Partie I : Généralités et algorithmique de base

SOMMAIRE

Chapitre 2 : Algorithmique de base

Algorithmique de base

I. Introduction

1. Notion d'algorithme

Dans la vie courante, un algorithme peut prendre la forme :

- d'une recette de cuisine;
- d'un itinéraire routier ;
- d'un mode d'emploi, etc…

Une recette de cuisine, par exemple, est un algorithme : à partir des ingrédients, elle explique comment parvenir au plat. De même, un itinéraire routier explique comment, à partir d'une position initiale, rejoindre une position finale en un certain nombre d'étapes. Etc.

Exemple : Préparer la pâte à tarte :

Les ingrédients :

250 g de farine ;

50 g de beurre ;

Un verre de lait.

Les actions élémentaires à réaliser

Début

Incorporer le beurre dans la farine

Pétrir le mélange jusqu'à ce qu'il soit homogène

Ajouter du lait

Mélanger

Si la pâte est trop sèche, alors ajouter du lait

Si la pâte à une bonne consistance, alors la laisser reposer ½ heures

Faire cuire la pâte

Fin

Un algorithme sert à transmettre un savoir faire. Il décrit les étapes à suivre pour réaliser un travail. Tout comme le savoir-faire du cuisinier se transmet sous la forme d'une recette, celui d'un informaticien se transmet sous la forme d'un algorithme.

2. *Définition d'un algorithme*

Le mot « algorithme » provient de la forme latine (Algorismus) du nom du mathématicien arabe AL KHWARIZMI. Il tenta une première définition :

«Un algorithme est une séquence d'opérations visant à la résolution d'un problème en un temps fini. »

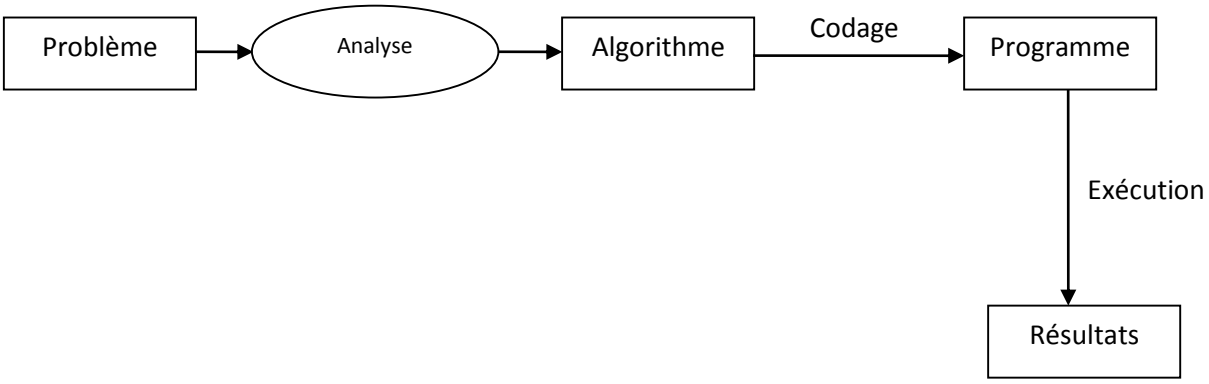
Définition : Un algorithme est une suite d'actions élémentaires qu'il faut appliquer sur des données primaires pour résoudre un problème et aboutir aux résultats.

3. *Algorithmique et programmation*

Tout problème à programmer doit être résolu d'abord sous forme d'algorithme, puis converti en programme par la traduction de l'algorithme dans le langage de votre choix.(Pascal, C, Java, Python …).

En effet, un algorithme est indépendant du langage de programmation à utiliser.

Un programme est un enchaînement d'instructions, écrit dans un langage de programmation, exécutée par un ordinateur, permettant de traiter un problème et de renvoyer des résultats. Il représente la traduction d'un algorithme à l'aide d'un langage de programmation.



