

Algorithmique & Langage C

SMI

Faculté Poly disciplinaire Khouribga

Introduction

L'algorithmique est un terme d'origine arabe (**Alkhaouarizmi** : الخوارزمي), comme algèbre, alcool etc.

Qu'est-ce que l'algorithmique ?

Un algorithme, c'est une suite finie d'instructions, qui une fois exécutée correctement, conduit à un résultat donné.

Exemple:

Cafetière



Expresso

instructions

Farine, oeufs, chocolat,
etc....



Gâteau

recette



L'algorithmique, vous la pratiquez tous les jours et depuis longtemps...

Introduction

Problème : Expliquer à la «machine» comment elle doit s'y prendre.

Mais... comment le lui dire ?

Comment le lui apprendre à le faire?

Comment s'assurer qu'elle fait ce travail aussi bien que nous ?

Mieux que nous?

Objectif : Demander à la «machine» d'effectuer un travail à notre place.

Les objectifs de ce cours

Résoudre des problèmes «comme» une machine ;

Savoir ***expliciter*** son raisonnement ;

Savoir ***formaliser*** son raisonnement ;

Concevoir (et écrire) des ***algorithmes*** (séquence d'instructions qui décrit comment résoudre un problème particulier).

Thèmes abordés dans ce cours

Apprentissage d'un langage, exemple: Le Langage C

Notions de base;

Structures de données;

Résolution de problèmes complexes ;

Algorithmique?

- Permet de décrire la résolution d'un problème en utilisant des opérations qui sont ceux des ordinateurs sans toutefois être particulier à un ordinateur précis.
- Un algorithme peut d'abord être écrit en langage « parlé » décrivant la succession des opérations qui doivent être faites. Puis par raffinement successif se transformer en langage algorithmique (*pseudocode*) qui se trouve à mi chemin entre le parlé et le code.
- Un algorithme, traduit dans un langage compréhensible par l'ordinateur (ou langage de programmation, exemple : le Turbo Pascal, C, C++,), donne un programme, qui peut ensuite être exécuté, pour effectuer le traitement souhaité.

Les étapes d'exécution d'un algorithme

Problème → Analyse → Algorithme → Programme → Compilation → Exécution

Structure de l'Algorithme

Un algorithme informatique combine généralement 4 briques:

- L'affectation de variables ;
- la lecture / écriture ;
- les tests ;
- les boucles.

Squelette d'un Algorithme

- Un algorithme doit être lisible et compréhensible par plusieurs personnes ;
- Il doit donc suivre des règles ;
- Il est composé d'une entête et d'un corps ;

Le corps est composé :

- du mot clef **début** ;
- d'une suite d'instructions;
- du mot clef **fin**.

Exemple de code:

Variables A, B, C : Entier

Début

C ← 4

A ← 3

B ← 7

C ← A + B

Fin

NOTION DE VARIABLE

- **Une variable est une entité qui contient une information :**
 - * une variable possède un nom, on parle **d'identifiant** ;
 - * une variable possède une valeur ;
 - * une variable possède un type qui caractérise l'ensemble des valeurs que peut prendre la variable.
- **L'ensemble des variables sont stockées dans la mémoire de l'ordinateur**
- **On peut faire l'analogie avec une armoire d'archive qui contiendrait des tiroirs étiquetés :**
 - * l'armoire serait la mémoire de l'ordinateur ;
 - * les tiroirs seraient les variables (i.e. l'étiquette correspondrait à l'identificateur) ;
 - * le contenu d'un tiroir serait la valeur de la variable correspondante ;
 - * la couleur du tiroir serait le type de la variable (bleu pour les factures, rouge pour les bons de commande, etc.).

VARIABLES

Dans un programme informatique, on va avoir en permanence besoin de stocker provisoirement des valeurs (des nombres, des chiffres, des caractères, des chaînes de caractère,).

Déclaration des variables:

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,...).

CHOIX D'IDENTIFICATEUR

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

- Un nom d'une variable doit commencer par une lettre alphabétique ;
- Doit être constitué uniquement de lettres, de chiffres et du soulignement _ (Éviter les caractères de ponctuation et les espaces) ;
- Doit être différent des mots réservés du langage (par exemple en Pascal : **integer, for, else, case, ...**)
- La longueur du nom doit être inférieure à la taille maximale spécifiée par le langage utilisé.

Conseil : pour la bonne lisibilité du code, il faut choisir des noms significatifs qui décrivent les données manipulées.

Exemples : TotalVentes2004, Prix_TTC, Prix_HT...

QU'EST CE QU'UN TYPE DE DONNÉES...

Le type d'une variable caractérise :

- l'ensemble de valeurs que peut prendre la variable ;
- l'ensemble des actions que l'on peut effectuer sur une variable ;

Lorsqu'une variable apparaît dans l'entête d'un algorithme lui associe un type en utilisant la syntaxe suivante :

IdentifiantDeLaVariable : Type de la variable

Exemple

age : Entier

taille : Chaîne de caractères

Une fois qu'un type de données est associé à une variable, cette variable ne peut plus en changer.

Une fois qu'un type de données est associé à une variable, le contenu de cette variable doit obligatoirement être du même type.

QU'EST CE QU'UN TYPE DE DONNÉES...

Par exemple, dans l'exemple précédent on a déclaré **a** et **b** comme des entiers :

- **a** et **b** dans cet algorithme ne pourront pas stocker des réels ;
- **a** et **b** dans cet algorithme ne pourront pas changer de type ;

Il y a deux grandes catégories de types :

- types simples ;
- types complexes ;

LES TYPES SIMPLES (PRIMITIFS)...

Le type d'une variable détermine l'ensemble des valeurs qu'elle peut prendre, les types simples offerts par la plus part des langage se subdivisent en deux grandes catégories :

- Ceux dont le nombre d'éléments est fini, les **dénombrables** ;
- Ceux dont le nombre d'éléments est infini, les **indénombrables**.

LES TYPES SIMPLES DÉNOMBRABLES...

- **booléen**, les variables ne peuvent prendre que les valeurs **VRAI** ou **FAUX** ;
- **intervalle**, les variables ne peuvent prendre que les valeurs entières définies dans cet intervalle, par exemple 1..10 ;
- **énuméré**, les variables ne peuvent prendre que les valeurs explicitées, par exemple les jours de la semaine (du lundi au dimanche). Ce sont les seuls types simples qui peuvent être définis par l'informaticien.
- **caractères**

Exemples :

masculin : booléen

mois : 1..12

jour : JoursDeLaSemaine

CAS DES ÉNUMÉRÉS ...

- Si vous voulez utiliser des énumérés, vous devez définir le type dans l'entête de l'algorithme en explicitant toutes les valeurs de ce type de la façon suivante :
- $\text{NomDuType} = \{\text{valeur1}, \text{valeur2}, \dots, \text{valeurn}\}$
- Exemple :
`JoursDeLaSemaine = {Lundi, Mardi, Mercredi, Jeudi, Vendredi, Samedi, Dimanche}`

LES TYPES SIMPLES INDÉNOMBRABLES...

- **Entier ;**
- **Réel ;**
- **Chaîne de caractères**, par exemple 'Ouarzazate' , 'FPO','FST'.

Exemples :

age : Entier

taille : Réel

nom : Chaîne de caractères

DÉCLARATION DE VARIABLES

Variables listes d'identifiant: **type**

Exemple:

```
Variables i, j, k: Entier  
      x, y : Réel  
      OK : Booléen  
      ch1, ch2 : Chaîne de caractères
```

INSTRUCTION D'AFFECTATION

La seule chose qu'on puisse faire avec une variable, c'est **l'affecter**, c'est-à-dire **lui attribuer une valeur** (ça consiste en fait à remplir ou à modifier le contenu d'une zone mémoire).

En pseudo code, l'affectation se note avec le signe \leftarrow (**flèche vers la gauche**)

Var \leftarrow **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 compatible

L'affectation ne modifie que ce qui est à gauche de la flèche

Exemples :

i \leftarrow 1	j \leftarrow i	k \leftarrow i+j
x \leftarrow 10.3		OK \leftarrow FAUX
ch1 \leftarrow "FPO"		ch2 \leftarrow ch1
x \leftarrow 4		x \leftarrow j

EXERCICES

Donner 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

EXERCICES

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

Variables A, B: Entier

Début

A ← 5

B ← 2

A ← B

B ← A

Fin

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

CORRECTION

Après

A ← **5**

B ← **2**

A ← **B**

B ← **A**

La valeur des variables est :

A = 5

B = ?

A = 5

B = 2

A = 2

B = 2

A = 2

B = 2

Les deux dernières instructions ne permettent donc pas d'échanger les deux valeurs de B et A, puisque l'une des deux valeurs (celle de A) est ici écrasée.

Si l'on inverse les deux dernières instructions, cela ne changera rien du tout, hormis le fait que cette fois c'est la valeur de B qui sera écrasée.

