L'algorithmique

Concepts fondamentales

Abdelali Saidi

saidi.a@ucd.ac.ma

Objectifs du cours

- Notions de base et Introduction à l'algorithmique
- Notion de sous-programmes et lien avec la compilation
- Structures algorithmiques fondamentales et élaboration des algorithmes
- Implantation des algorithmes dans un langage de programmation
- Introduction au test unitaire
- Utilisation des tableaux - Utilisation des chaînes de caractères
- Algorithmes fondamentaux de recherche d'un élément, parcours, tri, ...
- Avoir une première notion des performances des algorithmes utilisés

Conditions d'evaluation

- Deux contrôles continus par élément de module (50% chacun)
- Algorithmique : 30%
- Langage de programmation C : 40%
- Bureautique : 30%
- Un module est acquis par validation si sa note est supérieure ou égale à 12 sur 20 sans qu'aucune note des éléments le composant ne soit inférieure strictement à 6 sur 20.

Partie 1 : Introduction à l'algorithmique

1. Matériels et logiciels
2. Conception de logiciels
3. Le calcul d'un sphère
4. Composants d'un algorithme

Matériels et logiciels

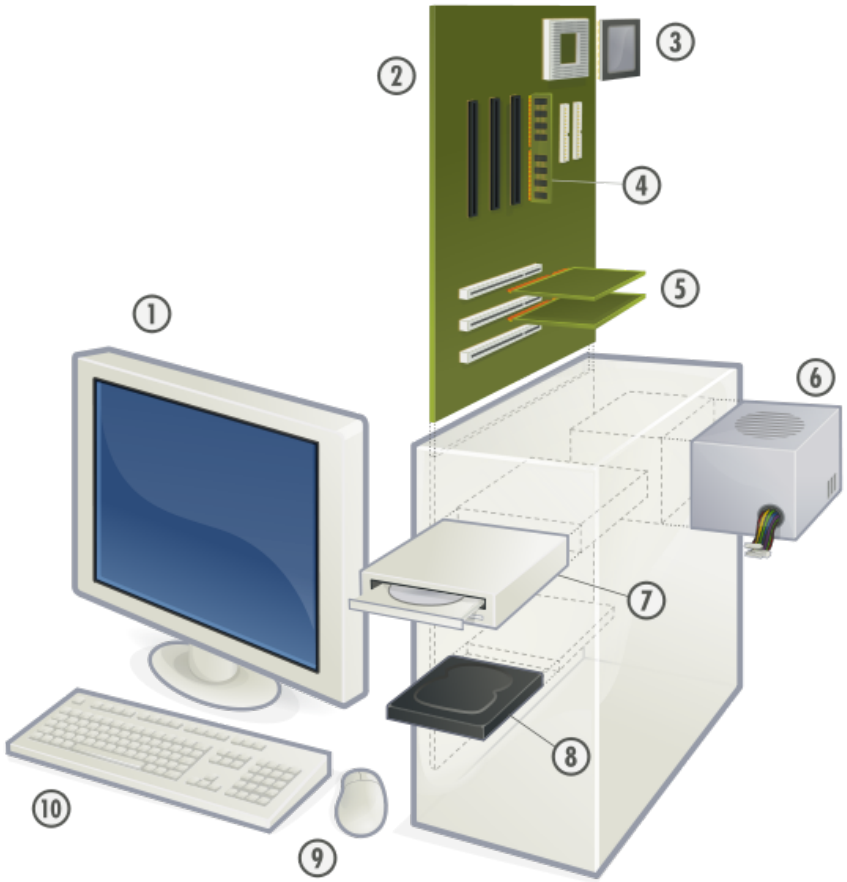


Figure: Composants matériel d'un ordinateur

Matériels et logiciels

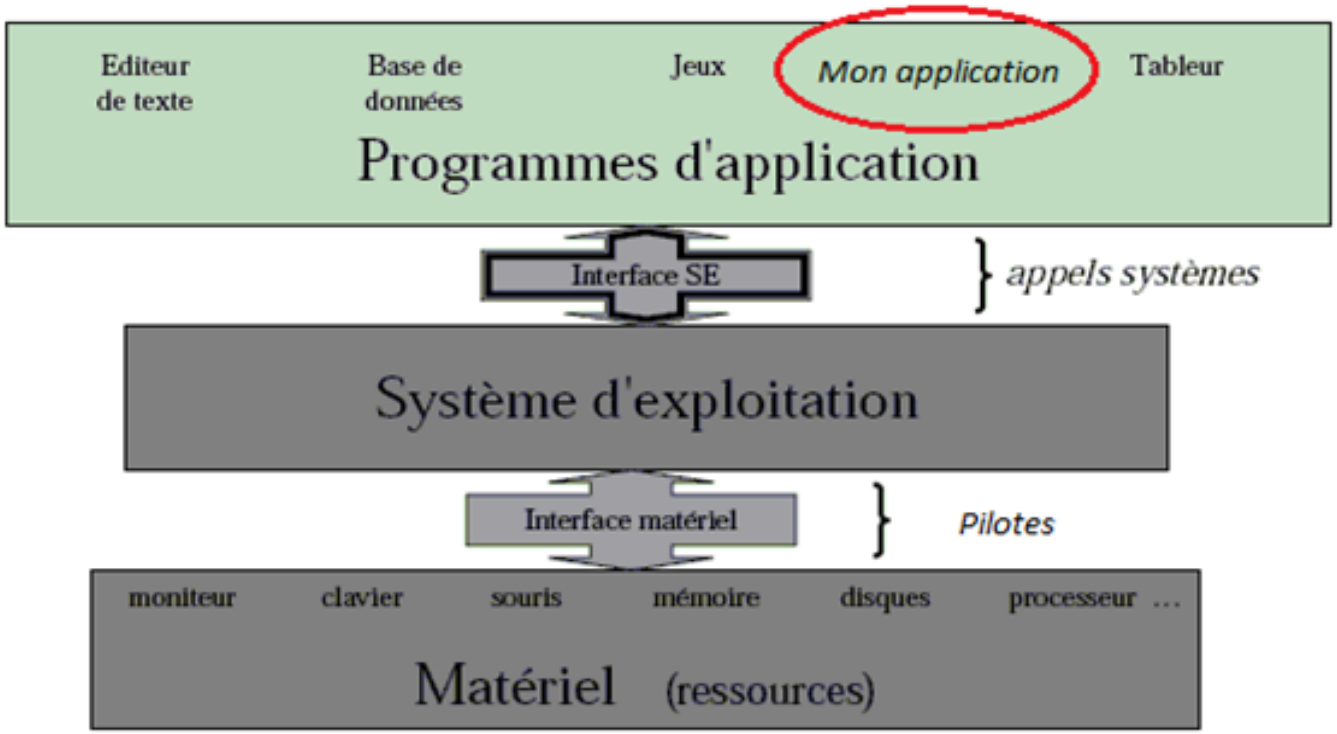


Figure: Situation du logiciel

Conception de logiciels

1. Résolution du problème
   - Étude des besoins (orientée clients)
   - Analyse (intrants, traitement, extrants, autres informations)
   - Conception de l'algorithme (architecture interne, interfaces)
2. Mise en oeuvre
   - Codification, programmation: traduction de l'algorithme
   - Jeu de tests, test unitaires, test systèmes
3. Exploitation
   - Déploiement (installation, documentation, formation)
   - Maintenance (mises à jour)

Définition

Origine

L'algorithmique est un terme d'origine arabe, hommage à Al Khawarizmi (780-850) auteur d'un ouvrage décrivant des méthodes de calculs algébriques.

Définition

Définition