Processus de développement d'un programme

4. *Analyse d'un problème*

L'étape d'analyse d'un problème sert à extraire 3 composantes :

- Les données d'entré : C'est les données indispensable à connaître pour pourvoir résoudre le problème.
- Les données de sortie : C'est les résultats recherchés.
- Le traitement : L'ensemble des opérations à appliquer sur les données d'entré pour aboutir aux résultats.

Exemple :

Problème : Calculer le montant TTC à payer pour un lot de PC portable sachant le taux de TVA est de 20%.

Travail à faire : Analyser le problème

- Les données d'entré :

Données	Identificateur	Type	Catégorie
Prix unitaire HT d'un PC	PU	Réel	Variable
Nombre de PC	Nb	Entier	Variable
Taux de TVA	TVA	Réel	Constante

- Les données de sortie :

Données	Identificateur	Type	Catégorie
Montant TTC	MTTC	Réel	Variable

- Traitement :

MHT = PU * Nb

MTVA = MHT * TVA

MTTC = MHT + MTVA

Exercice :

Problème : Calculer la surface totale d'un cylindre

Travail à faire : Analyser le problème

- Les données d'entré :

Données	Identificateur	Type	Catégorie
Rayon de la base	R	Réel	Variable
La hauteur	H	Réel	Variable
La constante Π	Pi	Réel	Constante

- Les données de sortie :

Données	Identificateur	Type	Catégorie
La surface totale	ST	Réel	Variable

- Traitement :

SB = Pi*R*R

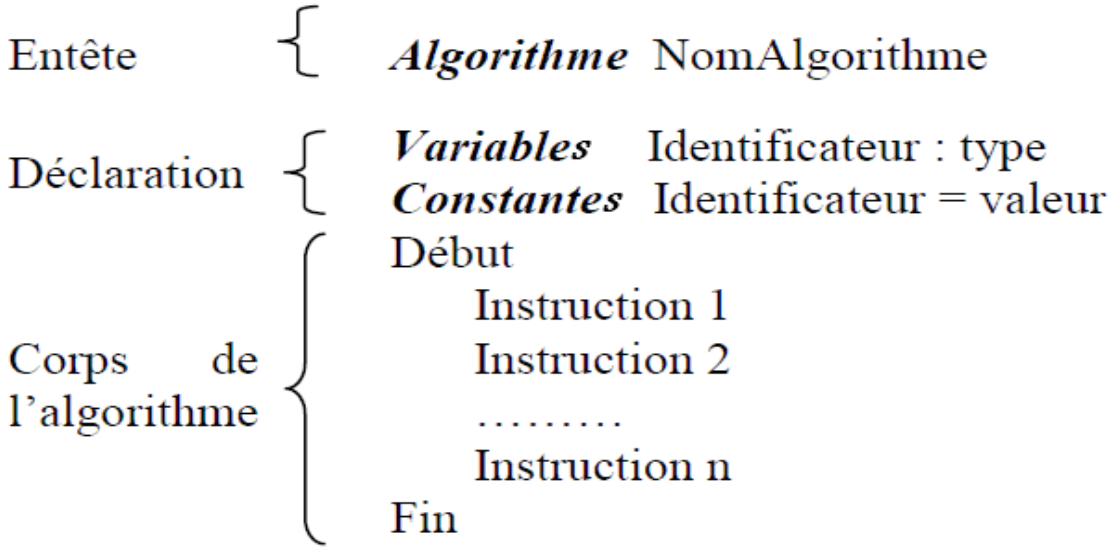
SL = 2*Pi*R*H

ST = 2*SB + SL

5. *Structure générale d'un algorithme*

On va adopter un structure pour l'écriture d'un algorithme, cette structure ne représente pas un standard mais son objectif c'est d'avoir une meilleur organisation des composants de l'algorithme.

La structure générale d'un algorithme se compose de trois parties :



La partie entête :

Cette partie commence par le mot Algorithme et le nom de l'algorithme, ce dernier doit donner une idée sur l'objet de l'algorithme.

Exemple : Pour un algorithme qui fait le calcul de la moyen des notes d'une classe, on peut lui donner comme nom (Moyen ; MoyenNote ; Moyen_Note ; …).