EXERCICES

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

A, B, C Entier

Début

C ← A

A ← B

B ← C

Fin

On est obligé de passer par une variable dite temporaire (la variable C)

EXERCICES

Exercice 1 : Quelles seront les valeurs des variables A et B après exécution des instructions suivantes ?

Variables A, B : Entier

Début

A ← 1

B ← A + 3

A ← 3

Fin

Exercice 2 : Quelles seront les valeurs des variables A, B et C après exécution des instructions suivantes ?

Variables A, B, C : Entier

Début

A ← 5

B ← 3

C ← A + B

A ← 2

C ← B - A

Fin

CORRECTION

Exercice 1

Après

A ← **1**
B ← **A** + **3**
A ← **3**

La valeur des variables est :

A = 1	B = ?
A = 1	B = 4
A = 3	B = 4

Exercice 2

Après

A ← **5**
B ← **3**
C ← **A** + **B**
A ← **2**
C ← **B** - **A**

La valeur des variables est :

A = 5	B = ?	C = ?
A = 5	B = 3	C = ?
A = 5	B = 3	C = 8
A = 2	B = 3	C = 8
A = 2	B = 3	C = 1

EXERCICES

Exercice 3 : Quelles seront les valeurs des variables A et B après exécution des instructions suivantes ?

```
Variables A, B : Entier
Début
    A ← 5
    B ← A + 4
    A ← A + 1
    B ← A - 4
Fin
```

Exercice 4 : Quelles seront les valeurs des variables A, B et C après exécution des instructions suivantes ?

```
Variables A, B, C : Entier
Début
    A ← 3
    B ← 10
    C ← A + B
    B ← A + B
    A ← C
Fin
```

CORRECTION

Exercice 3

Après

A ← **5**
B ← **A** + **4**
A ← **A** + **1**
B ← **A** - **4**

La valeur des variables est :

A = 5	B = ?
A = 5	B = 9
A = 6	B = 9
A = 6	B = 2

Exercice 4

Après

A ← **3**
B ← **10**
C ← **A** + **B**
B ← **A** + **B**
A ← **C**

La valeur des variables est :

A = 3	B = ?	C = ?
A = 3	B = 10	C = ?
A = 3	B = 10	C = 13
A = 3	B = 13	C = 13
A = 13	B = 13	C = 13

EXERCICES

Exercice 5 : On dispose de trois variables A, B et C. Ecrivez un algorithme transférant à B la valeur de A, à C la valeur de B et à A la valeur de C (toujours quels que soient les contenus préalables de ces variables).

Exercice 6 : Que produit l'algorithme suivant ?

```
Variables A, B, C : Caractères  
Début  
    A ← "423"  
    B ← "12"  
    C ← A + B  
Fin
```

Exercice 7 : Que produit l'algorithme suivant ?

```
Variables A, B, C : Caractères  
Début  
    A ← "423"  
    B ← "12"  
    C ← A & B  
Fin
```

CORRECTION

Exercice 5

Variables A, B, C : Caractères

Début

D ← C

C ← B

B ← A

A ← D

Fin

En fait, quel que soit le nombre de variables, une seule variable temporaire suffit...

OPÉRATEUR, OPÉRANDE ET EXPRESSION...

Un **opérateur** est un symbole d'opération qui permet d'agir sur des variables ou de faire des “calculs”.

Une **opérande** est une entité (variable, constante ou expression) utilisée par un opérateur.

Une **expression** est une combinaison d'opérateur(s) et d'opérande(s), elle est évaluée durant l'exécution de l'algorithme, et possède une valeur (son interprétation) et un type.

OPÉRATEUR, OPÉRANDE ET EXPRESSION...

Par exemple dans $a+b$:

a est l'opérande gauche

$+$ est l'opérateur

b est l'opérande droite

$a+b$ est appelé une expression

Si par exemple a vaut 2 et b vaut 3, l'expression $a+b$ vaut 5

Si par exemple a et b sont des entiers, l'expression $a+b$ est un entier

OPÉRATEURS...

Un opérateur est associé à un type de donnée et ne peut être utilisé qu'avec des variables, des constantes, ou des expressions de ce type

- Par exemple l'opérateur $+$ ne peut être utilisé qu'avec les types arithmétiques(naturel, entier et réel) ou (exclusif) le type chaîne de caractères
- **On ne peut pas additionner un entier et un caractère**

Toutefois exceptionnellement dans certains cas on accepte d'utiliser un opérateur avec deux opérandes de types différents, c'est par exemple le cas avec les types arithmétiques ($2+3.5$)

La signification d'un opérateur peut changer en fonction du type des opérandes.

OPÉRATEURS...

Exemple

L'opérateur + avec des entiers aura pour sens l'addition, mais avec des chaînes de caractères aura pour sens la **concaténation**

- $2+3$ vaut 5
- "bonjour" + " tout le monde" vaut "bonjour tout le monde"

OPÉRATEURS...

Une expression peut être une valeur, une variable ou une opération constituée de variable 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 : =, <, >, <=, >= ;
- Des opérateurs sur les chaînes : & (concaténation).

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

PRIORITÉ DES OPÉRATEURS

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

- $^$: (puissance) ;
- $*$, $/$ (multiplication, division) ;
- $\%$ (modulo) ;
- $+$, $-$ (addition, soustraction).

Exemple : $2+3*7$ vaut 23

En cas de besoin (ou de doute), on utilise les parenthèse pour indiquer les opérations à effectuer en priorité.

Exemple : $(2+3)*7$ vaut 35

LES INSTRUCTIONS D'ENTRÉES-SORTIES : LECTURE ET ÉCRITURE...

Communiquer avec l'utilisateur :

- La lecture permet d'entrer des données à partir du clavier ;
- En pseudo-code, on note: **lire(var)** : *La machine met la valeur entrée au clavier dans la zone mémoire nommée var.*

Remarque : le programme s'arrête lorsqu'il rencontre une instruction Lire et ne se poursuit qu'après la frappe d'une valeur au clavier.

- L'écriture permet d'afficher des résultats ou du texte sur l'écran ;
- En pseudo-code, on note : **écrire(var)** : La machine affiche le contenu de la zone mémoire var.

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.

LES INSTRUCTIONS D'ENTRÉES-SORTIES : LECTURE ET ÉCRITURE...

Un algorithme peut avoir des interactions avec l'utilisateur. Alors, Il peut afficher un résultat (du texte ou le contenu d'une variable) et demander à l'utilisateur de saisir une information afin de la stocker dans une variable.

- Pour afficher une information on utilise la commande **écrire** suivie entre parenthèses de la chaîne de caractères entre guillemets et/ou des variables de type simple à afficher séparées par des virgules, par exemple :

écrire("La valeur de la variable a est", a)

- Pour donner la possibilité à l'utilisateur de saisir une information on utilise la commande **lire** suivie entre parenthèses de la variable de type simple qui va recevoir la valeur saisie par l'utilisateur, par exemple :

lire(b)

EXEMPLE (LECTURE ET ÉCRITURE)

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

Algorithme CalculDouble

Variables A, B: Entier

Début

Écrire(" entrer la valeur de A")

Lire(A)

B←2*A

Écrire("le double de", A, "est:",
 B)

Fin

EXERCICE

Exercice : Écrire un algorithme qui vous demande de saisir votre nom puis votre prénom et qui affiche ensuite votre nom complet.

Correction :

Algorithme AffichageNomComplet

Variables Nom, Prenom, NomComplet: Chaîne de caractères

Début

 écrire("entrez votre nom")

 Lire(Nom)

 écrire("entrez votre prenom")

 Lire(prenom)

 NomComplet ← Nom & Prenom

 écrire("Votre nom complet est : ",
 NomComplet)

EXERCICES

Exercice : Quel résultat produit le programme suivant ?

```
Variables val, puiss: Entier
Début
    val ← 99
    puiss ← val^3
    Écrire val
    Écrire puiss
Fin
```

EXERCICES

Exercice : Écrire un algorithme qui demande un nombre à l'utilisateur, puis qui calcule et affiche le carré de ce nombre.

Correction

```
Variables nb, carre : Entier
Début
    Ecrire ("Entrez un nombre : " )
    Lire (nb)
    carre ← nb * nb
    Ecrire ("Son carré est : ", carre)
Fin
```

EXERCICES

Exercice : Écrire un algorithme qui lit le prix HT d'un article, le nombre d'articles et le taux de TVA, et qui fournit le prix total TTC correspondant. Faire en sorte que des libellés apparaissent clairement.

Correction

Variables nb, pht, ttva, pttc : Réel

Début

Ecrire ("Entrez le prix hors taxes :")

Lire (pht)

Ecrire ("Entrez le nombre d'articles :")

Lire (nb)

Ecrire ("Entrez le taux de TVA :")

Lire (ttva)

$pttc \leftarrow nb * pht * (1 + ttva)$

Ecrire ("Le prix toutes taxes est : ", pttc)

Fin

Instructions conditionnelles (Tests)...

