Algorithmique

Cours

CHAPITRE 1 : Introduction

La **DEMARCHE ALGORITHMIQUE** est une composante essentielle de l'**activité mathématique**.

- L'usage et le développement des algorithmes font partie de notre quotidien ; ils sont notamment à la base du fonctionnement des **automates**, des **calculatrices** et des **ordinateurs**.
- De nombreux algorithmes sont connus depuis l'Antiquité, dont l'**algorithme d'Euclide**, qui permet de calculer le PGCD de deux nombres entiers non nuls.

Ce cours se donne pour objectif la maîtrise des **COMPÉTENCES** suivantes :

- **comprendre** et **examiner** un algorithme préexistant, son fonctionnement ou son but ;
- **modifier** un algorithme pour obtenir un résultat précis ;
- **analyser** une situation : identifier les données d'entrée et de sortie, le traitement, les instructions... ;
- **créer** une solution algorithmique à un problème donné : comment écrire un algorithme en « langage courant » en respectant un code, identifier les boucles, les tests, les opérations d'écriture, d'affichage... ;
- **valider** la solution algorithmique par des traces d'exécution et des jeux d'essais simples ;
- **adapter** l'algorithme aux contraintes du langage de programmation : identifier si nécessaire la nature des variables... ;
- valider un programme simple.

Sont présentés dans ces pages quelques **EXEMPLES D'ALGORITHMES** dont l'écriture touche tous les domaines du programme :

- **fonctions** : étude numérique, étude asymptotique…
- **géométrie** : affichage, positionnement et déplacement d'objets géométriques simples (points, segments, cercles), colinéarité, orthogonalité…
- **statistique** : tris, détermination de certains indicateurs (moyenne, médiane, quartiles)…
- **probabilités** : modélisation de certains phénomènes à partir de fréquences observées…
- **numérique** : traitement de nombres, comparaisons, exactitude dans les calculs…

La mise en œuvre d'algorithmes peut se faire à l'aide de nombreux **LOGICIELS** :

- **des logiciels dédiés** : ALGOBOX, SCRATCH, EXECALGO, LINOTTE...
- **des logiciels de programmation** : PYTHON...
- **des logiciels liés au calcul scientifique** : SCILAB...
- **des logiciels de calcul formel** : XCAS, MAXIMA, WIRIS...

Le choix du logiciel (et plus généralement de l'outil informatique) dépend souvent de la complexité de l'algorithme et notamment du temps de calcul, de la nature, de la taille ou de la précision des nombres utilisés, de la lisibilité de l'algorithme ou de la nature de la sortie…

En résumé, on peut considérer dans une **PREMIERE APPROCHE** qu'un algorithme décrit un processus de résolution d'un problème défini, rédigé dans un langage formalisé et produisant un résultat en un temps fini.

Le mot **ALGORITHME** tire son nom du mathématicien persan **Al-Khuwarizmi** (né vers 780 - mort vers 850) qui a écrit en langue arabe le plus ancien traité d'algèbre baptisé « Abrégé de calcul par la complétion et la simplification » dans lequel il décrivait des procédés de calcul à suivre étape par étape pour résoudre des problèmes ramenés à des équations. Toutefois, les algorithmes existent depuis plus longtemps : les Mésopotamiens calculaient déjà en 1 800 avant J.-C. des valeurs approchées des racines carrées à l'aide d'algorithmes.

CHAPITRE 2 : Qu'est-ce qu'un algorithme ?

1) DEFINITION

Un **ALGORITHME** est une suite finie d'**instructions élémentaires** (règles), qui s'appliquent dans un ordre déterminé à un nombre fini de **données** pour fournir un **résultat**.

Exemple : Suivre une recette de cuisine, calculer une somme, tracer une figure dans le plan… sont autant d'activités pour lesquelles une série d'actions sont à effectuer une ou plusieurs fois afin d'obtenir un résultat.

Tout algorithme est donc caractérisé par :

- un ensemble d'**instructions** à exécuter
- un **ordre d'exécution** de ces différentes actions, déterminé par la logique d'enchaînement et conditionné par les **structures** mises en œuvre
- un **début** et une **fin**

Remarque : Dans la suite du cours, on entend par **TRAITEMENT** soit une instruction isolée, soit une succession d'instructions.

2) SQUELETTE D'UN ALGORITHME

Trois phases indissociables structurent un algorithme :

1- **La préparation du traitement**
2- **Le traitement de donnée(s)**
3- **La sortie de résultat(s)**

Exemple 1 : Lors de la conception d'un gâteau, il faut envisager plusieurs phases, rigoureusement ordonnées. Tout d'abord, il convient d'acheter les ingrédients ($1^{ère}$ phase) qui entrent dans la composition de la recette. Attention ! Un seul oubli pourrait en compromettre la réussite ! Ensuite, il faut suivre étape par étape les actions à mener comme peser les ingrédients, battre les œufs en neige, faire fondre le beurre… ($2^{ème}$ phase). Attention ! Il suffit d'inverser des étapes, de ne pas respecter les proportions… et le gâteau risquerait fort d'être raté ! Enfin, quoiqu'il advienne lors du suivi de la recette, on obtient un résultat ($3^{ème}$ phase). En principe, ce résultat doit ressembler au gâteau attendu, tant visuellement que sur les plans gustatif, olfactif et pourquoi pas tactile !

Exemple 2 : De nos jours, il n'est pas rare d'utiliser un navigateur GPS pour obtenir un itinéraire (but de l'algorithme). On entre alors le point de départ et le point d'arrivée (données d'entrée – $1^{ère}$ phase). Une série d'instructions (traitement des données – $2^{ème}$ phase) fournit en sortie une ligne brisée (résultat – $3^{ème}$ phase) qui symbolise le chemin à parcourir pour joindre ces deux points.