La partie déclaration :

Dans cette partie on doit déclarer l'ensemble des données dont on aura besoin dans l'algorithme

La partie Corps :

Contient l'ensemble des instructions (ordres) à exécuter pour pouvoir résoudre le problème.

II. Eléments de base d'un algorithme

1. Variables

L'essentiel du travail effectué par un programme d'ordinateur consiste à manipuler des données. Ces données peuvent être très diverses, mais dans la mémoire de l'ordinateur elles se ramènent toujours en définitive à une suite finie de nombres binaires.

Pour pouvoir accéder aux données, le programme (quel que soit le langage dans lequel il est écrit) fait usage d'un grand nombre de variables de différents types. Une VARIABLE possède:

- Un nom : Un variable apparaît en programmation sous un nom de variable.
- Un type : Pour distinguer les uns des autres les divers contenus possibles, on utilise différents types de variables (entiers, réels, chaîne de caractères …).
- Une valeur : Une fois la variable définie, une valeur lui est assignée, ou affectée.
- Une adresse : C'est l'emplacement dans la mémoire de l'ordinateur, où est stockée la valeur de la variable.

2. Types simples (entier, flottant, caractère)

Type Entier : pour manipuler les nombres entiers positifs ou négatifs. Par exemple : 5, -15, etc.

Type Réel : pour manipuler les nombres à virgule. Par exemple : 3.14, -15.5, etc.

Type Caractère : pour manipuler des caractères alphabétiques et numériques. Par exemple : 'a', 'A', 'z', ' ?', '1', '2' etc.

Type Chaîne : pour manipuler des chaînes de caractères permettant de représenter des mots ou des phrases. Par exemple : "bonjour, Monsieur"

Type Booléen : pour les expressions logiques. Il n'y a que deux valeurs booléennes : vrai et faux.

A un type donné, correspond un ensemble d'opérations définies pour ce type :

Type	Opérations possibles	Symbole ou mot clé correspondant
Entier	Addition Soustraction Multiplication Division Division Entière Modulo Exposant (puissance) Comparaisons	+ - * / DIV MOD ^ <, =, >, <=, >=, <>
Réel	Addition Soustraction Multiplication Division Exposant Comparaisons	+ - * / ^ <, =, >, <=, >=, <>

Caractère	Comparaisons	<, =, >, <=, >=, <>
Chaîne	Concaténation	<, =, >, <=, >=, <> +
Booléen	Comparaison Logiques	<, =, >, <=, >=, <> ET, OU, NON

Exemple :

5 / 2 = 2.5
5 Div 2 = 2
5 Mod 2 = 1
5 ^ 2 = 25
"Bonjour" + " " + "Monsieur" = "Bonjour Monsieur"

3. Affectation

L'affectation permet d'affecter une valeur à une variable. Physiquement permet de stocker une valeur dans une variable.

Symbolisée en algorithmique par "□", elle précise le sens de l'affectation.

Syntaxe : Variable □ Expression

Expression peut être soit :

- Identificateur
- Constante
- Expression arithmétique
- Expression logique

Sémantique :

Une affectation peut être définie en deux étapes :

- Evaluation de l'expression qui se trouve dans la partie droite de l'affectation
- Placement de cette valeur dans la variable.

Exemple :

```
0   Algorithme Calcul
1   Variables A, B, C, D : entier
2   Debut
3       A ← 10
4       B ← 30
5       C ← A+B
6       D ← C*A
7       C ← A+C
8       D ← C*A
9   Fin
```

Note : Les lignes sont numérotées pour faciliter l'explication.

Nous pouvons expliquer ce qui ce passe par le tableau suivant :

	Variable			
N° de ligne	A	B	C	D
0	?	?	?	?
1	?	?	?	?
2	?	?	?	?
3	10	?	?	?
4	10	30	?	?
5	10	30	40	?
6	10	30	40	400
7	10	30	50	400
8	10	30	50	500

Remarque :

Les variables numériques ne sont pas forcément initialisées à zéro. Leur valeur peut être n'importe quoi. C'est pour cette raison que nous avons mis un point d'interrogation avant qu'une première valeur ne soit affectée à la variable.