En informatique, un algorithme est suite finie d'opérations élémentaires constituant un schéma de calcul ou de résolution d'un problème.

Exemples d'algorithmiques

Recette de cuisine

Nous voulons préparer une recette dans la cuisine, nous aurons besoin :

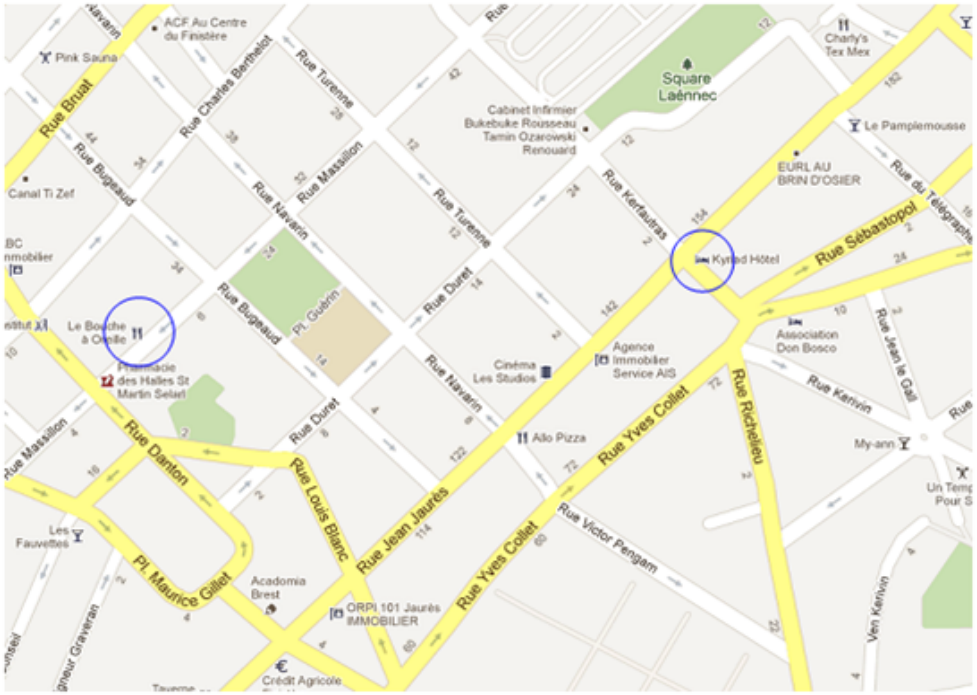
- Des ingrédients et du matériels (ces sont des entrées)
- D'une méthode de préparation ou d'exécution d'opérations
- D'un certain temps

Nous aurons finalement la recette prête :

Exemples d'algorithmiques

Indication routière

Méthode : Il s'agit ici de décrire une suite ordonnée d'instructions (aller tout droit, prenez la troisième à droite. . .) qui manipulent des données (carrefours, rues. . .) pour réaliser la tâche désirée (aller au restaurant).



Exemples d'opérations

Parmi les opérations que peut réaliser un algorithme :

- Lire un nombre du clavier
- Effectuer une opération mathématique
- Copier une valeur en mémoire
- Afficher une réponse à l'écran

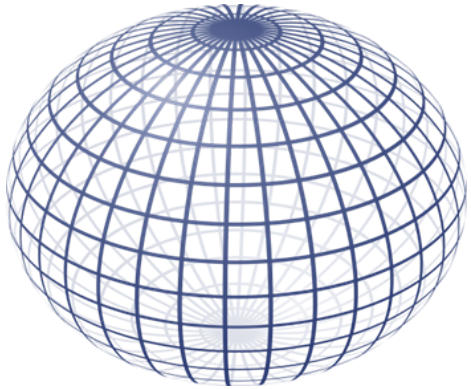
Le calcul d'un sphère

Problème :

Besoin d'un outil pour calculer le volume de sphères de dimension variable

Étapes de réalisation de cet outil de calcul :

- Analyse
- Conception
- Codification
- Mise au point



Le calcul d'un sphère

Analyse du problème

- Quelles sont les Entrée(s)?
  - Le rayon en mètre
  - Constante $\Pi = 3,14159265358979323 84$
- Quelles sont les Sortie(s)?
  - Le volume en mètre cube
- Quelles sont les Opérations ?
  - Demander le rayon à l'utilisateur
  - Appliquer la formule : $Volume = 4/3 \Pi Rayon^3$
  - Afficher le volume

Le calcul d'un sphère

Conception de l'algorithme

- $\Pi \leftarrow 3,14159$
- Lire Rayon (C'est l'ordinateur qui effectue cette lecture)
- Volume $\leftarrow 4/3 * \Pi * Rayon^3$
- Écrire Volume (C'est l'ordinateur qui effectue cette écriture)

Le calcul d'un sphère

Conception de l'algorithme

Lors de la conception d'un programme, sa qualité peut être discutée :

- Ex : Un logiciel rapide est couteux qu'un logiciel lent.