Jusqu'à présent les instructions d'un algorithme étaient **toutes** interprétées
Séquentiellement

Mais il se peut que l'on veuille conditionner l'exécution d'un algorithme. Par exemple la résolution d'une équation du second degré est conditionnée par le signe de Δ .

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

L'INSTRUCTION SI ALORS SINON...

L'instruction si alors sinon permet de conditionner l'exécution d'un algorithme à la valeur d'une expression booléenne.

Sa syntaxe est :

si expression booléenne alors

 suite d'instructions exécutées si l'expression est vrai

sinon

 suite d'instructions exécutées si l'expression est fausse

finsi

La deuxième partie de l'instruction est optionnelle, on peut avoir la syntaxe :

si expression booléenne alors

 suite d'instructions exécutées si l'expression est vrai

finsi

La condition peut être une condition simple ou une condition composée de plusieurs conditions

EXEMPLE...

```
Algorithme  ValeurAbsolue
Variables  nombre, laValeurAbsolue : Entier
début
    si unEntier >= 0 alors
        laValeurAbsolue ← nombre
    sinon
        laValeurAbsolue ← - nombre
    finssi
fin
```

TESTS IMBRIQUÉS

Algorithme TestImbriques

Variables n : Entier

Début

Écrire (" entrer un nombre: ")

Lire(n)

Si $n < 0$ alors

Écrire ("Ce nombre est négatif")

Sinon

Si $n = 0$ alors

Écrire ("Ce nombre est nul")

Sinon

Écrire ("Ce nombre est positif")

Finsi

Finsi

Fin

EXERCICE

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, 0,4 DH pour un nombre compris entre 10 et 20 et 0,3 DH au-delà.

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

CORRECTION

```
Variables    copies : Entier
             prix  : Réel

Début
    Ecrire("Nombre de photocopies: ")
    Lire(copies)
    Si copies < 10 Alors
        prix ← copies*0,5
    sinon
        si copies < 20 Alors
            prix ← copies*0,4
        sinon
            prix ← copies*0,3
        Finsi
    Finsi
    Ecrire("Le prix à payer est : ", prix)

Fin
```

L'INSTRUCTION CAS...

Lorsque l'on doit comparer une **même** variable avec plusieurs valeurs, comme par exemple :

```
si a=1 alors
    faire chose 1
sinon
    si a=2 alors
        faire chose 2
    sinon
        si a=4 alors
            faire chose 3
        sinon
            . . .
    finsi
finsi
```

On peut remplacer cette suite de **si** par l'instruction **cas**

L'INSTRUCTION CAS ...

Sa Syntaxe est :

```
cas où v vaut
    v1 : action1
    v2: action2
    v3 : action3
    autre : action n
```

Fincas

- Où v_1, v_2, v_2, v_3 sont des constantes de type scalaire (entier, énuméré ou caractère) ;
- **Action i** est exécuté si $v=v_i$, avec $i=1,2,3$ (on quitte ensuite l'instruction cas) ;
- **Action n** est exécuté si $v \neq v_i, i=1,2,3$.

EXEMPLE...

```
Algorithme InstructionCas
Variables mois : Entier
Variables message : Chaîne de Caractères
Début
    cas où mois vaut
        7,8 : message ← "Vacances"
        autre : message ← "Travail"
    Fincas
Fin
```

EXERCICES

Ecrire un algorithme qui demande deux nombres à l'utilisateur et l'informe ensuite si leur produit est négatif ou positif (on laisse de côté le cas où le produit est nul). Attention toutefois : on ne doit **pas** calculer le produit des deux nombres.

Correction

Variables m, n : Entier

Début

Ecrire ("Entrez deux nombres :")

Lire (m, n)

Si (m > 0 ET n > 0) OU (m < 0 ET n < 0)

Alors

Ecrire ("Leur produit est
positif")

Sinon

Ecrire ("Leur produit est
négatif")

Finsi

Fin

EXERCICES

Ecrire un algorithme qui demande deux nombres à l'utilisateur et l'informe ensuite si le produit est négatif ou positif (on inclut cette fois le traitement du cas où le produit peut être nul). Attention, on ne doit pas calculer le produit !

Correction

Variables m, n : Entier

Début

Ecrire ("Entrez deux nombres : ")

Lire (m, n)

Si m = 0 OU n = 0 Alors

Ecrire ("Le produit est nul")

Sinon

Si (m < 0 ET n < 0) OU (m > 0 ET n > 0) Alors

Ecrire ("Le produit est positif")

Sinon

Ecrire ("Le produit est négatif")

Finsi

Finsi

Fin

EXERCICES

Ecrire un algorithme qui demande un nombre à l'utilisateur, et l'informe ensuite si ce nombre est positif ou négatif (on inclut cette fois le traitement du cas où le nombre vaut zéro).

Correction

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

EXERCICES

Ecrire un algorithme qui demande l'âge d'un enfant à l'utilisateur. Ensuite, il l'informe de sa catégorie :

"Poussin" de 6 à 7 ans

"Pupille" de 8 à 9 ans

"Minime" de 10 à 11 ans

"Cadet" après 12 ans

Correction

Variable age : Entier

Début

Ecrire ("Entrez l'âge de l'enfant :
")

Lire (age)

Si (age >= 12) Alors

Ecrire ("Catégorie Cadet")

Sinon

EXERCICES

```
Si age >= 10 Alors
    Ecrire ("Catégorie Minime")
Sinon
    Si age >= 8 Alors
        Ecrire ("Catégorie Pupille")
    Sinon
        Si age >= 6 Alors
            Ecrire ("Catégorie Poussin")
        Finsi
    Finsi
Finsi
Finsi
```

Fin

On peut évidemment écrire cet algorithme de différentes façons, ne serait-ce qu'en commençant par la catégorie la plus jeune.

LES ITÉRATIONS

L'orque l'on veut répéter plusieurs fois un traitement, plutôt que de copier n fois la ou les instructions, on peut demander à l'ordinateur d'exécuter n fois un morceau de code

- il existe deux catégories d'itérations:
 - les itérations déterministes: quand le nombre de boucle des défini à l'entrée de la boucle
 - les itérations indéterministes: l'exécution de la prochaine boucle est conditionnée par une expression booléenne

LES ITÉRATIONS DÉTERMINISTES

Pour faire des boucles déterministes, il faut utiliser l'instruction **pour**

La syntaxe de l'instruction pour est la suivante :

```
pour compteur variantDe valeurInitiale à valeurFinale faire  
    instructions à exécuter à chaque boucle  
finpour
```

Compteur est la variable de contrôle. Il prend successivement les valeurs comprises entre **valeurInitiale** et **valeurFinale** avec un pas de 1 par défaut.

DÉROULEMENT DES BOUCLES POUR

La **valeurInitiale** est affectée à la variable compteur ;

On compare la valeur du **compteur** et la **valeurFinale** :

- * Si la valeur du **compteur** est $>$ à la **valeurFinale** dans le cas d'un pas positif (ou si **compteur** est $<$ à **valeurFinale** pour un cas négatif), on sort de la boucle et on continue avec l'instruction qui suit **FinPour** ;

- * Si **compteur** est \leq à la **valeurFinale** dans le cas d'un **pas** positif (ou si **compteur** est \geq à la **valeurFinale** pour un **pas** négatif), instructions seront exécutées ;

- Ensuite, la valeur de **compteur** est incrémentée de la valeur du **pas** si **pas** est positif (ou décrémenté si **pas** est négatif) ;

- On recommence l'étape : la comparaison entre **compteur** et **valeurFinale** est de nouveau effectuée, et ainsi de suite ...

EXEMPLE

Calcul de la somme des **n** premiers entiers positif.

Variables s, n, i : entier

Debut

Ecrire ("entrez la valeur n : ")

Lire (n)

s ← 0

Pour i allant de 1 à n

s ← s + i

FinPour

Ecrire (" La somme des n premiers entiers positifs
est : ", n)

Fin

EXEMPLE

Calcul de x à la puissance n où x est un réel non nul et n un entier positif ou nul.

```
Variables x, puiss : réel
              n,i : entier
Debut
    Ecrire ("entrez respectivement les
valeurs de x et n ")
    Lire (x, n)
    puiss ← 1
    Pour i allant de 1 à n
        puiss ← puiss*x
    FinPour
    Ecrire (x, " à la puissance ", n, "
est égal à ", puiss)
Fin
```

BOUCLE POUR : REMARQUE

Il faut éviter de modifier la valeur du **compteur** (et de **valeurFinale**) 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-5  
    écrire(" i= ", i)  
FinPour
```

LES ITÉRATIONS INDÉTERMINISTES

LES BOUCLES TANT QUE

- Il existe deux instructions permettant de faire des boucles indéterministes. Ces deux instructions sont **Tant que** et **Répéter ...jusque**.
- La syntaxe de l'instruction **Tant que** s'écrit :