Mais comment écrire un algorithme pour qu'il soit universellement compréhensible ? Un algorithme peut être soit écrit sous forme littérale **(langage algorithmique)**, soit représenté graphiquement **(algorigramme)**.

3) LANGAGE ET REGLES D'ECRITURE D'UN ALGORITHME

Un algorithme peut être écrit en utilisant un **langage de description d'algorithme** (LDA). Ce langage utilise un ensemble de mots clés et de structures permettant de décrire de manière complète et claire l'ensemble des opérations à exécuter sur des données pour obtenir des résultats. Un tel langage présente un réel avantage, celui de pouvoir être transcrit dans un langage de programmation structuré et lisible. Il ne faut donc pas confondre algorithme et programme.

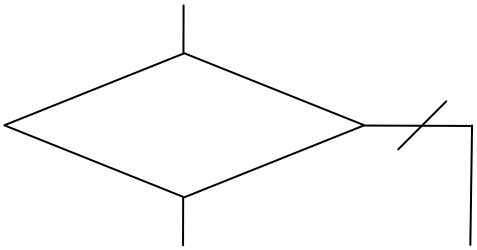
4) ALGORIGRAMME

En algorithmique, on peut aussi utiliser un **algorigramme**, c'est-à-dire représenter graphiquement l'algorithme à l'aide de symboles normalisés.

Exemples :

- **Symbole de traitement**
  (symbole général)

- **Symbole de test**
  (symbole de branchement)



- **Symbole auxiliaire**
  (symbole de début, fin ou interruption d'algorithme)

Remarque : On parle souvent indifféremment d'algorigramme ou d'organigramme.

1) LA PREPARATION DU TRAITEMENT

Cette phase consiste à repérer les données, c'est-à-dire les éléments nécessaires, voire indispensables, à la résolution. Elles peuvent être de nature :

- **numérique** (des nombres entiers ou réels)
- **textuelle** (des caractères, c'est-à-dire une lettre, un chiffre ou un symbole, ou des chaînes de caractères, c'est-à-dire une suite de caractères formés de lettre(s), de chiffre(s) et/ou de symbole(s))
- **booléenne** (de type logique, à deux valeurs possibles « vrai » ou « faux »)
- **graphique** (des points)

Autrement dit, après avoir précisé en en-tête le nom de l'algorithme afin d'en identifier le but, l'algorithme se compose des déclarations de constantes, de variables et de structures, qui correspondent à une liste exhaustive des ressources utilisées et manipulées dans le corps de l'algorithme.

Remarque : L'**entrée des données** (saisie de caractères ou de nombres sur le clavier, lecture de la position du pointeur de la souris, lecture d'un fichier contenant ces nombres ou caractères…) s'intègre dans cette phase de préparation du traitement.

2) LE TRAITEMENT DE DONNEES

Cette phase, qui correspond au corps de l'algorithme, consiste à spécifier toutes les étapes des instructions à donner pour une exécution automatique.

Il existe plusieurs types d'algorithmes, déterminés selon le type d'exécution des instructions :

- si les instructions s'exécutent en séquence, on parle d'**algorithme séquentiel** ;
- si les opérations s'exécutent sur plusieurs processeurs en parallèle, on parle d'**algorithme parallèle** ;
- si les instructions s'exécutent sur un réseau de processeurs, on parle d'**algorithme réparti** ou **distribué**.

Remarque : Ce cours ne s'intéresse qu'aux algorithmes séquentiels et il faut par conséquent entendre par traitement une ou plusieurs instructions en séquence.

3) LA SORTIE DE RESULTATS

Les résultats obtenus, graphiques ou sonores, peuvent être :

- affichés sur l'écran
- imprimés sur papier
- conservés dans un fichier
- conservés en mémoire jusqu'à la prochaine exécution
- perdus

CHAPITRE 4 : Constantes et variables

Les constantes et les variables sont des éléments fondamentaux, indispensables au bon déroulement d'un algorithme, caractérisés par un **identificateur**, une **valeur** et un **type**.

1) DEFINITIONS

- Une **VARIABLE** est une **donnée** (emplacement) stockée dans la mémoire de la calculatrice ou de l'ordinateur. Elle est repérée par un **identificateur** (nom de la variable constitué de lettres et/ou de chiffres, sans espace) et contient une **valeur** dont le **type** (nature de la variable) peut être un entier, un réel, un booléen, un caractère, une chaîne de caractères… Il ne faut pas confondre **constante** et **variable**.

- Une **CONSTANTE**, comme une variable, peut représenter un chiffre, un nombre, un caractère, une chaîne de caractères, un booléen. Toutefois, contrairement à une variable dont la valeur peut être modifiée au cours de l'exécution de l'algorithme, la valeur d'une constante ne varie pas.

Remarques :

- Ne pas confondre la variable et son identificateur. En effet, la variable possède une valeur (son contenu) et une adresse (emplacement dans la mémoire où est stockée la valeur). L'identificateur n'est que le nom de la variable, c'est-à-dire un constituant de cette variable.
- Le type d'une variable détermine l'ensemble des valeurs qu'elle peut prendre et les opérations réalisables qu'elle peut subir.
- L'utilisation d'une variable doit être précédée de sa déclaration. La syntaxe pour déclarer une variable est la suivante :

```
Variables :
    identificateur de la variable_1 : type de la variable_1
    identificateur de la variable_2 : type de la variable_2
    …
    identificateur de la variable_N : type de la variable_N
```

- Si la valeur de la variable peut changer au cours du déroulement de l'algorithme, en revanche son type est figé lors de déclaration.