Il existe trois fondamentaux critères d'un bon algorithme :

- **Correct**: Il faut que le programme exécute correctement ses tâches pour lesquelles il a été conçu.
- **Complet**: Il faut que le programme considère tous les cas possibles et donne un bon résultat dans chaque cas.
- **Efficace**: Il faut que le programme exécute sa tâche avec efficacité de telle sorte qu'il se déroule en un temps minimal et qu'il consomme un minimum de ressources

Le calcul d'un sphère

Codification

Dans cette phase, l'algorithme est traduit dans un langage de programmation.

- Langage de programmation
- C, C++, Java
- Environnement de développement

Fichier résultant

- Fichiers sources
- Fichiers exécutables
- Documentation

Le calcul d'un sphère

Mise au point

Réalisation des tests pour s'assurer de la qualité du programme.

Entrée (Rayon)	**Sortie (Volume)**
1	4,1887
2	33,5103
0	Erreur
-1	Erreur

Composants d'un algorithme

Représentation de l'algorithme

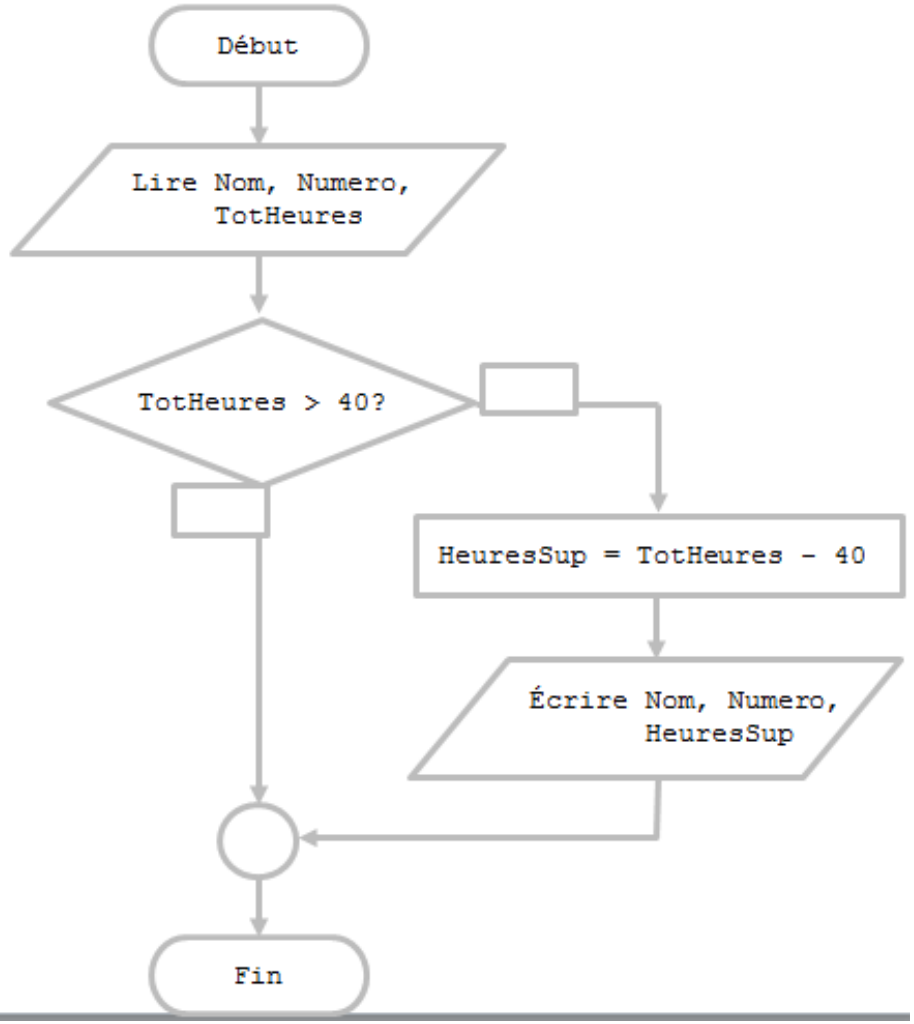
Historiquement, deux façons pour représenter un algorithme:

- L'organigramme [Morphogramme]: 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

Composants d'un algorithme

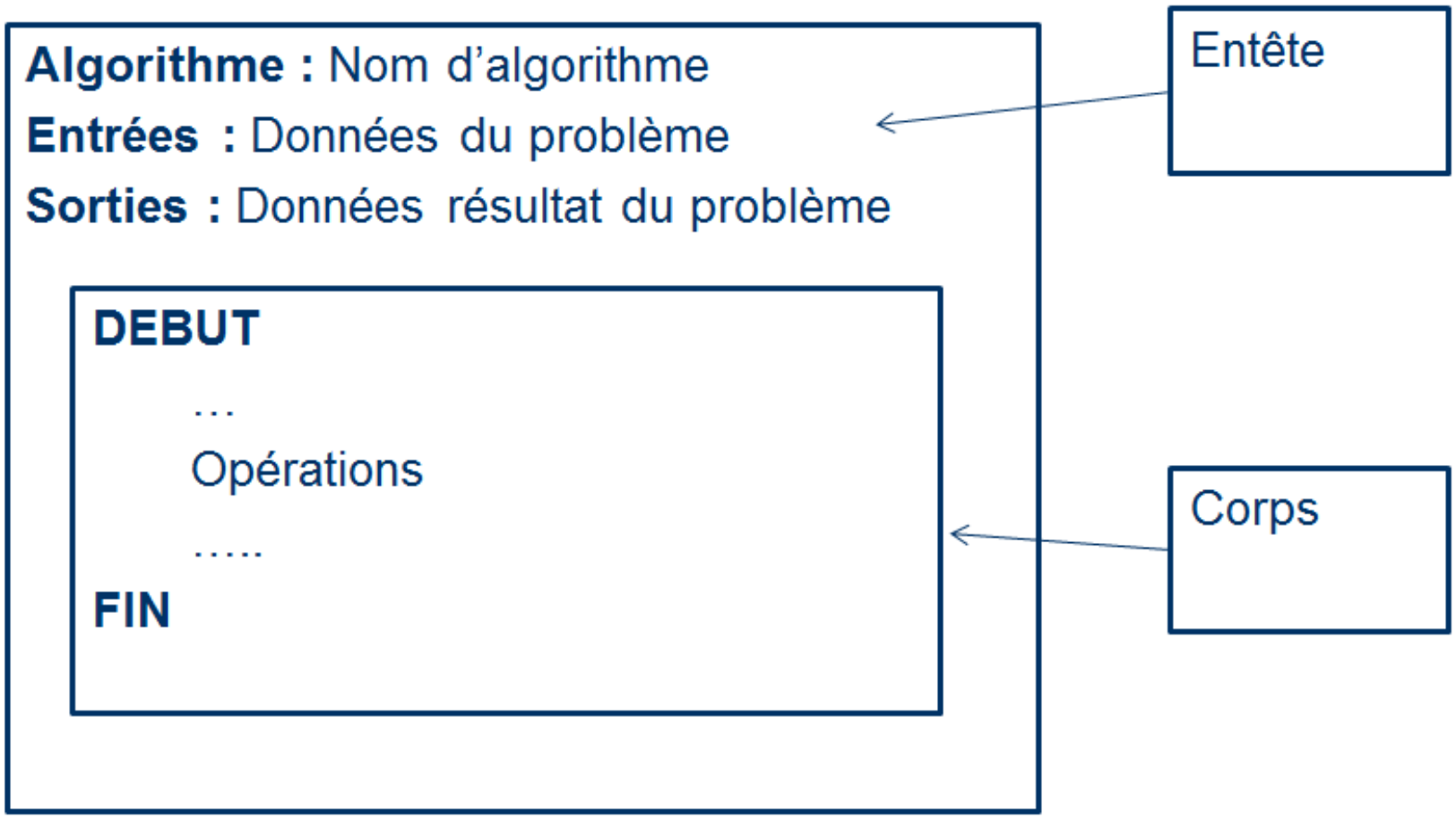
Exemple

```
Algorithme #test
LIRE Nom, Numero, TotHeures
SI TotHeures > 40 ALORS
    HeuresSup = TotHeures - 40
    ÉCRIRE Nom, Numero, HeuresSup
FINSI
```