4. Entrées / sorties standards et fonctions de la bibliothèque

L'instruction d'entrée :

L'instruction d'entrée ou de lecture permet à l'utilisateur de saisir des données au clavier pour qu'elles soient utilisées par l'algorithme.

Cette instruction assigne (affecte) une valeur entrée au clavier dans une variable.

Syntaxe : Lire (identificateur)

Exemple :

Lire (A) **:** Cette instruction permet à l'utilisateur de saisir une valeur au clavier qui sera affectée à la variable A.

Remarque :

Lorsque le programme rencontre cette instruction, **l'exécution s'interrompt** et attend que l'utilisateur tape une valeur. Cette valeur est rangée en mémoire dans la variable désignée.

L'instruction de sortie :

L'instruction de sortie (d'écriture) permet d'afficher des informations à l'utilisateur à travers l'écran.

Syntaxe : Ecrire (expression)

Expression peut être une valeur, un résultat, un message, le contenu d'une variable...

Exemple :

Ecrire (A) : Cette instruction permet d'afficher à l'écran la valeur de la variable A.

Remarque : On utilise ECRIRE lorsque l'ordinateur doit afficher une information à l'utilisateur, et LIRE lorsque l'utilisateur doit communiquer une information à l'ordinateur.

Exercice d'application:

Ecrire un algorithme qui permet de saisir le prix HT (PHT) d'un article et de calculer son prix total TTC (PTTC). TVA=20%.

Solution :

Les données d'entrée :
    Prix hors taxe : PHT
    Taux de TVA : TVA
Les données de sortie :
    Prix TTC : PTTC
Le traitement :
    PTTC = PHT + (PHT * TVA)

```
0   Algorithme Calcul_PTTC
1   Variables PHT, PTTC : réel
2   Constante TVA ← 0,2
3   Début
4       Ecrire ("Entrez le prix hors taxes :")
5       Lire (PHT)
6       PTTC ← PHT + (PHT * TVA)
7       Ecrire ("Le prix TTC est ", PTTC)
8   Fin
```

Explication de l'algorithme :

N° de ligne	Explication
0	Déclare un algorithme dont le nom est Calcul_PTTC
1	Déclare les différentes variables utilisées par l'algorithme
2	Déclare la constante TVA
3	Marque le début des traitements effectués par l'algorithme
4	Affiche à l'écran le message : « Entrez le prix hors taxes : »
5	Permet à l'utilisateur de saisir une valeur au clavier qui sera affectée à la variable PHT.
6	Calcul le prix TTC et affecte le résultat à la variable PTTC
7	Affiche le résultat à l'écran
8	Marque la fin de l'algorithme

III. Structures de contrôle

Après avoir précisé la notion d'instruction, nous décrivons les différentes structures de contrôle de base disponibles en algorithmique. Ces structures permettent de contrôler l'ordre dans lequel seront effectuées les instructions.

1. Les structures alternatives (de choix)

Souvent les problèmes nécessitent l'étude de plusieurs situations qui ne peuvent pas être traitées par les séquences d'actions simples. Puisqu'on a plusieurs situations, et qu'avant l'exécution, on ne sait pas à quel cas de figure on aura à exécuter, dans l'algorithme on doit prévoir tous les cas possibles.

Il n'y a que **deux formes possibles** pour ce type des instructions ; la première est la plus simple, la seconde la plus complexe.

a) L'instruction Si … Alors … Finsi :

Syntaxe :

Si condition **Alors**

Traitement

FinSi

Le traitement sera exécuté si et seulement si la condition est vérifiée (valeur booléennes Vrai).

Exemple:

Ecrire un algorithme qui permet de calculer la valeur absolue d'un nombre réel.

$$\begin{cases} |X| = X & \text{si } X > 0 \\ |X| = -X & \text{si } X < 0 \end{cases}$$

```
Algorithme valeur_absolue1
Variables  X : réel
Début
        Lire(X)
        Si X<0 Alors
                X ← -X
        Finsi
        Ecrire ("la valeur absolue est : ", X)
Fin
```

b) L'instruction Si … Alors … Sinon … Finsi :

Syntaxe :

Si condition **Alors**

Traitement 1

Sinon

Traitement 2

Finsi

Le Traitement N°1 sera exécuté si et seulement si la condition est vérifiée (valeur booléennes Vrai) si la condition est fausse c'est le traitement N° 2 qui va être exécuté.