2) CONVENTIONS DE NOMMAGE

Le nom d'un algorithme, d'une variable ou d'une constante doit respecter les **règles** suivantes :

- commencer par une lettre ;
- ne comporter ni caractère spécial (comme l'espace) ni ponctuation ;

- ne pas être un mot du langage algorithmique (comme « algorithme », « début », « fin », « variable », « non », « ou », « et », « si », « sinon », « pour »…)

CHAPITRE 5 : OPERANDES ET OPERATEURS

1) DEFINITIONS

- Un **OPERATEUR** est un outil qui permet d'agir sur une variable ou d'effectuer des calculs.
- Un **OPERANDE** est une donnée utilisée par un opérateur.

Exemple : Dans « $7 - x$ », « $-$ » désigne l'opérateur ; « 7 » et « x » sont les opérandes.

2) TYPES D'OPERATEURS

Il existe plusieurs types d'opérateurs :

- Les **opérateurs arithmétiques** qui permettent d'effectuer des opérations arithmétiques entre opérandes numériques :
  - Opérateurs élémentaires : « $+$ », « $-$ », « $\times$ », « $\div$ », « $/$ » (division entière)
  - Changement de signe : « $-$ »
  - Elévation à la puissance : « ^ »
  - Reste d'une division entière : « $modulo$ » (ou « mod »)
- Les **opérateurs de comparaison** (« $=$ », « $\neq$ », « $>$ », « $<$ », « $\geq$ » et « $\leq$ ») qui permettent de comparer deux opérandes et produisent une valeur booléenne, en s'appuyant sur des relations d'ordre :
  - Ordre naturel pour les entiers et les réels
  - Ordre lexicographique ASCII pour les chaînes de caractère
- Les **opérateurs logiques** qui combinent des opérandes booléennes pour former des expressions logiques plus complexes :
  - Opérateur unaire : « non » (négation)
  - Opérateurs binaires : « et » (conjonction), « ou » (disjonction), « $ou\_exclusif$ »
- L'**opérateur de concaténation** qui permet de créer une chaîne de caractères à partir de deux chaînes de caractère en les mettant bout à bout.
- L'**opérateur d'affectation**, représenté par le symbole « $\leftarrow$ », qui confère une valeur à une variable ou à une constante.

$a \leftarrow 7$ (affectation de la valeur 7 à la variable (ou à la constante) a)

Remarque : Les opérateurs dépendent du type de la constante ou de la variable :

- **Opérateurs sur les entiers et les réels :** addition, soustraction, multiplication, division, division entière, puissance, comparaisons, modulo (reste d'une division entière)
- **Opérateurs sur les booléens :** comparaisons, négation, conjonction, disjonction
- **Opérateurs sur les caractères :** comparaisons
- **Opérateurs sur les chaînes de caractères :** comparaisons, concaténation

3) PRIORITE DES OPERATEURS

A chaque opérateur est associée une priorité. Lors de l'évaluation d'une expression, la priorité de chaque opérateur permet de définir l'ordre d'exécution des différentes opérations. Aussi, pour lever toute ambiguïté ou pour modifier l'ordre d'exécution, on peut utiliser des parenthèses.

- **Ordre de priorité décroissante des opérateurs arithmétiques et de concaténation :**
  - « ^ » (élévation à la puissance)
  - « − » (changement de signe)
  - « × », « ÷ » et « / »
  - « $modulo$ »
  - « + » et « − »
  - « + » (concaténation)
- **Ordre de priorité décroissante des opérateurs logiques :**
  - « non »
  - « et »
  - « ou »
  - « $ou\_exclusif$ »

Remarques :

- La question de l'ordre de priorité des opérateurs de comparaison ne se pose pas.
- Les opérations entre parenthèses sont prioritaires.

DEFINTIONS :

- L'instruction d'**AFFECTATION** permet d'attribuer une valeur à une variable. Cette instruction est notée « *identificateur* **prend** *valeur* ».
  Comme il a été vu précédemment, on peut aussi noter : « *identificateur* ← *valeur* ».

- L'**ENTREE** ou la **lecture** de données correspond à l'opération qui permet de saisir des valeurs pour qu'elles soient utilisées par le programme. Cette instruction est notée « **saisir** *identificateur* » ou « **lire** *identificateur* ».

- La **SORTIE** ou l'**écriture** des données permet l'affichage des valeurs des variables après traitement. Cette instruction est notée « **afficher** *identificateur* ».

Remarques :

- L'affectation d'une valeur à une variable peut être assimilée au rangement d'un objet dans un tiroir, sur la façade duquel figure l'identificateur, et qui ne peut contenir qu'un objet à la fois.
- En sortie, on peut également afficher un message suivi d'une expression ; l'instruction est alors notée « **afficher** (*message*, *expression)* ».

Exemple 1 (numérique) : Proposons un algorithme qui permet de calculer le prix toutes taxes comprises d'un article après l'application d'une TVA à 19,6 % sur le prix hors taxes.