Composants d'un algorithme

Structure générale



Langages de programmation

- Pourquoi un langage de programmation ?
  - L'ordinateur ne sait exécuter qu'un certain nombre d'opérations élémentaires.
  - L'ordinateur ne comprend que le langage machine.
  - Tout type d'ordinateur a son propre langage machine
- Nécessité d'un langage commun entre le programmeur et l'ordinateur
  - Utilisable sur n'importe quel ordinateur.
  - Destiné à formuler un algorithme.
  - Comporte une syntaxe et une sémantique.
  - Exemples : C, C++, Pascal, Delphi, Fortran, Java, PHP, ...

Langages de programmation

Quel que soit le langage de programmation, il est nécessaire de le traduire en langage machine

1. Compilation :
   - L'ensemble du programme (écrit en langage de programmation) est traduit en langage machine.
   - La suite d'instructions ainsi obtenue est exécutée par l'ordinateur.
2. Interprétation :
   - Les instructions (en langage de programmation) sont traduites l'une après l'autre en langage machine et exécutées directement.
   - Traduction et exécution à la volée.

Langages de programmation

Algorithme

- Une suite finie d'instructions élémentaires.
- Est écrit dans un langage universel 'pseudo-code'.
- Est indépendant du langage de programmation.
- Sert à résoudre un problème informatique.

Programme

- Vient en aval de l'algorithme.
- Est écrit dans un langage de programmation.
- Résultat de la traduction de l'algorithme.
- Est indépendant de l'architecture de l'ordinateur.

Partie 2 : Concepts de variable et affectation

5. Éléments de base
6. Notion de Variable
7. Notion de constante
8. Notion d'affectation

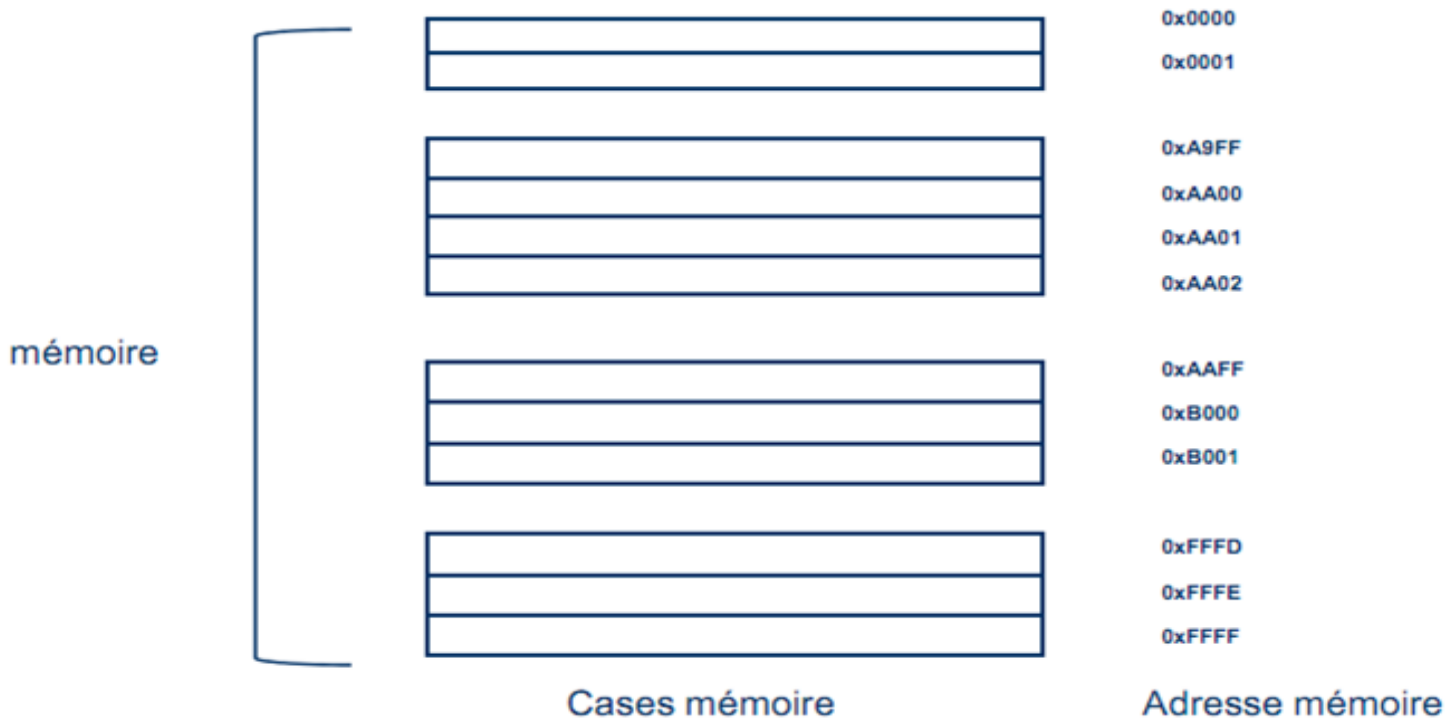
éléments de base

Un algorithme est formé de quatre types d'instructions considérées comme des petites briques de base :

1. l'affectation de variables
2. la lecture et/ou l'écriture
3. les tests
4. les boucles

Notion de Variable

- La mémoire vive stocke les logiciels en cours d'exécution
- La mémoire contient des cases mémoires stockant les données.