```
Tant que condition faire  
    instructions (traitement)  
FinTantQue
```

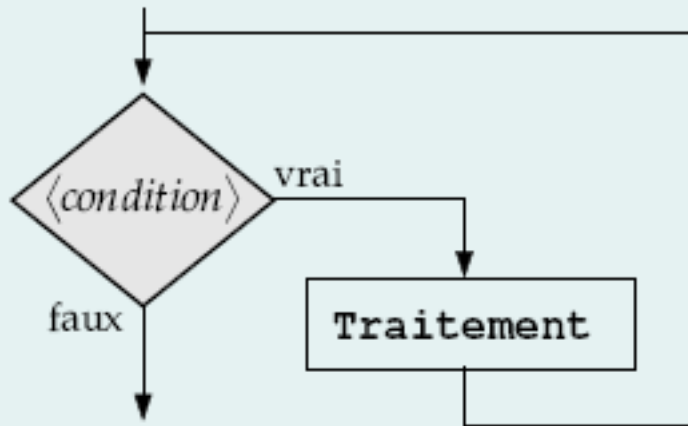
- * Qui signifie que tant que la condition est vraie on exécute les instructions;
- * Si la condition est fausse, on sort de la boucle et on exécute l'instruction qui est après **FinTantQue**;

LES BOUCLES TANT QUE

Remarque: **Traitement** peut ne jamais être exécuté.

Attention aux boucles infinies comme :

```
i ← 1  
TantQue i > 0  
    i ← i + 1  
FinTantQue
```



- 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.

EXEMPLE

On veut écrire un algorithme qui demande un nombre entier positif **n** puis calcul la somme **s** des entiers compris entre **0** et **n**

```
Variables n, s, i : Entier  
  Début  
    Ecrire ("Donner la valeur de n")  
    lire(n)  
    si n < 0 alors  
      Ecrire ("Le nombre n doit être  
positif")  
    Sinon  
      s ← 0  
      i ← 0  
      TantQue i ≤ n  
        s ← s + i  
        i ← i + 1  
      FintantQue  
      Ecrire("La somme s vaut : ", s)  
    Finsi  
  Fin
```

LES ITÉRATIONS INDÉTERMINISTES...

RÉPÉTER ...JUSQU'À

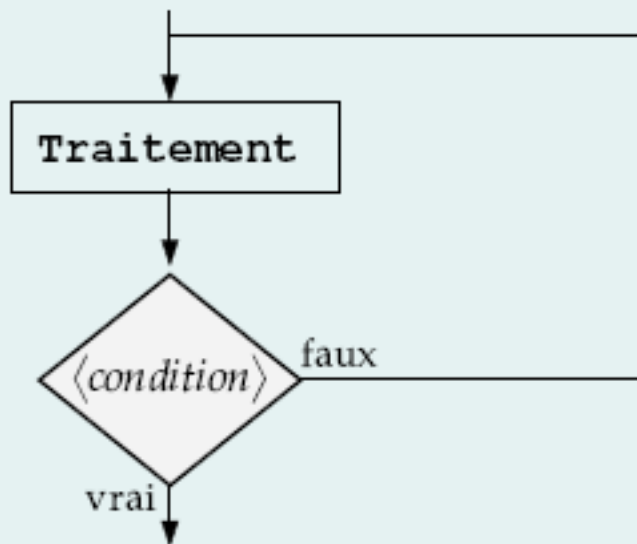
La syntaxe de l'instruction **Répéter Jusqu'à** s'écrit :

Répéter

instructions

Jusqu'à ce que condition

qui signifie que les instructions sont exécutées jusqu'à ce que la condition soit vraie (tant qu'elle est fausse)



Remarque: **Traitement** est au moins exécuté une fois.

Attention aux boucles infinies comme :

$i \leftarrow 1$

Répéter

$i \leftarrow i + 1$

Jusqu'à ce que $i < 0$

EXEMPLE

On veut écrire un algorithme qui détermine le premier nombre entier **n** tel que la somme **s** de **1** à **n** dépasse strictement : 100 (NB : Utiliser l'instruction **Répéter ... Jusqu'à**).

Variables s, i : entier

Debut

s ← 0

i ← 0

Répéter

i ← i + 1

s ← s + i

Jusqu'à (s > 100)

Ecrire ("La valeur cherchée est
n = ", i)

Fin

COMPARAISON BOUCLES

TANT QUE ET RÉPÉTER ... JUSQU'À

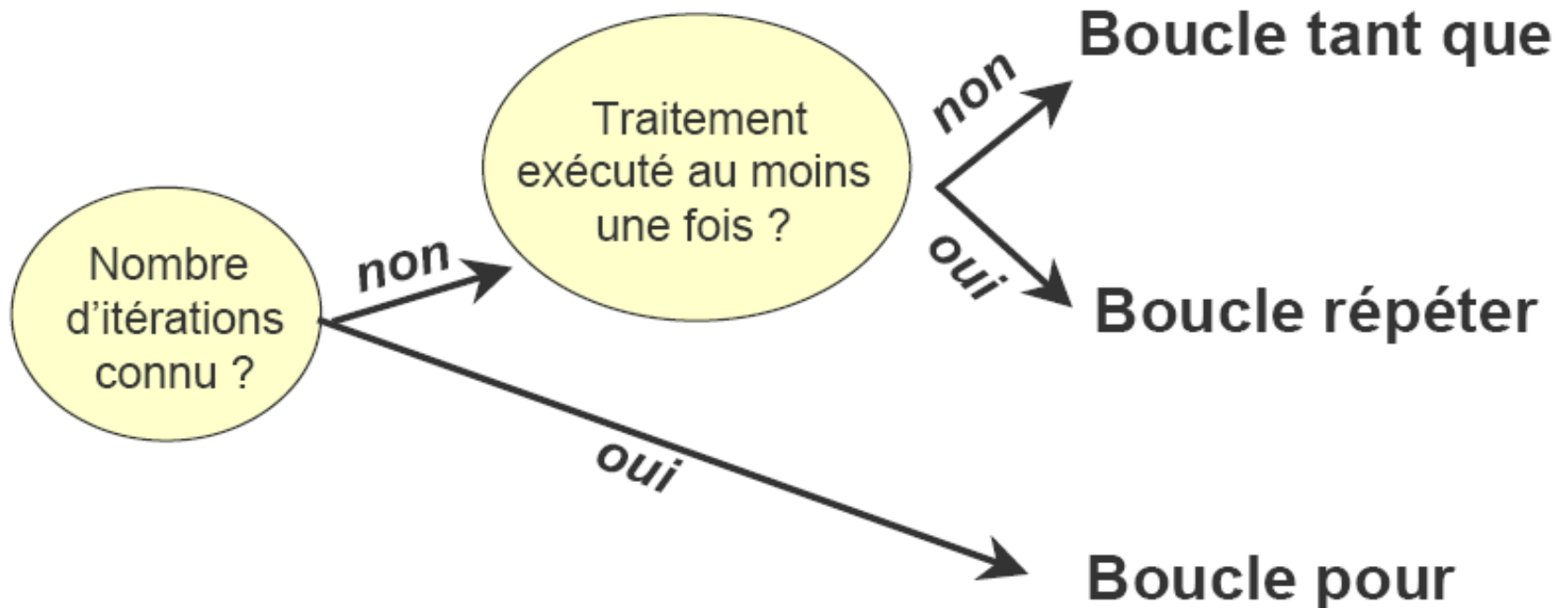
Boucle **Tant que**

- condition vérifiée **avant** chaque exécution du traitement ;
- le traitement peut donc ne pas être exécuté ;

Boucle **Répéter ... Jusqu' à**

- condition vérifiée **après** chaque exécution du traitement ;
- le traitement est exécuté au moins une fois ;

CHOISIR POUR, TANT QUE ET RÉPÉTER ... JUSQU'À



EXERCICES

Écrire un algorithme qui demande à l'utilisateur un nombre **n** compris entre **10** et **4** jusqu'à ce que la réponse convienne.

Correction

Algorithme JeuDeNombres

Variable n : **Entier**

Début

n ← 0

TantQue n < 4 ou n > 10

Ecrire ("Donner un nombre
entre 10 et 4")

Lire (n)

FinTantQue

Ecrire ("Bonne réponse.")

Fin

EXERCICES

Écrire un algorithme qui demande un nombre compris entre 10 et 20, jusqu'à ce que la réponse convienne.

En cas de réponse supérieure à 20, on fera apparaître un message : « Plus petit ! », et inversement, « Plus grand ! » si le nombre est inférieur à 10.

Correction

Variable N : Entier

Début

N ← 0

TantQue N < 10 ou N > 20

Ecrire ("Entrez un nombre
entre 10 et 20")

Lire (N)

Si N < 10 **Alors**

Ecrire ("Plus grand !")

Finsi

Si N > 20 **Alors**

Ecrire ("Plus petit !")

FinSi

FinTantQue

Fin

EXERCICES

Écrire un algorithme qui demande un nombre de départ, et qui ensuite affiche les dix nombres suivants. Par exemple, si l'utilisateur entre le nombre 17, le programme affichera les nombres de 18 à 27.

Correction