Exemple:

Ecrire un algorithme qui permet d'afficher la valeur absolue de la différence entre deux nombres réels.

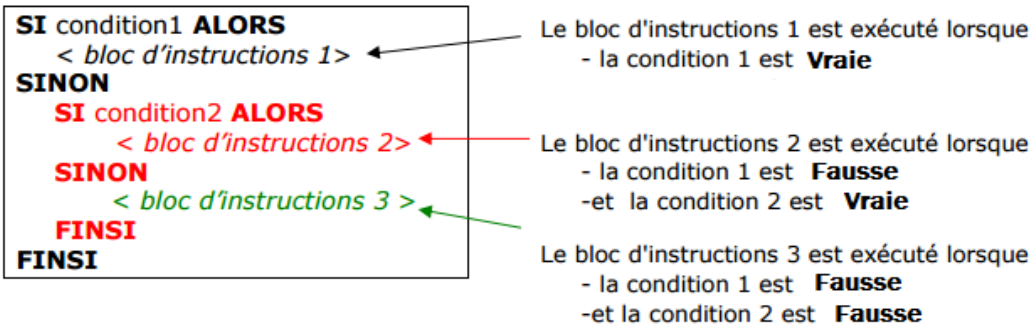
$$\begin{cases} |X-Y| = X-Y & \text{si } X>Y \\ |X-Y| = -(X-Y) = Y-X & \text{si } X<Y \end{cases}$$

```
Algorithme valeur_absolue2
Variables X, Y, Z : réels
Début
        Ecrire("X=")
        Lire(X)
        Ecrire("Y=")
        Lire(Y)
        Si X >Y Alors
                        Z ← X-Y
        Sinon
                        Z ← Y-X
        Finsi
        Ecrire ("la valeur absolue de la différence est : ", Z)
Fin
```

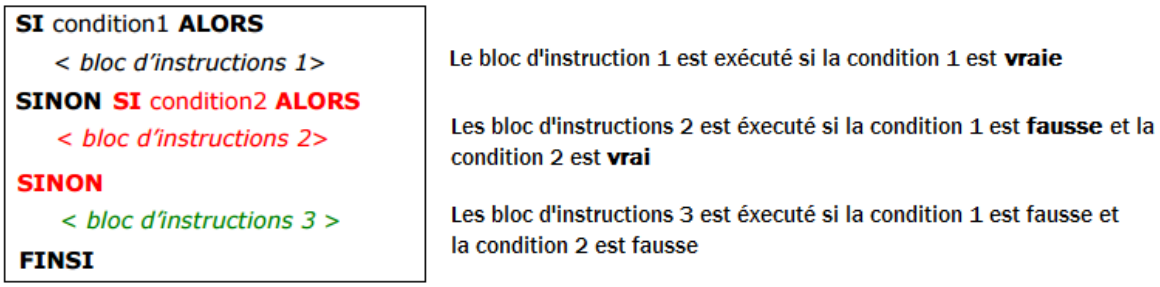
c) Les structures alternatives imbriquées :

Il peut arriver que l'une des parties d'une structure alternative contienne à son tour une structure alternative. Dans ce cas, on dit qu'on a des structures alternatives imbriquées les unes dans les autres.

Syntaxe 1 :



Syntaxe 2 :



Exemple : Ecrire un algorithme qui donne l'état de l'eau selon sa température.