Notion de Variable

- Une variable désigne une case (emplacement) mémoire dont le contenu peut changer au cours d'exécution d'un algorithme/logiciel (d'où le nom de variable)
- Une variable est définit par :
  - un nom (i.e., Identificateur)
  - un type
  - une valeur

Notion de Variable

Règles de nomination

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

- Le nom doit commencer par une lettre alphabétique
  - Exemples : "E1" ("1E" n'est pas valide)
- Le nom doit être constitué uniquement de lettres, de chiffres et du soulignement (Éviter les caractères de ponctuation et les espaces)
  - Exemples corrects: "TEXTE2016" , "TEXTE_2016"
  - Exemples incorrects : "TEXTE 2016" , "TEXTE-2016" , "TEXTE;2016"
- Le nom doit être différent des mots réservés (par exemple en C: int, float, double, switch, case, for, main, return, ...)
- La longueur du nom doit être inférieure à la taille maximale spécifiée par le langage utilisé

Notion de Variable

Type

- Le type d'une variable détermine l'ensemble -intervalle des valeurs qu'elle peut prendre.
- Les types offerts par la plus part des langages sont :
  1. Type numérique (entier ou réel)
  2. Booléen
  3. Caractère
  4. Chaîne de caractère

Notion de Variable

Déclaration d'une variable

- Toute variable utilisée dans un programme doit avoir fait l'objet d'une déclaration préalable.
- En pseudo-code, la déclaration de variables est effectuée par la forme suivante :
  - Variables nomVariable1, nomVariable2 : type
- Exemple :

```
Variables   i, j, k : entier
            x, y : réel
            OK: booléen
            Ch1, ch2 : chaîne de caractères
```

Notion de constante

- Une constante est une variable dont la valeur ne change pas au cours de l'exécution du programme.
- Pour déclarer une constante, on utilise la forme suivante :
  - Constante NOM_CONSTANTE = valeur : type,. . .
- Par convention les noms de constantes sont en **majuscules**

Notion de constante

Exemple

Constante PI=3.14 : réel
MAX = 100 : entier
MIN = -123 : entier signé

Notion d'affectation

- L'affectation est une opération qui consiste à attribuer une valeur à une variable (c'est-à-dire remplir ou modifier le contenu d'une zone mémoire)
- En pseudo-code, l'affectation est notée par le signe $\leftarrow$
  - Var $\leftarrow$ e : attribue la valeur de e à la variable Var
- On dit "Var reçoit la valeur de e", "la valeur de e est affecté à Var"
- Caractéristiques :
  - 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

Notion d'affectation

Exemple 1

- Soient i, j, k : entier; x :réel; ok :booléen; ch1,ch2 :chaine de caractères
- Que doit contenir les variables à la fin d'éxecution ?

```
i ←1
j ←i
k ←i+j
x ←10.3
OK ←FAUX
ch1 ←  « EST"
ch2 ←ch1
x ←4
x ←j
```

Notion d'affectation

Exemple 2

- Soient i, j, k : entier; x : réel; ok : booléen; ch1,ch2 : chaine de caractères

$i \leftarrow 10.3$
$OK \leftarrow$ " $TEXTE$"
$j \leftarrow x$
$k \leftarrow ch1\&ch2$

- Ces exemples ci-dessus sont invalides

Notion d'affectation

Remarques

- Lors d'une affectation, la valeur de droite est affectée à la variable de gauche. Ainsi, $A \leftarrow B$ est différente de $B \leftarrow A$
- l'affectation est différente d'une équation mathématique :
  - Les opérations $x \leftarrow x + 1$ et $x \leftarrow x - 1$ sont différents des équations $x = x - 1$ et $x = x - 1$
  - $A + 1 \leftarrow 3$ n'est pas possible en langages de programmation et n'est pas équivalente à $A \leftarrow 2$
- Certains langages donnent des valeurs par défaut aux variables déclarées. Pour éviter tout problème il est préférable d'initialiser les variables déclarées.

Partie 3 : Expressions et opérateurs

9 Expressions

10 Opérateurs

Expressions

L'affectation

Var $\leftarrow$ e : attribue la valeur de e à la variable Var

- e peut être une
  - valeur
  - variable
  - expression

Expressions

Une expression est une opération constituée de variables reliées par des opérateurs.

Expressions

Caractéristiques

Var $\leftarrow$ expression

- Une expression est placé toujours à droite de signe $\leftarrow$
- L'évaluation de l'expression fournit une valeur unique qui est le résultat
- Ce résultat est stocké dans la variable à gauche de $\leftarrow$

Exemple

Var $\leftarrow$ expression

- A $\leftarrow$ 30 + 15
- B $\leftarrow$ A + 15

Opérateurs

Selon le nombre d'opérandes, Il existe deux types d'opérateurs :

- Opérateur unaire : qui agit sur un opérateur binaire
- Opérateur binaire : qui agit sur deux opérandes

Exemples

Dans l'expression c $\leftarrow$ a + b, "+" est un opérateur Dans l'expression $a++$; "++" est un opérateur unaire

Opérateurs

Catégories

- des opérateurs arithmétiques
- des opérateurs logiques
- des opérateurs relationnels
- des opérateurs sur les chaînes

Opérateurs

Les opérateurs arithmétiques

Symbole d'opérateur	**Opération**	Exemple
+	Addition	c ← a + b
-	Soustraction	c ← a - b
/	Division	c ← a / b
%	Modulo	c ← a % b
*	Multiplication	c ← a * b

Opérateurs

Les opérateurs logiques

Symbole d'opérateur	**Opération**	Exemple
||	OU logique	c ← a || b
&&	ET logique	c ← a && b
!	NON logique	c ← !a