Algorithme Calcul_Prix_TTC (nom de l'algorithme)

Constantes

TVA : réel (ici, seule la variable TVA est déclarée ; son type est réel car le nombre 19,6 % n'est pas entier ; remarque : cette constante peut aussi être intégrée dans les variables)

Variables

$prixHT$, $prixTTC$: réels (déclaration des variables pour lesquelles il reste préférable de donner un identificateur explicite ; ici $prixHT$ désigne le prix hors taxe et $prixTTC$ le prix toutes taxes comprises)

Début

TVA prend 0,196 ; (on affecte la valeur 0,196, soit 19,6 %, à la variable TVA ; on peut également noter cette affectation $TVA \leftarrow 0{,}196$)

afficher (« Renseigner le prix hors taxes de l'article. ») ; (affichage de la phrase entre guillemets)

saisir ($prixHT$) ; (lecture de la valeur de la variable $prixHT$)

$prixTTC$ prend ($prixHT + prixHT \times TVA$) ; (calcul du prix TTC avec l'opérateur × prioritaire devant l'opérateur +)

afficher (« Après application de la TVA, le prix de l'article devient », $prixTTC$, « toutes taxes comprises. ») (sortie du résultat qui proposera l'affichage de la 1ère locution entre guillemets, puis le prix hors taxes qui avait été saisi par l'utilisateur, puis la 2e locution entre guillemets, et enfin le prix toutes taxes comprises qui a été calculé dans le corps de l'algorithme)

Fin

Avec le logiciel AlgoBox, on obtient l'algorithme suivant :

```
VARIABLES
    TVA EST_DU_TYPE NOMBRE
    prixHT EST_DU_TYPE NOMBRE
    prixTTC EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
    TVA PREND_LA_VALEUR 0.196
    AFFICHER "Renseigner le prix hors taxes de l'article."
    LIRE prixHT
    prixTTC PREND_LA_VALEUR prixHT+prixHT*TVA
    AFFICHER "Après application de la TVA, le prix de l'article devient : "
    AFFICHER prixTTC
    AFFICHER " toutes taxes comprises."
FIN_ALGORITHME
```

Affichages après lancement de l'algorithme avec AlgoBox :

```
***Algorithme lancé***
Renseigner le prix hors taxes de l'article.
Après application de la TVA, le prix de l'article devient : 143.52 toutes taxes comprises
***Algorithme terminé***
```

Exemple 2 (fonction) : Ecrivons un algorithme avec AlgoBox puis un programme avec la calculatrice qui permettent de donner la forme canonique d'une fonction polynôme f de degré 2, en considérant que, pour tout x réel, $f(x) = ax^2 + bx + c$ $(a \neq 0)$.

Rappel :

Si f est une fonction polynôme de degré 2 définie par $f(x) = ax^2 + bx + c$ $(a \neq 0)$, alors sa forme canonique est :

$$f(x) = a(x-\alpha)^2 + \beta$$

avec

$$\begin{cases} \alpha = -\dfrac{b}{2a} \\ \beta = f(\alpha) = \dfrac{-b^2+4ac}{4a} \end{cases}$$

- Avec le logiciel AlgoBox :

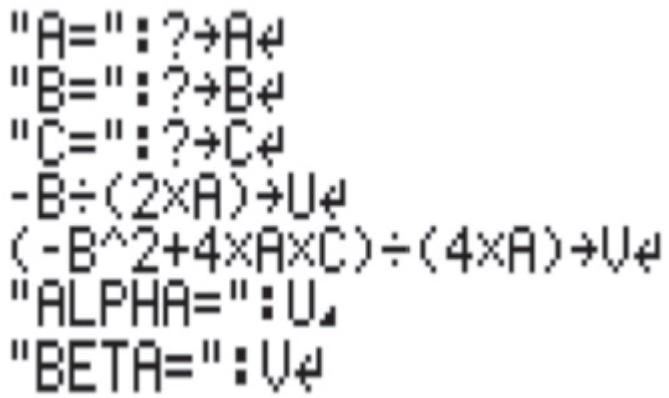
```
VARIABLES
    a EST_DU_TYPE NOMBRE
    b EST_DU_TYPE NOMBRE
    c EST_DU_TYPE NOMBRE
    alpha EST_DU_TYPE NOMBRE
    beta EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
    AFFICHER "Soit un trinôme du second degré ax²+bx+c."
    AFFICHER "Donner la valeur de a (a non nul) : "
    LIRE a
    AFFICHER a
    AFFICHER "Donner la valeur de b : "
    LIRE b
    AFFICHER b
    AFFICHER "Donner la valeur de c : "
    LIRE c
    AFFICHER c
    alpha PREND_LA_VALEUR -b/(2*a)
    beta PREND_LA_VALEUR (-pow(b,2)+4*a*c)/(4*a)
    AFFICHER "La forme canonique est a(x-alpha)²+beta avec :"
    AFFICHER "alpha = "
    AFFICHER alpha
    AFFICHER "beta = "
    AFFICHER beta
FIN_ALGORITHME
```

La commande « pow(x,n) » permet de calculer x^n.

Affichage après lancement de l'algorithme avec AlgoBox :

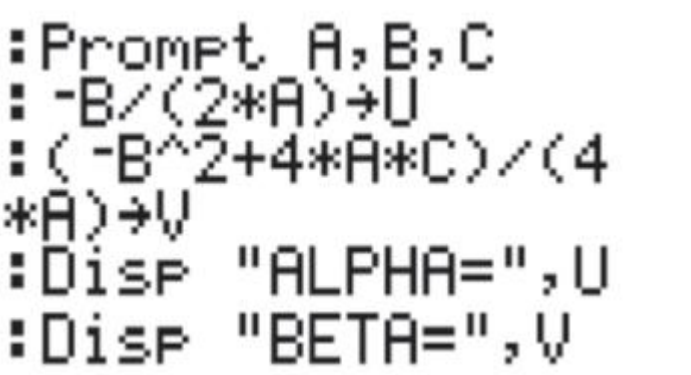
```
***Algorithme lancé***
Soit un trinôme du second degré ax²+bx+c.
Donner la valeur de a (a non nul) : 1
Donner la valeur de b : 3
Donner la valeur de c : 2
La forme canonique est a(x-alpha)²+beta avec :
alpha = -1.5
beta = -0.25
***Algorithme terminé***
```

- Avec une calculatrice Casio :



- La lecture des variables se fait grâce à « : ? ».
- L'affectation est donnée par la commande « → »
- L'affichage est rendu possible par l'instruction « : ».