Première solution avec des SI imbriquées première syntaxe:

```
Algorithme température de l'eau
Variables Temp : Réel
Début
        Ecrire ("Entrez la température de l'eau : ")
        Lire(Temp)
        Si Temp =< 0 Alors
                Ecrire ("C'est de la glace")
        Sinon
                Si Temp < 100 Alors
                        Ecrire ("C'est du liquide")
                Sinon
                        Ecrire ("C'est de la vapeur")
                Finsi
        FinSi
Fin
```

Deuxième solution avec des SI imbriquées deuxième syntaxe:

```
Algorithme température de l'eau
Variables Temp : Réel
Début
        Ecrire ("Entrez la température de l'eau : ")
        Lire(Temp)
        Si Temp =< 0 Alors
                Ecrire ("C'est de la glace")
        Sinon Si Temp < 100 Alors
                Ecrire ("C'est du liquide")
        Sinon
                Ecrire ("C'est de la vapeur")
        Finsi
Fin
```

Troisième solution avec des SI successives:

```
Algorithme température de l'eau
Variables Temp : Réel
Début
        Ecrire ("Entrez la température de l'eau : ")
        Lire(Temp)
        Si Temp =< 0 Alors
                Ecrire ("C'est de la glace")
        FinSi
        Si   Temp > 0   ET   Temp < 100   Alors
                Ecrire ("C'est du liquide")
        FinSI
        Si Temp >= 100 Alors
                Ecrire ("C'est de la vapeur")
        Finsi
Fin
```

2. Les instructions d'itération ou de répétition (les boucles)

Les structures répétitives aussi appelées boucles, permettent de répéter un traitement (c'est à dire une instruction simple ou composée) autant de fois qu'il est nécessaire : soit un nombre déterminé de fois, soit tant qu'une condition est vraie.

Il existe trois grands types principaux de structures répétitives :

- la structure **TantQue…Faire** : qui permet d'effectuer un traitement tant qu'une condition est satisfaite.
- la structure **Pour…Faire** : qui permet de répéter un traitement un certain nombre de fois.

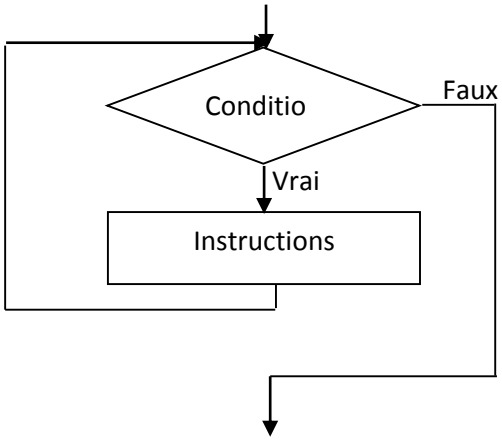
Seule la boucle **TantQue** est fondamentale. Avec cette boucle, on peut réaliser toutes les autres boucles alors que l'inverse n'est pas vrai. La boucle **Pour** est très utilisée aussi car elle permet de simplifier la boucle **TantQue** lorsque le nombre de tour de boucle est connu d'avance.

a) La boucle TantQue :

La boucle **TantQue** permet de répéter un traitement tant qu'une expression conditionnelle est vraie. **Si la condition n'est pas vraie, le traitement ne sera pas exécuté**. On voit donc que la boucle **TantQue** a un point commun avec la structure conditionnelle réduite où si la condition n'est pas vraie, le traitement n'est pas exécuté.

Syntaxe :

```
Tantque <condition de continuation> Faire
        Instruction 1
            ……
        Instruction N
FinTantque
```



Les instructions seront exécutées tant que la condition est vérifiée (valeur booléenne vrai).

Le test est effectué à l'entrée de la boucle, donc les instructions peuvent être exécutées zéro fois si la condition est fausse à l'entrée de la boucle.

Remarque :

Dans le corps de la boucle (instructions), au moins une instruction doit obligatoirement faire évoluer les valeurs des variables testés dans la condition. Sinon la boucle sera infinie (c-à-d qu'elle ne s'arrête jamais).

Exemple 1 : Une boucle infinie

```
Algorithme boucle_infinie ;
Variables
        A : réel
Début
        A← 6
        Tantque A<10 faire
                Ecrire(A)
        FinTantque
Fin
```

Commentaire :

Dans le corps de la boucle de l'exemple, rien ne fait évoluer la variable A qui reste toujours inférieure à 10. Il y aura donc un affichage de la valeur 6 éternellement.