La variable c est de type booléen

Opérateurs

La table de vérité

x	**y**	x et y	x ou y	x xor y
Vrai	Vrai	Vrai	Vrai	Faux
Vrai	Faux	Faux	Vrai	Vrai
Faux	Vrai	Faux	Vrai	Vrai
Faux	Faux	Faux	Faux	Faux

x	**non x**
Vrai	Faux
Faux	Vrai

Opérateurs

Les opérateurs relationnels

Symbole d'opérateur	**Opération**	Exemple
$=$	Égal à	c $\leftarrow$ a $=$ b
$<$	Inférieur à	c $\leftarrow$ a $<$ b
$<=$	Inférieur ou égal à	c $\leftarrow$ a $<=$ b
$>$	Supérieur à	c $\leftarrow$ a $>$ b
$>=$	Supérieur ou égal à	c $\leftarrow$ a $>=$ b
$<>$	Non égal à	c $\leftarrow$ a $<>$ b

La variable c est de type booléen

Opérateurs

Les opérateurs sur les chaînes de caractères

Symbole d'opérateur	**Opération**	Exemple
&	Concaténation	c ← "Bonjour" & "à tous"

La variable c est de type chaîne de caractères

Opérateurs

Remarques

- On ne peut pas additionner un entier et un caractère
- Pour certains langages de programmation, il est possible qu'on additionne quelques nombres de différents types
- L'opérateur + est utilisé dans certains langages de programmation comme opérateur de concaténation
  - En algorithmique : "bonjour" + " tout le monde" est incorrect
- La division "x/y" donne :
  - un entier si x et y sont entiers
  - un réel si x et y sont entiers
- Avant d'utiliser une variable dans une expression, il est nécessaire qu'une valeur lui ait été affectée.

Opérateurs

Priorité des opérateurs

Pour les opérateurs arithmétiques donnés ci-dessus, l'ordre de priorité est le suivant (du plus prioritaire au moins prioritaire) :

1. () : les parenthèses
2. ^ (élévation à la puissance)
3. * , / (multiplication, division)
4. % (modulo)
5. + , - (addition, soustraction)

En cas de besoin, on utilise les parenthèses pour indiquer les opérations à effectuer en priorité.

Opérateurs

Priorité des opérateurs : Exemple

- 9 + 3 * 4 ???
- (9 + 3) * 4 ???
- 9+3-2 ??? (À priorité égale, l'évaluation de l'expression se fait de gauche à droite)

En cas de besoin, on utilise les parenthèses pour indiquer les opérations à effectuer en priorité.

Partie 4 : Lecture et écriture

11 Introduction

12 L'écriture

13 La lecture

14 Exemple

Introduction

Les instructions de lecture et écriture permettent à la machine de communiquer avec l'utilisateur.

- Les instructions d'écriture : permettent au programme de communiquer des données à l'utilisateur en les écrivant à l'écran ou en l'écrivant dans un fichier ou une base de données.
- Les instructions de lecture : permettent à l'utilisateur de rentrer des données au clavier pour qu'elles soient lues par le programme.

Remarque

A première vue, on peut avoir l'impression que ce sont de fausses définitions !!! Quand l'utilisateur doit écrire au clavier, on appelle cette instruction la lecture, et quand il doit lire sur l'écran on l'appelle l'écriture.

L'écriture

Ecriture de résultats du programme ou d'autres informations : Ecrire ( )

- Programme réalise l'écriture des données à l'écran ou bien dans un fichier
- L'utilisateur peut donc réaliser la lecture des données affichées.

Exemple

Début
Ecrire("Bienvenue en Algorithmique !!!")
Fin

L'écriture

Il existe plusieurs façons pour écrire une expression sur l'écran :

- Ecrire la valeur¡contenu¿ d'une variable A : Ecrire (A)
- Ecrire un message préventif : Ecrire ("Entrer un entier ?"
- Ecrire un résultat à l'écran : Ecrire ('La valeur de A est" , A)

La lecture

La lecture des données saisies par l'utilisateur : Lire( )

- Programme en Attente de saisie
- Utilisateur saisit des données au clavier
- Le programme fait la Lecture des données
- Le programme stocke ces données dans des variables préalablement déclarées

Exemple

```
Variables A : Entier
Début
Lire(A)
Fin
```

La valeur entrée au clavier dans la zone mémoire nommée A

La lecture

Remarques

- L'instruction lire cause l'arrêt du programme afin de pouvoir récupérer une entrée.
- Comme il est possible au programmeur d'affecter une valeur à une variable de son choix, on peut permettre à un simple utilisateur du programme de faire pareil avec l'instruction lire.
- Il est conseillé avant de lire une variable, d'écrire des messages à l'écran, afin de prévenir l'utilisateur de ce qu'il doit taper au clavier.

Exemple

Variables Nb : Entier
Début
    Ecrire (“Saisissez un entier”)
    Lire(Nb)
    Ecrire (“Vous avez saisi le nombre ”, Nb, “Merci pour votre temps”)
Fin

Partie 5 : Les tests

15 Introduction

16 La condition

17 Exemple de tests

18 Exemple de tests

19 Tests imbriqués

Introduction

Définition

Un test est une instruction qui permet à l'algorithme de choisir l'ensemble d'instructions à exécuter selon l'évaluation d'une condition.

Introduction

Structure 1

Variables Nb : Entier
Début
    Si Condition **Alors**
        Instruction 1
        Instruction 2
        ...
    FinSI
Fin

Introduction

Structure 2

Variables Nb : Entier
Début
    Si Condition **Alors**
        Instruction 1
        ...
    Sinon
        Instruction 1
        ...
    FinSI
Fin

Introduction

Structure 2

Variables Nb : Entier
Début
    Si Condition **Alors**
        Instruction 1
        ...
    SinonSi
        Instruction 1
        ...
    Sinon
        Instruction 1
        ...
    FinSI
Fin

La condition

Définition

- Une condition est une expression qui permet aux structures conditionnels de savoir quel bloc d'instructions à exécuter
- Une condition se comporte généralement de trois composants :
  - Une valeur
  - Un opérateur de comparaison
  - Une variable
- On dit qu'une expression est une condition quand elle est associée par exemple à un test **SI...ALORS...SINON**.