```
Algorithme NombresSuivant
Variables N, i : Entier
Début
  Ecrire ("Donner un nombre : ")
  Lire (N)
  Ecrire ("Les 10 nombres suivants
    sont : ")
    Pour i ← N + 1 à N + 10
      Ecrire (i)
    Finpour
Fin
```

EXERCICES

Écrire un algorithme qui demande un nombre de départ, et qui ensuite écrit la table de multiplication de ce nombre, présentée comme suit (cas où l'utilisateur entre le nombre 7). La table de 7 est :

7 x 1 = 7 7 x 2 = 14 7 x 3 = 21 ... 7 x 10 = 70

Correction

Algorithme TbleDeMultiplication

Variables N, i : **Entier**

Debut

Ecrire ("Entrez un nombre : ")

Lire (N)

Ecrire ("La table de multiplication
de ", N)

Pour i ← 1 à 10

Ecrire (N, " x ", i, " = ",
N*i)

Finpour

Fin

EXERCICES

Écrire un algorithme qui demande un nombre de départ, et qui calcule la somme des entiers jusqu'à ce nombre. Par exemple, si l'on entre 5, l'algorithme doit calculer : $1 + 2 + 3 + 4 + 5 = 15$.

NB : On souhaite afficher uniquement le résultat final.

Correction

Algorithme Somme

Variables N, i, Som : **Entier**

Début

Ecrire (" Donner un nombre : ")

Lire (N)

 Som \leftarrow 0

Pour i \leftarrow 1 à N

 Som \leftarrow Som + i

Finpour

Ecrire ("La somme est : ", Som)

Fin

EXERCICES

Écrire un algorithme qui demande un nombre de départ, et qui calcule sa factorielle.

NB : La factorielle de 8, notée 8 !, vaut

$1 \times 2 \times 3 \times 4 \times 5 \times 6 \times 7 \times 8$

Correction

Algorithme Factorielle

Variables N, i, F : **Entier**

Début

Ecrire ("Donner un nombre : ")

Lire (N)

 F ← 1

Pour i ← 1 **à** N

 F ← F * i

Finpour

Ecrire ("La factorielle de ", N, " est : ", F)

Fin

EXERCICES

Écrire un algorithme qui demande successivement 5 nombres à l'utilisateur, et qui lui affiche ensuite quel était le plus grand parmi ces 20 nombres :

```
Entrez le nombre numéro 1 : 12
```

```
Entrez le nombre numéro 2 : 14
```

```
etc.
```

```
Entrez le nombre numéro 20 : 6
```

```
Le nombre le plus grand était : 14
```

Modifiez ensuite l'algorithme pour que le programme affiche de surcroît en quelle position avait été saisie ce nombre :

```
Il a été saisi en position numéro : 2
```

EXERCICES

Correction

Algorithme PlusGrandNombre

Variables N, i, PG : **Entier**

Début

PG \leftarrow 0

Pour i \leftarrow 1 à 20

Ecrire ("Entrez le nombre numéro ", i, " : ")

Lire (N)

Si i = 1 ou N > PG **Alors**

 PG \leftarrow N

FinSi

Finpour

Ecrire ("Le nombre le plus grand était : ", PG)

Fin

En ligne 4, on peut mettre n'importe quelle valeur dans PG, il est nécessaire que cette variable soit affectée pour que le premier passage en ligne 8 (i.e. N > PG) ne provoque pas d'erreur.

EXERCICES

Pour la version améliorée, cela donne :

Algorithme PlusGrandNombreAmélioré

Variables N, i, PG, IPG : **Entier**

Début

PG \leftarrow 0

Pour i \leftarrow 1 à 20

Ecrire ("Entrez le nombre numéro ", i, " : ")

Lire (N)

Si i = 1 **ou** N > PG **Alors**

 PG \leftarrow N

 IPG \leftarrow i

FinSi

Finpour

Ecrire (" Le nombre le plus grand était : ", PG)

Ecrire ("Il a été saisi en position numéro ", IPG)

Fin

EXERCICES

Réécrire l'algorithme précédent, mais cette fois-ci on ne connaît pas d'avance combien l'utilisateur souhaite saisir de nombres. La saisie des nombres s'arrête lorsque l'utilisateur entre un 0.

Correction

Algorithme PlusGrandNombre

Variables N, i, PG, IPG : **Entier**

Début

N ← 1

i ← 0

PG ← 0

TantQue N <> 0

i ← i + 1

Ecrire ("Entrez le nombre numéro ", i, " : ")

Lire (N)

EXERCICES

Si $i = 1$ ou $N > PG$ **Alors**

$PG \leftarrow N$

$IPG \leftarrow i$

FinSi

FinTantQue

Ecrire ("Le nombre le plus grand était : ", PG)

Ecrire ("Il a été saisi en position numéro ", IPG)

Fin

EXERCICES

Lire la suite des prix (en Dirhams entiers et terminée par zéro) des achats d'un client. Calculer la somme qu'il doit, lire la somme qu'il paye, et simuler la remise de la monnaie en affichant les textes "10 Dirhams", "5 Dirhams" et "1 Dirham" autant de fois qu'il y a de coupures de chaque sorte à rendre.

Correction

Algorithme GestionAchat

Variables prixArticle, sommeDue, montantVerse, reste, Nb10DH,
Nb5DH, NB1DH : Entier

Début

prixArticle \leftarrow 1

sommeDue \leftarrow 0

TantQue prixArticle \neq 0

Ecrire ("Donner le prix de l'article : ")

Lire (prixArticle)

sommeDue \leftarrow sommeDue + prixArticle

FinTantQue

Ecrire ("Vous devez :", sommeDue, " Dirhams")

EXERCICES

```
Ecrire ("Montant versé :")
Lire (montantVerse)
reste ← montantVerse - sommeDue
Nb10DH ← 0
TantQue Reste >= 10
    Nb10DH ← Nb10DH + 1
    reste ← reste - 10
FinTantQue
Nb5DH ← 0
Si Reste >= 5
    Nb5DH ← 1
    reste ← reste - 5
FinSi
Nb1DH ← reste
Ecrire ("Rendu de la monnaie :")
Ecrire ("Billets de 10 DHS : ", Nb10DH)
Ecrire ("Billets de 5 DHS : ", Nb5DH)
Ecrire ("Pièces de 1 DH : ", Nb1DH)
```

Fin

TABLEAUX

Jusqu'à présent, nous avons utilisé que des variables qui prennent un emplacement unique dans la mémoire.

Par contre, pour de nombreuses applications, on souhaite regrouper plusieurs données dans une seule variable dans un ordre particulier, mais dont le type n'est pas standard.

Exemple:

On a 50 valeurs réelles (Les notes des étudiants par exemple). Pour loger ces valeurs de même types, de même nature et qui vont subir, sans doute, le même traitement, consiste à déclarer 50 variables, appelées par exemple N1, N2, N3, etc.

Moyenne $\leftarrow (N1+N2+N3+N4+N5+N6+N7+N8+N9+\dots+N48+N49+N50) / 50$

TABLEAUX

Les inconvénients de cette méthode:

Difficultés d'accéder rapidement à la note d'un élève donné (le 5^{ème} par exemple, il faut compter sur ses doigts pour déterminer la lettre qui correspond à 5) ;

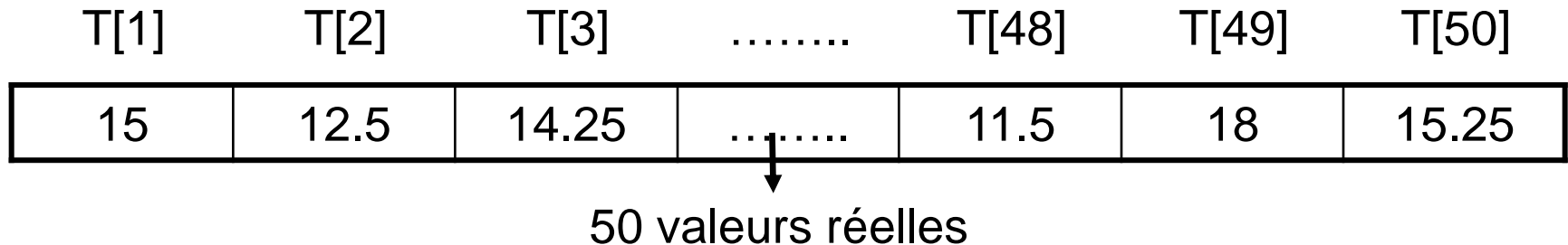
Pour lire ces 50 notes sur le clavier, il faut 50 lignes, de programmation (lecture de données) ce qui va rendre les algorithmes très longs et parfois impossibles ;

La même chose pour les opérations d'écriture de ces données sur l'écran ;

... etc.

TABLEAUX

Solution : On a la possibilité de réunir toutes ces valeurs dans une seule variable dite structurée et composée de 50 cases juxtaposées et de mêmes "dimensions" car elles vont recevoir des données de même type. Cette variable s'appelle: un tableau qu'on peut présenter par le schéma suivant:



Le type tableau est donc un type dit structuré. Il permet de stocker plusieurs éléments de même type et il permet aussi l'accès à ces éléments avec un indice.

Remarque : l'indice du tableau commence de 1 pour certains langages et de 0 pour d'autres.

TABLEAUX : DÉCLARATION D'UN TABLEAU

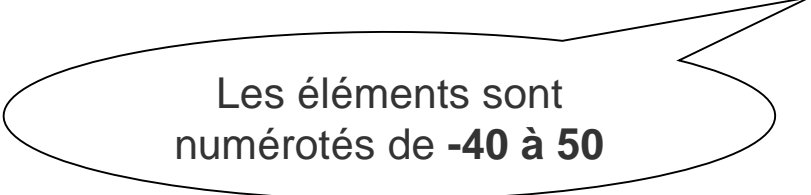
Exemple : Déclaration d'un tableau pouvant contenir jusqu'à 35 entiers:

Variable tabl : tableau[1..35] d'entiers

Où : **tabl** : nom du tableau;
tableau : mot clé;
[1..35] : indices minimale et maximale (i.e. taille);
d'entiers : type des éléments.

Autre exemple : Déclaration d'un tableau qui contiendra les fréquences des températures comprises entre -40°C et 50°C

Variable températures : tableau [-40..50] de réels



Les éléments sont
numérotés de **-40 à 50**

TABLEAUX : DÉFINITION D'UN TYPE TABLEAU

type nom = description

Exemple : Déclaration d'un nouveau type Mot, tableau de 10 caractères