- Avec une calculatrice Texas Instruments :



- L'instruction « *prompt* » permet de demander à l'utilisateur de saisir une valeur qui sera affectée à la variable.
- L'instruction d'affectation est donnée par « → »
- L'instruction « *disp* » permet l'affichage.

Exemple 3 (géométrie) : Proposons un algorithme qui permet de donner la distance *AB* dans un repère du plan, les coordonnées des points *A* et *B* étant renseignées par l'utilisateur.

- En langage algorithmique simple :

Algorithme Distance_entre_deux_points

Entrée

Saisir les coordonnées x_A , y_A , x_B , y_B respectives des points A et B

Traitement

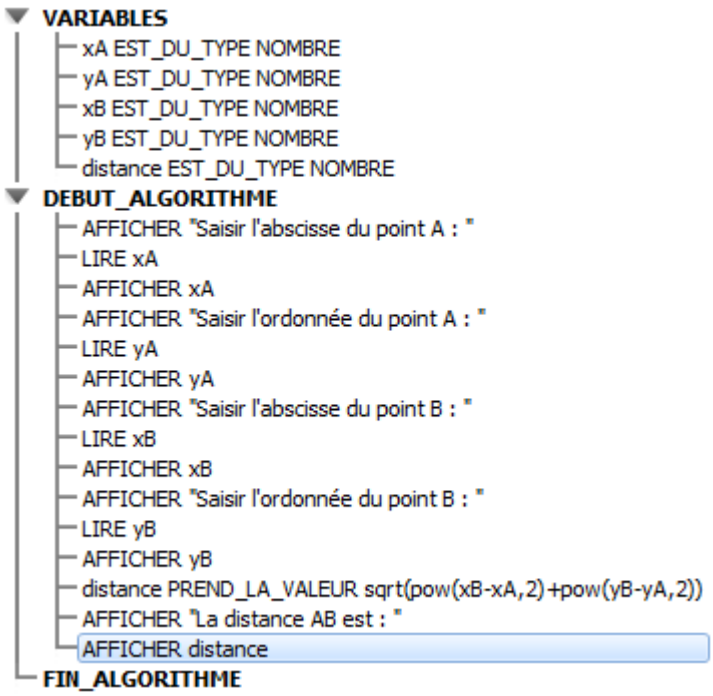
Affecter à la variable $distance$ la valeur $\sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$

Sortie

Afficher $distance$

Rappel : Si $A(x_A\,;\,y_A)$ et $B(x_B\,;\,y_B)$ sont deux points d'un repère du plan, alors :
$AB = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$.

- Algorithme avec le logiciel AlgoBox :



Affichages après lancement de l'algorithme avec AlgoBox :

```
***Algorithme lancé***
Saisir l'abscisse du point A : 5
Saisir l'ordonnée du point A : -4
Saisir l'abscisse du point B : 2
Saisir l'ordonnée du point B : 0
La distance AB est : 5
***Algorithme terminé***
```

Remarque : La commande « sqrt(x) » permet de calculer $\sqrt{x}$.

CHAPITRE 7 : Structures de contrôle

Le traitement de données est parfois conditionné et se réalise de manière spécifique. On parle alors de **STRUCTURES DE CONTROLE**. Ces structures algorithmiques peuvent être organisées suivant quatre familles principales :

- **les structures linéaires**
- **les structures alternatives**
- **les structures répétitives**
- **les structures de choix**

DEFINITIONS :

- Une **STRUCTURE LINEAIRE** (ou **STRUCTURE SEQUENTIELLE**) se caractérise par une suite de traitements à exécuter successivement, dans l'ordre énoncé.
- Une **CONDITION** est une expression logique booléenne, prenant la valeur « vrai » ou « faux » (c'est-à-dire « oui » ou « non »).
- Une **STRUCTURE ALTERNATIVE** (ou **STRUCTURE CONDITIONNELLE**) n'offre que deux issues possibles à la poursuite de l'algorithme, qui s'excluent mutuellement. Selon qu'une condition est vraie ou fausse, on effectue un traitement ou un autre. On parle de traitements conditionnels. Une structure alternative est donc une structure de test.
- Une **STRUCTURE REPETITIVE** (ou **STRUCTURE ITERATIVE**) répète l'exécution d'un traitement, dans un ordre précis, un nombre déterminé ou indéterminé de fois. Une structure itérative est aussi appelée boucle.
- Une **STRUCTURE DE CHOIX** permet, en fonction de plusieurs conditions de type booléen, d'exécuter des traitements différents selon les valeurs que peut prendre une même variable.