La condition

Types de conditions

Il existe deux types de conditions :

- condition simple est formée au plus de deux opérandes et un opérateur.
- condition composée est une condition composée de plusieurs conditions simples reliées par des opérateurs logiques: ET, OU, OU exclusif (XOR) et NON

La condition

Exemples

- condition simple :
  - A > B
  - A = B
  - A && B
- condition composée :
  - (X>=2) ET (X<=6)
  - (N%3=0) OU (N%2=0)

Exemple de tests

Test 1

Algorithme valeurAbsolue1
Rôle : affiche la valeur absolue d'un entier
Entrées: la valeur de x
Sorties : la valeur absolue
Variable x : Réel
Début
    Ecrire(" Entrez un réel : ")
    Lire (x)
    Si $x < 0$ **alors**
        Ecrire ("la valeur absolue de ", x, "est:",-x)
    Sinon
        Ecrire ("la valeur absolue de ", x, "est:",x)
    FinSI
Fin

Exemple de tests

Test 2

Algorithmes valeurAbsolue1
Rôle : affiche la valeur absolue d'un entier
Entrées: la valeur de x
Sorties : la valeur absolue
Variable x : Réel
Début
    Ecrire(" Entrez un réel : ")
    Lire (x)
    Si $x < 0$ **alors**
        x $\leftarrow$ -x
    FinSI
    Ecrire ("la valeur absolue de ", x, "est:",x)
Fin

Tests imbriqués

Un test est dit imbriqué s'il contient d'autres tests. Il permet de gagner en temps d'exécution du programme.

Si Condition1 **Alors**
    Si Condition2 **Alors**
        Instructions
    Sinon
        Instructions
    FinSI
Sinon
    Si Condition3 **Alors**
        Instructions
    Sinon
        Instructions
    FinSI
FinSI

Tests imbriqués

Exemple 1

Version 1: Algorithme qui affiche le signe d'un nombre **sans** tests imbriqués

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

Tests imbriqués

Exemple 2

Version 2: Même algorithme **avec** des tests imbriqués.

```
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

Comparaison

- la version 1 fait trois tests systématiquement
- la version 2, si le nombre est négatif on ne fait qu'un seul test
- utiliser les tests imbriqués pour limiter le nombre de tests et placer d'abord les conditions les plus probables

Partie 6 : Les boucles

20 Présentation

21 La boucle Tant que

22 La boucle répéter jusqu'à

23 La boucle Pour

24 Choix d'un type de boucle

Présentation

Définition

- Les boucles servent à répéter l'exécution d'un bloc d'instructions un certain nombre de fois.
- En littérature, on les appellent aussi :
  - des structures répétitives
  - des structures itératives.
- Si une boucle répète N fois un bloc d'instructions; on dit alors la boucle a réalisé N itérations.

Présentation

Types

On distingue trois types de boucles en langages de programmation :

- Les boucles **TANT QUE**
- Les boucles **POUR**
- Les boucles **Répéter . . . Jusqu'à**

La boucle Tant que

Sert à répéter des instructions tant qu'une condition donnée est vraie.

Structure

TantQue (condition)

instructions

FinTantQue

Principe

1. si la condition est vraie,
   - On exécute "instructions",
   - On teste de nouveau la condition.
   - Si elle est encore vraie, on reprend, ...
2. si la condition est fausse,
   - on sort de la boucle et
   - on exécute les instructions suivantes

La boucle Tant que

Remarques

- Le nombre d'itérations dans une boucle **TantQue** est inconnu au moment d'entrée dans la boucle. Ce nombre 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 infiniment.
- Attention aux boucles infinies !!!

La boucle Tant que

Exemple

Variables som, i, N : entier
Entrées : N
Début
    Ecrire ('Entrer N ?")
    Lire (N)
    $som \leftarrow 0$
    $i \leftarrow 0$
    TantQue $(i < 0)$
        $som \leftarrow som + i$
        $i \leftarrow i + 1$
    FinTantQue
    Ecrire (" La somme des N premiers entiers = ", som)
Fin

La répéter jusqu'à

Structure

Répéter

instructions

Jusqu'à condition

Principe

- Exécuter instructions
- Si condition est fausse,
  - revenir et exécuter instructions ....
- Si condition est vraie,
  - Sortir de la boucle

La boucle répéter jusqu'à

Exemple

Un algorithme qui fait la somme des N premiers entiers.

Variables som, i, N : entier
Entrée N
Début
    Ecrire (" Entrez N ")
    Lire (N)
    $som \leftarrow 0$
    $i \leftarrow 1$
    Répéter
        $som \leftarrow som + i$
        $i \leftarrow i + 1$
    Jusqu'à $I > N$
    Ecrire (x, " La somme des N premiers entiers = ", som)
Fin

La boucle Pour

Permet de répéter un bloc d'instructions en faisant évoluer un compteur (variable particulière) entre une valeur initiale et une valeur finale.

Structure

Pour compteur ← Valeur initiale **à** Valeur finale **PAS** ← Valeur du pas
    instructions
FinPour

La boucle Pour

Remarques

- Le nombre d'itérations est connu avant le début de la boucle
- **Compteur** est une variable de type entier. 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

La boucle Pour

Exemple

Calcul de x^n :

Variables x, puiss : réel
n, i : entier
Debut
Ecrire (“ Entrez une valeur et sa puissance ”)
Lire (x,n)
puiss $\leftarrow$ 1
Pour *i* $\leftarrow$ 1 **à** n **PAS** $\leftarrow$ 1
puiss $\leftarrow$ *puiss* $*$ *x*
FinPour
Ecrire (x, “ à la puissance ”, n, “ est égal à ”, puiss)
Fin

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 **répéter 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**

Partie 5 : Les Tableaux

25 Les tableaux à une dimension

26 Les tableaux à deux dimension

Les Tableaux

On suppose qu'on voudrait récupérer 25 valeurs pour calculer leurs moyenne. Sans tableau, on va procéder de la maniére suivante :

- On déclare 25 variables de type réel (cela fera 25 cases mémoires pour les valeurs et 25 autres cases mémoires pour leurs identifiants)
- On demande à l'utilisateur de saisir 25 valeurs
- On stocke ces valeurs dans les 25 valeurs précédemment crées
- On calcule la moyenne

Les Tableaux

Présentation

Un tableau permet de récupérer un ensemble de valeurs qui seront accessible par un même identifiant. Les valeurs peuvent être de n'importe quel type sauf qu'il n'est pas possible de stocker dans un même tableau des valeurs de types différents.

Les tableaux à une dimension

Déclaration

Pour déclarer un tableau il faut préciser le nombre et le type de valeurs qu'il contiendra.

- Tableau nomDuTableau [N] : LE type des valeurs
- Sachant que N est le nombre des valeurs à stocker.

Les valeurs seront accessibles grâce à un indice $i \in [0; N-1]$:

- nomDuTableau [i]

Exemple

- Déclaration : Tableau Note [25] : Réel
- Accès à la 10 valeur : Note [9]

Les tableaux à une dimension

Exemple : Calcul de la moyenne de 25 valeurs.

Variables tableau Note[25], i, somme : Entier
moyenne : Réel
Début
Pour $i \leftarrow 1$ **à** 24 **PAS** $\leftarrow 1$
Ecrire("Entrez une note : ")
Lire(Note[i])
FinPour
$somme \leftarrow 0$
Pour $i \leftarrow 1$ **à** 24 **PAS** $\leftarrow 1$
$somme \leftarrow somme + Note[i]$
FinPour
$moyenne \leftarrow somme/25$
Ecrire (" La moyenne des notes est : ", moyenne)
Fin

Les tableaux à une dimension

Exemple : Calcul de la moyenne de 25 valeurs.

Proposez une amélioration pour ne pas fixer le nombre des valeurs à saisir.

Les tableaux à une dimension

Exemple : Calcul de la moyenne de n valeurs (n<200).

