programme principale (ou la procédure appelante) et la procédure appelée
- Les paramètres placés dans la déclaration d'une procédure sont appelés **paramètres formels**. Ces paramètres peuvent prendre toutes les valeurs possibles mais ils sont abstraits (n'existent pas réellement)
- Les paramètres placés dans l'appel d'une procédure sont appelés **paramètres effectifs**. ils contiennent les valeurs pour effectuer le traitement
- Le nombre de paramètres effectifs doit être égal au nombre de paramètres formels. L'ordre et le type des paramètres doivent correspondre

Variables locales et globales (1)

- On peut manipuler 2 types de variables dans un module (procédure ou fonction) : des **variables locales** et des **variables globales**. Elles se distinguent par ce qu'on appelle leur **portée** (leur "champ de définition", leur "durée de vie")
- Une **variable locale** n'est connue qu'à l'intérieur du module ou elle a été définie. Elle est créée à l'appel du module et détruite à la fin de son exécution
- Une **variable globale** est connue par l'ensemble des modules et le programme principale. Elle est définie durant toute l'application et peut être utilisée et modifiée par les différents modules du programme

Variables locales et globales (2)

- La manière de distinguer la déclaration des variables locales et globales diffère selon le langage
  - En général, les variables déclarées à l'intérieur d'une fonction ou procédure sont considérées comme variables locales
- En pseudo-code, on va adopter cette règle pour les variables locales et on déclarera les variables globales dans le programme principale
- **Conseil :** Il faut utiliser autant que possible des variables locales plutôt que des variables globales. Ceci permet d'économiser la mémoire et d'assurer l'indépendance de la procédure ou de la fonction

Exercice 39

- Ecrire une fonction MAX qui calcule le maximum de deux valeurs
- Ecrire un algorithme qui permet de calculer le maximum de trois valeurs

Exercice 40

- Calculer et afficher la somme :
  S= a! + b! + c!
- Définir une procédure ECHANGER permettant d'échanger les valeurs de deux variables.
  Exploiter cette procédure pour inverser les éléments d'un tableau.

Récursivité

- Un module (fonction ou procédure) peut s'appeler lui-même: on dit que c'est un module **récursif**
- Tout module récursif doit posséder un cas limite (cas trivial) qui arrête la récursivité
- Exemple : Calcul du factorielle

```
Fonction fact (n : entier ) : entier
        Si (n=0) alors
                retourne (1)
        Sinon
                retourne (n*fact(n-1))
        Finsi
FinFonction
```

Exercice 41

- Ecrire des fonctions récursives permettant de calculer les sommes suivantes :
- $S= x^n$
- S= a+b
- $S=1^3+2^3+3^3....+n^3$

Exercice 42

- Ecrivez une fonction récursive (puis itérative) qui calcule le terme n de la suite de Fibonacci définie par :

$$U(0)=U(1)=1$$
$$U(n)=U(n-1)+U(n-2)$$

Les Structures

Définition

- Les structures servent à rassembler au saint d'une seule entité un ensemble fini d'éléments de type différent. La syntaxe de construction d'une structure est la suivante :

```
Structure   Nom_structure
            Attribut 1 :  Type
            Attribut 2 :  Type
                .
                .
                .
            Attribut  N :  Type
Fin Structure
```

Exemple

Structure Etudiant
Nom : Chaine
Note: Réel
Fin Structure
Variables e1, e2 : Etudiant
Début
Ecrire « Entrer le nom du premier étudiant **»**
Lire e1.Nom
e2.Note ← 18
Fin

Exercice 43

- Définir une structure ETUDIANT constituée des éléments suivants : Nom(chaine), Tnote(20) (un tableau de 20 notes), Moyenne(réel), Résultat (booléen).
- Ecrire un algorithme qui lit les informations de N étudiants (Nom et Notes) et qui calcule la moyenne de chaque étudiant ainsi que son résultat (« Admis » , « Non Admis ». Afficher ensuite le nom du majorant de la classe.

Exercice 44

- Définir une structure EMPLOYE constituée des éléments suivants : Nom(chaine), Salaire(réel) et DateNaissance(structure). La DateNaissance étant une structure constituer de : Jour , Mois et Année (des entiers)
- Ecrire un algorithme qui lit et affiche les informations d'un employé en calculant son âge.