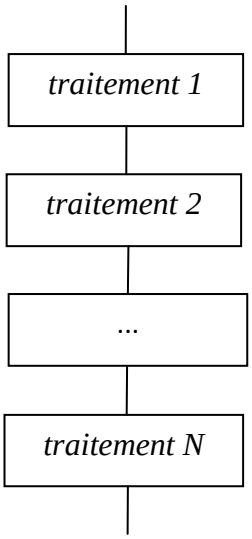
On note très simplement les structures linéaires, comme une suite de traitements à exécuter dans l'ordre où ils sont énoncés :

Faire *traitement 1* ;

Faire *traitement 2* ;

… ;

Faire *traitement N* ;



Algorigramme

Exemple (statistique) : Proposons un algorithme qui calcule la moyenne de 3 notes, toutes au même coefficient.

```
VARIABLES
    note1 EST_DU_TYPE NOMBRE
    note2 EST_DU_TYPE NOMBRE
    note3 EST_DU_TYPE NOMBRE
    moyenne EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
    AFFICHER "Rentrer la première note."
    LIRE note1
    AFFICHER "Rentrer la deuxième note."
    LIRE note2
    AFFICHER "Rentrer la troisième note."
    LIRE note3
    AFFICHER "Les trois notes renseignées sont : "
    AFFICHER note1
    AFFICHER " ; "
    AFFICHER note2
    AFFICHER " ; "
    AFFICHER note3
    moyenne PREND_LA_VALEUR (note1+note2+note3)/3
    AFFICHER "La moyenne des trois notes est : "
    AFFICHER moyenne
FIN_ALGORITHME
```

Affichage après lancement de l'algorithme avec AlgoBox :

```
***Algorithme lancé***
Rentrer la première note.
Rentrer la deuxième note.
Rentrer la troisième note.
Les trois notes renseignées sont :
15.5 ; 10 ; 11
La moyenne des trois notes est :
12.166667
***Algorithme terminé***
```

CHAPITRE 9 : Structures alternatives

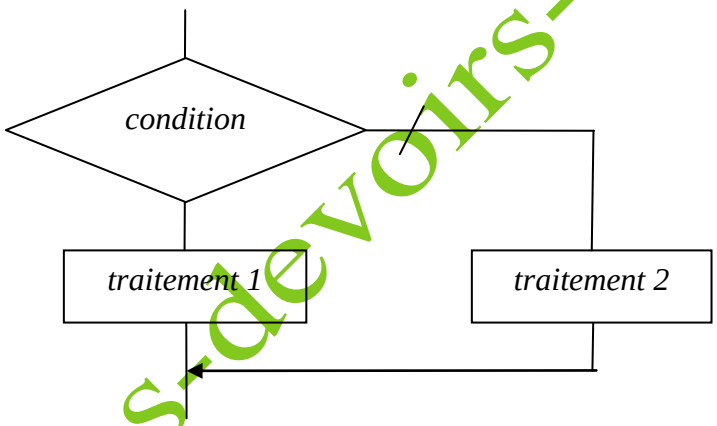
La résolution de certains problèmes nécessite parfois la mise en place d'un test pour effectuer une tâche :

- si le test est positif, on effectue un certain traitement ;
- sinon, c'est-à-dire si le test est négatif, on effectue un autre traitement.

En algorithmique, on traduit cette structure alternative à l'aide d'**INSTRUCTIONS CONDITIONNELLES**.

1) STRUCTURE ALTERNATIVE COMPLETE

Si *condition*
Alors *traitement 1* ; (instructions à effectuer si la *condition* est vérifiée)
Sinon
traitement 2 ; (instructions à effectuer si la *condition* n'est pas vérifiée)
FinSi

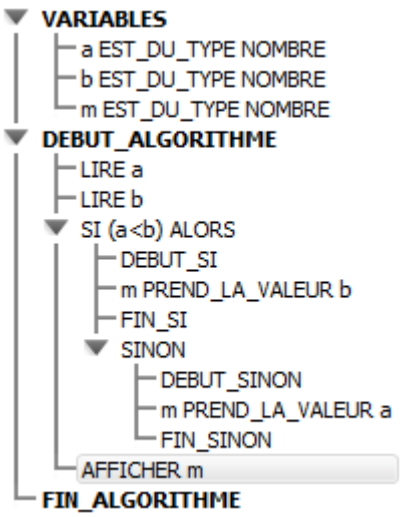


Algorigramme

Exemple 1 (numérique) : Proposons un algorithme qui compare deux nombres.

Dans cet algorithme généré par AlgoBox, l'utilisateur est invité à saisir deux nombres (variables a et b).
Si la valeur de la variable a est plus petite que la valeur de la variable b, alors la variable m est affectée de la valeur de la variable b.
Sinon, c'est-à-dire si la valeur de la variable a est plus grande que la valeur de la variable b, alors m prend la valeur de la variable a.

Le résultat retourné est la valeur de la variable m, qui correspond au plus grand des deux nombres choisis initialement par l'utilisateur.



Exemple 2 (géométrie) : Ecrivons un algorithme qui permet de savoir si deux vecteurs sont colinéaires ou non.

```
VARIABLES
    xU EST_DU_TYPE NOMBRE
    yU EST_DU_TYPE NOMBRE
    xV EST_DU_TYPE NOMBRE
    yV EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
    AFFICHER "Saisir l'abscisse du premier vecteur : "
    LIRE xU
    AFFICHER xU
    AFFICHER "Saisir l'ordonnée du premier vecteur : "
    LIRE yU
    AFFICHER yU
    AFFICHER "Saisir l'abscisse du deuxième vecteur : "
    LIRE xV
    AFFICHER xV
    AFFICHER "Saisir l'ordonnée du deuxième vecteur : "
    LIRE yV
    AFFICHER yV
    SI (xU*yV-yU*xV==0) ALORS
        DEBUT_SI
        AFFICHER "Les vecteurs sont colinéaires."
        FIN_SI
        SINON
            DEBUT_SINON
            AFFICHER "Les vecteurs ne sont pas colinéaires."
            FIN_SINON
FIN_ALGORITHME
```