```
Constante MAX 200
Variables tableau Note[MAX], i, somme, n : Entier
          moyenne : Réel
Début
    Ecrire ("Entrez le nombre des valeurs :")
    Lire(n)
    Pour i ← 1 à n - 1 PAS ← 1
        Ecrire("Entrez une note : ")
        Lire(Note[ i ])
    FinPour
```

Les tableaux à une dimension

Exemple : Calcul de la moyenne de n valeurs (la suite.

somme $\leftarrow 0$
Pour *i* $\leftarrow 1$ **à** n - 1 **PAS** $\leftarrow 1$
    somme $\leftarrow$ *somme* $+$ *Note*[*i*]
FinPour
moyenne $\leftarrow$ *somme*/*n*
Ecrire (“ La moyenne des notes est : ”, moyenne)
Fin

Les tableaux à une dimension

Remarques

- A la compilation la constante Max sera remplacée par le nombre 200
- Même si on donne à n la valeur 10, l'ordinateur va réserver la place pour 200 valeurs de type réel.

Les tableaux à une dimension

Les tableaux dynamiques

Lorsqu'on ne connait pas le nombre des valeurs qu'un utilisateur voudrait saisir, déclarer un tableau avec une taille maximale n'est pas une solution optimisée pour la mémoire de l'ordinateur. La meilleure solution est de déclarer un tableau sans préciser sa taille en utilisant l'instruction :

- Allocation(nom,nombre,type)

Les tableaux à une dimension

Les tableaux dynamiques

```
Variables tableau Note[ ], i, somme, n : Réel
          i, n : Entier
Début
    Ecrire ("Entrez le nombre des valeurs :")
    Lire(n)
    allocation(Notes,n,réel)
    Pour i ← 1 à n - 1 PAS ← 1
        Ecrire("Entrez une note : ")
        Lire(Note[ i ])
    FinPour
...
Libère(Note)
```

Les tableaux à deux dimension

Déclaration

- Tableau nomDuTableau [M][N] : LE type des valeurs
- Sachant que M est le nombre de lignes et N le nombre de colonnes à stocker
- Cette déclaration signifie : réserver un espace de mémoire pour M x N valeurs

Les valeurs seront accessibles grâce à deux indices $i \in [0; M-1]$ et $j \in [0; N-1]$:

- nomDuTableau [i] [j]

- Déclaration : Tableau matrice [25] [25] : Réel
- Accès à la 10ème valeur de la 20ème : matrice [19] [9]
- Initialisation : $T_1[3][3] = \{\{ 1, 2, 3 \}, \{ 4, 5, 6 \}, \{ 7, 8, 9 \}\}$

Les tableaux à deux dimension

Lecture d'une matrice

```
Variables tableau matrice [ 10 ][ 10 ], i, j, k, n : Entier
Début
    Ecrire ("Entrez le nombre de lignes et de colonnes :")
    Lire(n,k)
    Pour i ← 0 à n - 1 PAS ← 1
        Ecrire("Entrez les éléments de la ", i+1, "ligne : ")
            Pour j ← 0 à k - 1 PAS ← 1
                Lire(matrice[ i ][ j ])
            FinPour
    FinPour
Fin
```

Les tableaux à deux dimension

Ecriture d'une matrice

```
Variables tableau matrice [ 10 ][ 10 ], i, j, k, n : Entier
Début
    Pour i ← 0 à n - 1 PAS ← 1
        Pour j ← 0 à k - 1 PAS ← 1
            Ecrire(matrice[ i ][ j ])
        FinPour
    FinPour
Fin
```

Les tableaux à deux dimension

Exemple : Somme de deux matrices

```
Constante N 20
variables Tableau A[ N ][ N ],B[ N ][ N ],C[ N ][ N ], i, j, n : Entier
Ecrire(" Donner la taille des matrices(¡20) : ")
Lire(n)
Début
    Pour i ← 0 à n - 1 PAS ← 1
        Ecrire("donner les éléments de la ",i," ligne : ")
        Pour j ← 0 à n - 1 PAS ← 1
            Lire(A[ i ][ j ])
        FinPour
    FinPour
```

Les tableaux à deux dimension

Exemple : Somme de deux matrices (Suite)

```
Pour i ← 0 à n - 1 PAS ← 1
        Ecrire("donner les éléments de la ",i," ligne : ")
        Pour j ← 0 à n - 1 PAS ← 1
            Lire(B[ i ][ j ])
        FinPour
FinPour
Pour i ← 0 à n - 1 PAS ← 1
        Pour j ← 0 à n - 1 PAS ← 1
            C[ i ][ j ] = A[ i ][ j ] + B[ i ][ j ]
        FinPour
FinPour
```

Les tableaux à deux dimension

Exemple : Somme de deux matrices (Suite)

```
    Pour i ← 0 à n - 1 PAS ← 1
        Pour j ← 0 à n - 1 PAS ← 1
            Ecrire(C[ i ][ j ])
        FinPour
    FinPour
Fin
```