```
type mot = tableau [1..10] de caractères  
variables nom, verbe : mot
```


UTILISATION D'UN TABLEAU : PAR LES INDICES

Accès en écriture :

Écrire (tabl[4])

signifie que le contenu du tableau à l'indice 4 est affiché à l'écran.

Accès en lecture :

tabl[3] ← 18

signifie que la valeur 18 est placée dans le tableau à l'indice 3.

Lire (tabl[5])

signifie que la valeur entrée par l'utilisateur est enregistrée dans le tableau à l'indice 5

Attention :

tabl ← 18

nom[2] ← 3

EXEMPLES

Ecrire un algorithme permettant d'entrer 10 valeurs réelles au clavier, les stocker dans un tableau, calculer leur somme et les afficher avec leur somme à l'écran.

Algorithme TableauSomme

Variables V : tableau[1..10] de réels;
 S : Réel
 i : Entier

Début

Pour i allant de 1 à 10 **faire**
 Ecrire ("Entrer l'élément ", i)
 Lire (V[i])
 FinPour

TABLEAUX EXEMPLES

$S \leftarrow 0$

Pour i allant de 1 à 10 **faire**

$S \leftarrow S + V[i]$

FinPour

Pour i allant de 1 à 10 **faire**

Ecrire("l'élément ", i , " est : ", $V[i]$)

FinPour

Ecrire("la somme des éléments du tableau est ", S)

Fin

TABLEAUX À DEUX DIMENSIONS

Définition : Le type de base d'un tableau à une dimension est quelconque, il peut être en particulier de type tableau. On parlera alors d'un tableau. Ce dernier peut se déclarer de la manière suivante:

Exemple : Déclaration d'un tableau à deux dimensions contenant **n** lignes et **m** colonnes de réels:

Variable tabl : tableau [1..n] de tableau [1..m] de réels

Où : **tabl** : la variable tableau à deux dimensions,
[1..n] : les indices de la première dimension (lignes),
[1..m] : les indices de la deuxième dimension (colonnes).
de réels : type de base des éléments du tableau.

Pour simplifier, cette déclaration peut se faire de la manière suivante :

Variable tabl : tableau [1..n , 1..m] de réels

TABLEAUX À DEUX DIMENSIONS

Exemple de déclaration

Variable points : tableau [1..2 , 1..7] d'entiers

	1	2	3	4	5	6	7
1	10	3	25	14	2	1	8
2	9	20	7	12	2	4	7

tableau à 2 lignes et 7 colonnes

Accès en écriture : **Écrire (points[1,4])**

signifie que la valeur contenue en ligne 1 colonne 4 du tableau est affichée à l'écran!.

Accès en écriture : **points[2,7] ← 6**

signifie que la valeur 6 est placée dans le tableau en ligne 2 colonne 7.

EXAMPLE

Ecrire un algorithme permettant de saisir les données d'un tableau à deux dimension (6, 7), de faire leur somme, produit et moyenne et de les afficher avec les résultats de calcul à l'écran.

Algorithme TableauDeuxDimensions

Variables réels T : tableau[1..6 , 1..7] de

S, P, M : réels

i, j : entiers

Début

Pour i allant de 1 à 6 faire

Pour j allant de 1 à 7 faire

```

Ecrire("Entrer l'élément", i, " ", j, " ")
T["", i, " ", " ", j, ""] = " "

```

Lire ($T[i, j]$)

FinPour

FinPour

$$S \leftarrow 0$$
$$P \leftarrow 1$$

EXEMPLE