Rappel :

Deux vecteurs $\vec{u}$ et $\vec{v}$ de coordonnées respectives $\begin{pmatrix} x_{\vec{u}} \\ y_{\vec{u}} \end{pmatrix}$ et $\begin{pmatrix} x_{\vec{v}} \\ y_{\vec{v}} \end{pmatrix}$ sont colinéaires si et seulement si :

$$x_{\vec{u}}y_{\vec{v}} - y_{\vec{u}}x_{\vec{v}} = 0$$

Affichages après lancement de l'algorithme avec Algo Box :

```
***Algorithme lancé***
Saisir l'abscisse du premier vecteur : 1
Saisir l'ordonnée du premier vecteur : 2
Saisir l'abscisse du deuxième vecteur : 4
Saisir l'ordonnée du deuxième vecteur : 8
Les vecteurs sont colinéaires.
***Algorithme terminé***
```

```
***Algorithme lancé***
Saisir l'abscisse du premier vecteur : 1
Saisir l'ordonnée du premier vecteur : 2
Saisir l'abscisse du deuxième vecteur : 4
Saisir l'ordonnée du deuxième vecteur : -7
Les vecteurs ne sont pas colinéaires.
***Algorithme terminé***
```

Remarque : Il existe des conditions simples et des conditions complexes :

- une **condition simple** peut correspondre à un test d'égalité (par exemple :$A = B$) ou d'inégalité (par exemple : $A \leq B$) ;
- une **condition complexe** est une combinaison logique de conditions simples (par exemple : $A = B$ et $B < C$).

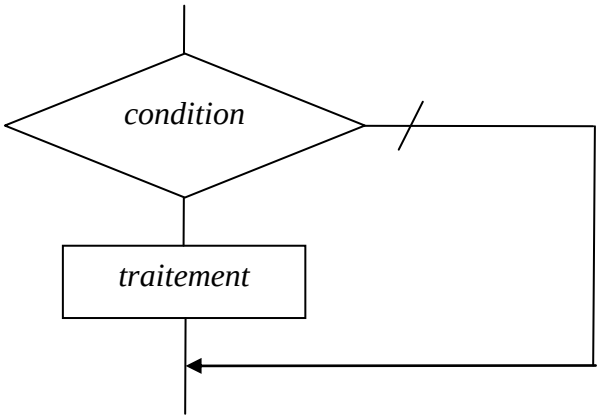
2) STRUCTURE ALTERNATIVE REDUITE

La structure proposée ci-dessus est qualifiée de « complète » mais, selon le cas, il se peut que, si la *condition* n'est pas vérifiée, il n'y ait pas à effectuer de *traitement 2*. On écrira ainsi la structure alternative « réduite ».

Si *condition*

Alors *traitement* (instructions à effectuer si la *condition* est vérifiée)

FinSi (la condition s'achève sans qu'on ait eu à effectuer de *traitement 2* lorsque la *condition* n'a pas été vérifiée ; l'algorithme passe alors à l'instruction suivante)



Algorigramme

Exemple 3 (fonction) : Ecrivons un algorithme qui calcule la racine d'un nombre, si elle existe.

Rappel : La fonction racine carrée est définie sur $\mathbb{R}^+$.

```
VARIABLES
    nb EST_DU_TYPE NOMBRE
    racine EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
    AFFICHER "Saisir un nombre : "
    LIRE nb
    AFFICHER nb
    SI (nb>=0) ALORS
        DEBUT_SI
        racine PREND_LA_VALEUR sqrt(nb)
        AFFICHER "La racine carrée du nombre est : "
        AFFICHER racine
        FIN_SI
FIN_ALGORITHME
```

Affichages après lancement de l'algorithme avec AlgoBox :

- Lorsque le nombre saisi est supérieur ou égal à 0 :

```
***Algorithme lancé***
Saisir un nombre : 121
La racine carrée du nombre est : 11
***Algorithme terminé***
```

- Lorsque le nombre saisi est strictement négatif :

```
***Algorithme lancé***
Saisir un nombre : -4

***Algorithme terminé***
```

3) STRUCTURES ALTERNATIVES IMBRIQUEES

Plusieurs structures alternatives peuvent être imbriquées, si bien que dans un traitement peut (peuvent) figurer une ou plusieurs structure(s) alternative(s).

Pour une meilleure lisibilité de l'algorithme, on utilise l'**INDENTATION**, qui consiste à écrire les instructions sur des lignes différentes en procédant à des décalages.

Exercice 4 (géométrie) : Ecrivons un algorithme qui précise si un triangle ABC est rectangle (et dans ce cas en quel point) ou s'il ne l'est pas.