Exemple 2 :

Ecrire un algorithme permettant de lire une suite de nombres réels sur clavier. Le dernier élément à lire est Zéro. L'algorithme doit afficher le plus petit élément de la suite ainsi que la somme des éléments lus.

```
Algorithme Somme_min
Variables Elt , S , min : Réels
Debut
        S← 0
        Lire(elt)
        min ←elt
        TantQue elt<>0 Faire
                S←S+elt
                Si elt < min Alors
                        min←elt
                FinSi
                Lire(elt)
        FinTantQue
        Ecrire(S)
        Ecrire(min)
Fin
```

Commentaire :

Dans cet exemple la condition d'arrêt est lorsque l'utilisateur donne la valeur 0.

b) La boucle Pour :

Cette boucle est utilisée lorsque le nombre d'itérations (répétitions) est connu à l'avance ou à l'arrivé à la boucle en exécution.

Syntaxe :

```
Pour C=VI à VF par PAS faire
        …..    // Block d'instructions à répéter
FinPour
```

VI : La valeur initiale de la boucle.
VF : La valeur finale de la boucle.
C : Le compteur d'itération qui au début prend la valeur initiale VI et s'incrémente par la valeur « PAS » après chaque répétition, si la valeur de C devient plus grande que VF le programme sort de la boucle pour continuer le reste de l'algorithme.

Exemples :

Pour i←1 à 20 par 1 faire Ecrire(i) FinPour Cette boucle permet d'afficher les nombres 1 jusqu'à 20	Pour i←1 à 20 par 2 faire Ecrire(i) FinPour Cette boucle permet d'afficher les nombres 1, 3, 5, 7, 9, 11, 13, 15, 17, 19
Pour i←1 à 20 par 5 faire Ecrire(i) FinPour Cette boucle permet d'afficher les nombres 1, 6, 11, 16	Pour i←5 à 1 par -1 faire Ecrire(i) FinPour Cette boucle permet d'afficher les nombres 5, 4, 3, 2, 1
Pour i←5 à 1 par -6 faire Ecrire(i) FinPour Cette boucle permet d'afficher le nombre 5	Pour i←5 à 2 par 1 faire Ecrire(i) FinPour Cette boucle n'affiche rien, aucune répétition n'est effectuée parce que la valeur finale est supérieure à la valeur finale.

Remarque : On peut écrire la boucle **Pour** sans spécifier le pas d'incrémentation, dans ce cas le pas d'incrémentation par défaut est 1.

Exemple :

```
Pour i←1 à 20 faire
        Ecrire(i)
FinPour

Cette boucle permet d'afficher les nombres 1, 3, 5, 7, 9, 11, 13, 15, 17, 19
```

Exercices corrigés :

Exercice 1 :

Ecrire un algorithme qui calcule la somme des entiers de 1 à un entier entré par l'utilisateur.

```
Algorithme somme
Variables s, n, i : entiers
Début
    Ecrire("Donner un entier : ")
    Lire(n)
    s ←0
    Pour i←1 à n faire
        s ← s + i
    FinPour
    Ecrire("La somme = ", s)
Fin
```

Exercice 2 :

Ecrire un algorithme qui calcule et affiche le factoriel d'un entier positif entré par l'utilisateur.

```
Algorithme factoriel
Variables f, n, i : entiers
Début
    Ecrire("Donner un entier : ")
    Lire(n)
    f ←0
    Pour i←1 à n faire
        f ← f * i
    FinPour
    Ecrire(n, " !=" , f)
Fin
```

Exercice 3 :

Ecrire un algorithme qui calcule a^b pour deux entiers a et b entré par l'utilisateur.

```
Algorithme factoriel
Variables a, b, p , i : entiers
Début
    Ecrire("Donner a : ")
    Lire(a)
    Ecrire("Donner b : ")
    Lire(b)
    p ←1
    Si b>=0 alors
        Pour i←1 à b faire
            p ← p * a
        FinPour
    SiNon
        Pour i←1 à -b faire
            p ← p * a
        FinPour
        p=1/p
    FinSI
    Ecrire("Résultat : ", p)
Fin
```