```
Pour i allant de 1 à 6 faire
  Pour j allant de 1 à 7 faire
    S ← S + T[i,j]
    P ← P * T[i,j]
  FinPour
FinPour
M ← S/40
Pour i allant de 1 à 6 faire
  Pour j allant de 1 à 7 faire
    Ecrire("L'élément T[",i,"",j,""]="",T[i,j])
  FinPour
FinPour
Ecrire("La somme des éléments du tableau : ",S)
Ecrire("Le produit des éléments du tableau : ",P)
Ecrire("La moyenne des éléments du tableau : ",M)
Fin
```

SOUS-ALGORITHMES

- Considérons un algorithme de calcul du nombre de combinaisons d'un nombre P dans un nombre N entiers: CNP donné par la formule suivante :

$$CNP = N! / P!(N-P)!$$

Où: $N!$ est la factorielle de N :

$$N! = 1*2*3*....*N$$

Sous-algorithmes

Algorithme: Calcul_CNP;

Variables

N, P, FN, FP, CNP, i: entiers;

Début

Lire(N,P);

(*Calcul de la factorielle de N*)

FN ← 1;

Pour i allant de 1 à N Faire

FN ← FN*i;

Fin Pour ;

(*Calcul de la factorielle de P*)

FP ← 1;

Pour i allant de 1 à P Faire

FP ← FP*i;

FinPour i;

(*Calcul de la factorielle de N-P*)

FNP ← 1;

Pour i allant de 1 à N-P Faire

FNP ← FNP*i;

FinPour i;

CNP ← FN/(FP*FNP);

Ecrire(CNP);

Fin

Sous-algorithmes

Cet algorithme n'est pas bien structuré, car il contient des répétitions.

On peut vider cet algorithme de ces répétitions en plaçant les instructions répétées dans un sous programme de calcul de la factorielle d'un nombre quelconque qu'on appellera à chaque fois qu'on a besoin (ici 3 fois).

Ceci nous permettra d'obtenir un programme modulaire, moins encombrant et facile à lire.

Programme principale

1) Appel de la Factorielle de N

2) Appel de la Factorielle de P

3) Appel de la Factorielle de N-P



Factorielle de X

■ L'algorithme de calcul de CNP après transformation peut se présenter ainsi:

**Sous Algorithme Factorielle(X:
entier);**
Variable
 FX, i: Entiers;
Début
 (/*Calcul de la factorielle de X*/)
 FX←1;
 Pour i allant de 1 à X faire
 FX ← FX*i;
 FinPour i;
 Retourner (FX)
FIN

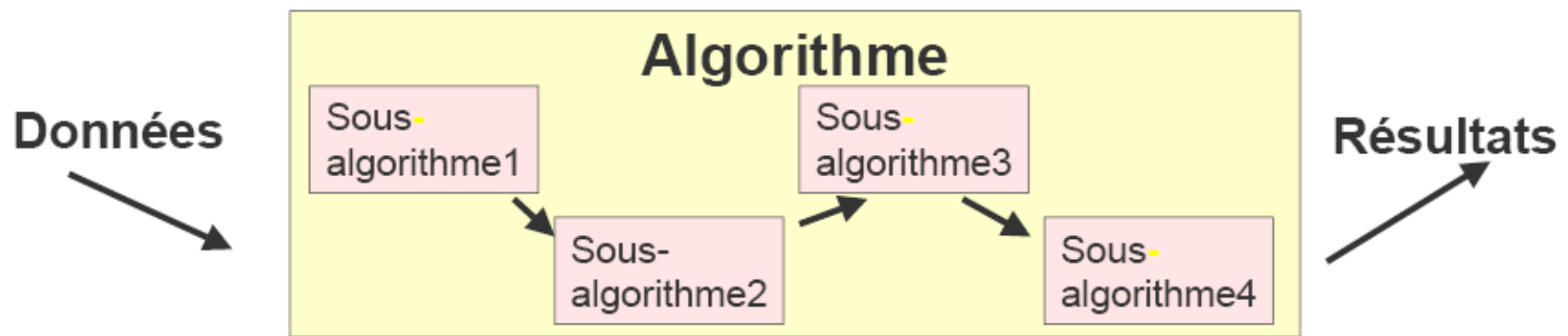
Algorithme principale
Algorithme Calcul_CNP;
Variable
 N, P, FN, FP, CNP, i: entiers;
Début
 Lire(N,P);
 FN← Factorielle(N);
 FP← Factorielle(P);
 FNP← Factorielle(N-P);
 CNP ← FN/(FP*FNP);
 Ecrire(CNP);
FIN

Les sous - Algorithmes

■ Définition.

un sous Algorithme est une partie d'un algorithme qui porte un nom spécifique donné et qui peut être appelé par ce nom, selon les besoins, pour exécuter une tâche bien déterminée.

Algorithme complexe et long ➡ Sous Algorithmes



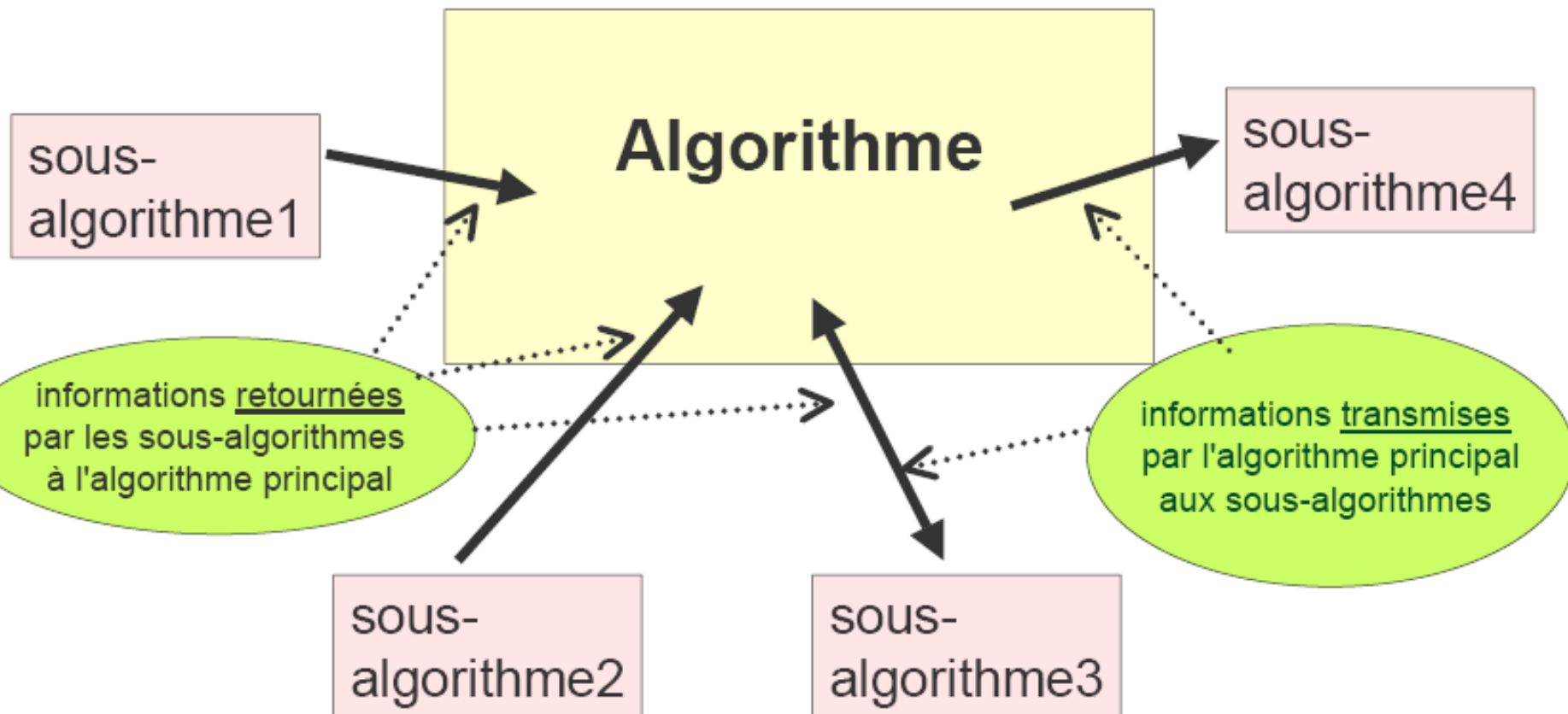
Un algorithme **appelle** un sous-algorithme : cet algorithme *passé* "momentanément" le contrôle de l'exécution du traitement au sous-algorithme.

Un sous-algorithme est conçu pour faire un traitement bien **défini**, bien **délimité**, si possible *indépendamment* du contexte particulier de l'algorithme appelant.

Remarque : un sous-algorithme peut en appeler un autre.

Les sous – Algorithmes

Comment les informations circulent...



Les sous – Algorithmes

- Donc écrire un Algorithme qui résout un problème revient toujours à écrire des sous-algorithmes qui résolvent des sous parties du problème initial
- En algorithmique il existe deux types de sous-Algorithmes :
 - Les fonctions
 - Les procédures
- Un sous-Algorithme est obligatoirement caractérisé par un nom (un identifiant) unique
- Lorsqu'un sous Algorithme a été explicité (on a donné l'algorithme), son nom devient une nouvelle instruction, qui peut être utilisé dans d'autres (sous-) Algorithmes
- Le (sous-)programme qui utilise un sous-Algorithme est appelé (**sous-**) **Algorithme appelant**

Les sous – Algorithmes

- Nous savons maintenant que les variables, les constantes, les types définis par l'utilisateur et que les sous-Algorithmes possèdent un nom
- Ces noms doivent suivre certaines règles :
 - Ils doivent être explicites (à part quelques cas particuliers, comme par exemple les variables i et j pour les boucles)
 - Ils ne peuvent contenir que des lettres et des chiffres
 - Ils commencent obligatoirement par une lettre
 - Les types commencent toujours par une majuscule
 - Les constantes ne sont composées que de majuscules
 - Lorsqu'ils sont composés de plusieurs mots, on utilise les majuscules (sauf pour les constantes) pour séparer les mots (par exemple JourDeLaSemaine)

Les différents types de variable...

- **Définitions :**

La **portée** d'une variable est l'ensemble des sous-Algorithmes où cette variable est connue (les instructions de ces sous-Algorithmes peuvent utiliser cette variable)

- Une variable définie au niveau d'un Algorithme principal (celui qui résout le problème initial, le problème de plus haut niveau) est appelée **variable globale**
 - Sa portée est totale : **tout** sous-Algorithme d'un Algorithme principal peut utiliser cette variable
- Une variable définie au sein d'un sous Algorithme est appelée **variable locale**
 - La portée d'un variable locale est uniquement le sous-Algorithme qui la déclare
- Lorsque le nom d'une variable locale est identique à une variable globale, la variable globale est localement masquée
- Dans ce sous-Algorithme la variable globale devient inaccessible

Structure d'un Algorithme...

- Un Algorithme doit suivre la structure suivante :

Algorithme *nom du Algorithme*

Définition des constantes

Définition des types

Déclaration des variables globales

Définition des sous-Algorithmes

début

instructions du Algorithme principal

fin

Les sous – Algorithme (suite...)

En programmation, on distingue deux types de sous-algorithmes:

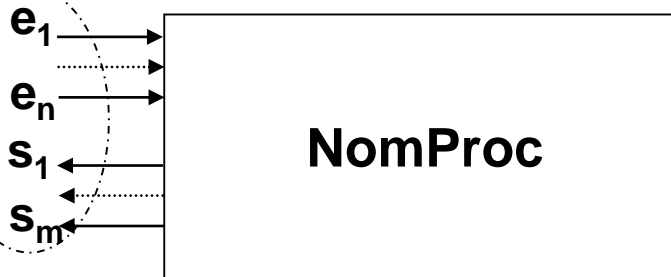
- Les fonctions qui renvoient des valeurs résultats et qui peuvent faire partie d'une expression
- Les procédures qui ne renvoient pas directement des valeurs résultats et qui ne peuvent pas faire partie d'une expression, mais qui sont appelées pour réaliser des tâches précises.

Les procédures

- Une procédure est un sous algorithme qui doit être déclaré avant toute utilisation. Cette déclaration comporte le nom de la procédure (un identificateur), la liste de ses paramètres de travail ainsi que l'ensemble des instructions qu'elle utilise.
- **Définition:**

Procédure **NomProc**($e_1 : \text{Type_}e_1 \dots e_n : \text{Type_}e_n$) ($s_1 : \text{Type_}s_1 \dots s_m : \text{Type_}s_m$)

**Paramètres
formels
(Définitions)**



Cas particuliers:

- Procédure $P1()(); (*\text{Ni paramètres d'entrée ni de sortie}*)$
 - Exemple : procédure qui efface l'écran: CLRSCR
- Procédure $P1(e_1 : \text{Type_}e_1)(); (*\text{pas de paramètres de sortie}*)$
 - Exemple : procédure d'écriture d'une donnée: Ecrire (x)
- Procédure $P3() (s_1 : \text{Type_}s_1) (*\text{pas de paramètres d'entrée}*)$
 - Exemple : procédure de lecture d'une donnée: Lire (x)

Exemple 1:

- Cas général.
 - Une procédure qui calcule la somme de deux nombres réels: a et b. cette somme est récupérée dans un paramètre de sortie: S.

Procédure Somme(a,b: réels)(S:réel);

 Début

$S \leftarrow a + b$;

 Fin



Appel d'une procédure

X

NomProc(e'_1, \dots, e'_n) (s'_1, \dots, s'_m)

Les paramètres e'_i et s'_j sont appelés: **paramètres réels** ou **paramètres d'appel** car ils contiennent les valeurs réelles à transmettre aux paramètres formels de la procédure.

Algorithme Calcul_somme;

Variables

X, Y, Z: réels

Début

Lire (X)

Lire (Y)

Somme (X, Y)(Z)

Écrire ("la somme de ces deux nbres est : ", Z)

Fin

Les différents types de variable...

Variables
Globales à tout
le programme

Algorithme Calcul_CNP;
Variable

N, P, FN, FP, CNP, i: entiers;

Début

Lire(N,P);

FN ← Factorielle(N)(FN);

FP ← Factorielle(P)(FP);

FNP ← Factorielle(N-P)(FNP);

CNP ← FN/(FP*FNP);

Écrire(CNP);

FIN

nom de la procédure

Procédure Factorielle(X: entier)(FX: entier);

Variable

i: Entiers;

liste des
paramètres

variables locales
à la procédure

Début

(*Calcul de la factorielle de X*)

FX ← 1;

Pour i ← 1 à X Faire
 FX ← FX*i;

FinPour i;

FIN

Exemple

- Ecrire un Algorithme permettant de remplir un tableau puis l'afficher en utilisant les procedures

Règle de la variable globale

- Un bon algorithme (Programme) doit contenir le minimum possible de variables globales.
- En fait , une fois déclarées, les variables globales occupent leurs places mémoire du début jusqu'à la fin de l'exécution du programme. Alors que les variables locales ne prennent leurs places dans la mémoire qu'au moment de l'appel du sous-Algorithme. Une fois l'exécution de ce dernier est finie, ces variables disparaissent.
- **Règle:**
Une variable ne doit être déclarée globale que si elle intervient dans au moins deux sous algorithmes différents non successifs et indépendants.
 - **Successifs:** veut dire qu'un sous algorithme peut s'exécuter à partir d'un autre sous- algorithme. Dans ce cas, le premier peut transmettre la valeur de la variable aux deuxième comme paramètre d'entrée.
 - **Indépendants:** veut dire que les sous algorithmes sont déclarés séparément. Si un sous algorithme est déclaré à l'intérieur d'un autre sous algorithme (chose possible dans certains langages de prog), la variable peut être déclarée locale dans le premier et peut s'utiliser automatiquement dans le deuxième.

Transmission de données à une procédure ...

- **Règle d'appel d'un sous algorithme.**

Lors de l'appel d'un sous algorithme, les valeurs (ou les adresses) des paramètres réels (e'_i , s'_j) sont transférées aux paramètres formels (e_i , s_j) et les paramètres formels deviennent des variables locales au sous algorithme.

- **Étude de la transmission de données.**

➤ Reprenons l'exemple de la procédure permettant de calculer la factorielle d'un nombre entier:

Procédure Factorielle(N: entier)(FN: entier);

Variable

i: Entiers;

Début

(*Calcul de la factorielle de X*)

FX ← 1;

Pour i ← 1 à N Faire

FN ← FN*i;

FinPour i;

FIN

Étude de la transmission de données

Procédure Factorielle(X: entier)(FX: entier);

Variable

i: Entiers;

Début

(*Calcul de la factorielle de X*)

FX ← 1;

Pour i ← 1 à X Faire

FX ← FX*i;

FinPour i;

FIN

Les différents appels de cette procédure peuvent se présenter comme suit :

- **Factorielle (4)(Z) : Appel Possible** (paramètre d'entrée est cste, celui de sortie est une var déclarée)
- **Factorielle (X)(Z): Appel Possible** (paramètre d'entrée est variable déclarée, celui de sortie est une var déclarée)
- **Factorielle (3*X+2Y-1) (Z): Appel possible** (paramètre d'entrée est une expression, celui de sortie est var déclarée)
- **Factorielle (X) (6): Appel impossible** (paramètre d'entrée est variable déclarée, celui de sortie est une constante)
- **Factorielle (X) (Y+Z): Appel impossible**(paramètre d'entrée est variable déclarée, celui de sortie est une expression)

Conclusion:

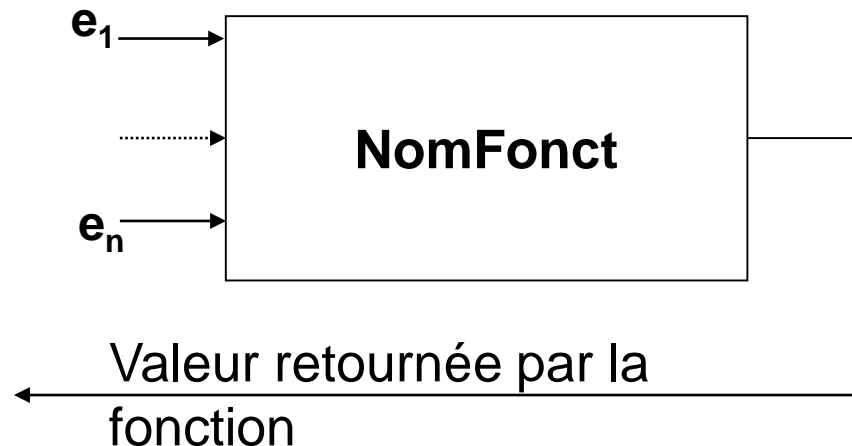
Un paramètre de sortie doit être une variable déclarée qui a un emplacement dans la mémoire de données et par conséquent, qui a une adresse.

Les Fonctions....

- Comme pour les procédures, une fonction est un sous programme qui doit être déclaré avant toute utilisation.

- **Définition:**

Fonction **NomFonct**(e_1 : Type_ e_1 ... e_n : Type_ e_n): type_de_retours



On rajoute donc à la définition d'une procédure, type de la valeur retournée par la fonction.

Les Fonctions....

- Les fonctions sont des sous-programmes admettant des paramètres et retournant un **seul** résultat (comme les fonctions mathématiques $y=f(x,y,\dots)$)
 - une fonction possède un seul type, qui est le type de la valeur retournée
 - le passage de paramètre est **uniquement en entrée** : c'est pour cela qu'il n'est pas précisé
 - lors de l'appel, on peut donc utiliser comme paramètre des variables, des constantes mais aussi des résultats de fonction
 - la valeur de retour est spécifiée par l'instruction **retour**

Les Fonctions....

- On déclare une fonction de la façon suivante :

fonction *nom de la fonction (paramètre(s) de la fonction) : type de la valeur retournée*

Déclaration *variable locale l : type l; ...*

début

*instructions de la fonction avec au moins une fois l'instruction **retour(une valeur)***

fin

- On utilise une fonction en précisant son nom suivi des paramètres entre parenthèses
- Les parenthèses sont toujours présentes même lorsqu'il n'y a pas de paramètre

Les Fonctions....

- Exemple

une fonction qui calcule la somme de deux nombres réels: a et b.

Fonction Somme (a,b: réels)(): réel;

Variables

Début

Retour (a+b);

Fin

Les Fonctions....

- **Algorithme Valeur_Absolue**

Variables

a : Entier, b : Naturel

fonction abs (unEntier : Entier)(): Naturel

Varibales

valeurAbsolue : Naturel

Début

si unEntier ≥ 0 alors

valeurAbsolue \leftarrow unEntier

sinon

valeurAbsolue \leftarrow -unEntier

finsi

retour(valeurAbsolue)

fin

Début

écrire("Entrez un entier :")

lire(a)

b \leftarrow abs(a)

écrire("la valeur absolue de ",a," est ",b)

Fin

Exercice

- Écrire un algorithme permettant de calculer le maximum de deux nombres à travers une fonction

- **Algorithme Calcul_Max**

Variables

X, Y, Z : Réels;

fonction Maxdedeuxnombre (Nb1, Nb2 : Réels) : Réel

Varibales

M : Réel

Début

si Nb1 \geq Nb2 **alors**

M \leftarrow Nb1

sinon

M \leftarrow Nb2

finsi

retour(M)

fin

Début

écrire("Entrez le premier nombre :")

lire(X)

écrire("Entrez le 2^{ème} nombre :")

lire(Y)

Z \leftarrow **Maxdedeuxnombre** (X, Y)

écrire("le maximum de ces deux nombres est : ", Z)

Fin