Rappel : D'après la réciproque du théorème de Pythagore, si $AB^2 = AC^2 + BC^2$, alors le triangle ABC est rectangle en C.

```
VARIABLES
    xA EST_DU_TYPE NOMBRE
    yA EST_DU_TYPE NOMBRE
    xB EST_DU_TYPE NOMBRE
    yB EST_DU_TYPE NOMBRE
    xC EST_DU_TYPE NOMBRE
    yC EST_DU_TYPE NOMBRE
    ABcarre EST_DU_TYPE NOMBRE
    ACcarre EST_DU_TYPE NOMBRE
    BCcarre EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
    AFFICHER "Saisir l'abscisse du point A : "
    LIRE xA
    AFFICHER xA
    AFFICHER "Saisir l'ordonnée du point A : "
    LIRE yA
    AFFICHER yA
    AFFICHER "Saisir l'abscisse du point B : "
    LIRE xB
    AFFICHER xB
    AFFICHER "Saisir l'ordonnée du point B : "
    LIRE yB
    AFFICHER yB
    AFFICHER "Saisir l'abscisse du point C : "
    LIRE xC
    AFFICHER xC
    AFFICHER "Saisir l'ordonnée du point C : "
    LIRE yC
    AFFICHER yC
    ABcarre PREND_LA_VALEUR pow(xB-xA,2)+pow(yB-yA,2)
    ACcarre PREND_LA_VALEUR pow(xC-xA,2)+pow(yC-yA,2)
    BCcarre PREND_LA_VALEUR pow(xC-xB,2)+pow(yC-yB,2)
    SI (ABcarre+BCcarre==ACcarre) ALORS
        DEBUT_SI
        AFFICHER "Le triangle ABC est rectangle en B."
        FIN_SI
        SINON
            DEBUT_SINON
            SI (ABcarre+ACcarre==BCcarre) ALORS
                DEBUT_SI
                AFFICHER "Le triangle ABC est rectangle en A."
                FIN_SI
                SINON
                    DEBUT_SINON
                    SI (ACcarre+BCcarre==ABcarre) ALORS
                        DEBUT_SI
                        AFFICHER "Le triangle est rectangle en C."
                        FIN_SI
                        SINON
                            DEBUT_SINON
                            AFFICHER "Le triangle n'est pas rectangle."
                            FIN_SINON
                    FIN_SINON
            FIN_SINON
FIN_ALGORITHME
```

Affichages après lancement de l'algorithme avec AlgoBox :

```
***Algorithme lancé***
Saisir l'abscisse du point A : 1
Saisir l'ordonnée du point A : 1
Saisir l'abscisse du point B : 3
Saisir l'ordonnée du point B : 2
Saisir l'abscisse du point C : -1
Saisir l'ordonnée du point C : 2
Le triangle n'est pas rectangle.
***Algorithme terminé***
```

```
***Algorithme lancé***
Saisir l'abscisse du point A : -5
Saisir l'ordonnée du point A : 0
Saisir l'abscisse du point B : 0
Saisir l'ordonnée du point B : 0
Saisir l'abscisse du point C : 0
Saisir l'ordonnée du point C : 8
Le triangle ABC est rectangle en B.
***Algorithme terminé***
```

Remarques :

- Dans cet algorithme, on choisit de déclarer les variables ABcarre, ACcarre et BCcarre pour rendre plus lisible l'écriture des tests mais elles ne s'avèrent pas obligatoires.
- Les tests sont ici imbriqués : on vérifie d'abord si le triangle ABC est rectangle en B puis, s'il ne l'est pas, on vérifie s'il est rectangle en A et enfin, s'il n'est pas rectangle en C, on affiche que le triangle n'est pas rectangle. Si le triangle est, à un moment donné, rectangle en un point (cas du test vérifié), alors on affiche en quel point ABC est rectangle.
- On peut également tracer le triangle grâce aux instructions :

```
TRACER_SEGMENT (xA,yA)->(xB,yB)
TRACER_SEGMENT (xA,yA)->(xC,yC)
TRACER_SEGMENT (xB,yB)->(xC,yC)
```

Remarque : Il s'avère parfois nécessaire d'exécuter plusieurs fois de suite le même traitement, c'est-à-dire la même série d'instructions. Dans ce cas, il convient de faire appel aux **structures répétitives**.

CHAPITRE 10 : Les structures répétitives

Toutes les structures itératives répètent l'exécution de traitement(s).

Deux cas sont cependant à envisager, selon que :

- le nombre de répétitions est connu à l'avance : c'est le cas des **boucles itératives**
- le nombre de répétitions n'est pas connu ou est variable : c'est le cas des **boucles conditionnelles**

1) La structure POUR … DE … A …, FAIRE

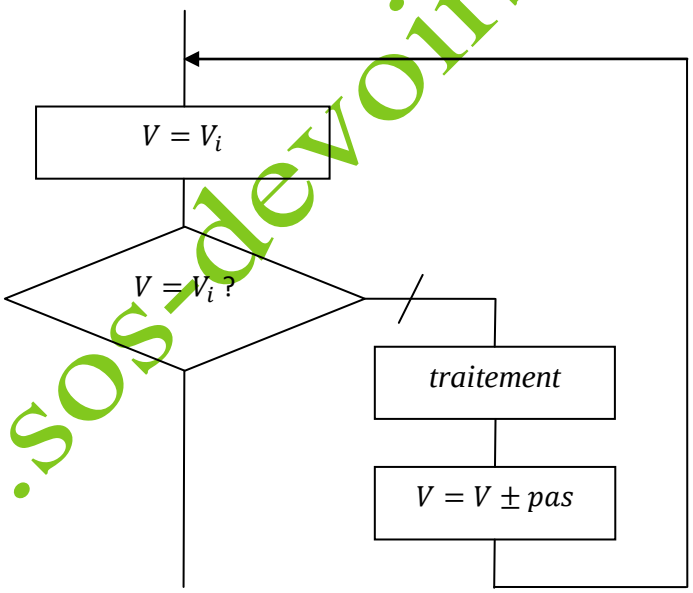
Cette structure est une **BOUCLE ITERATIVE** ; elle consiste à répéter un certain traitement un nombre de fois fixé à l'avance.

En algorithmique, on traduit cette structure itérative à l'aide des instructions :

Pour i **de** 1 **jusqu'à** N (on répète un nombre connu de fois le même traitement ; ici, de 1 à N, donc N fois)

Faire *traitement 1* (instructions à effectuer)

FinPour



Algorigramme

Remarques :

- La variable i est un **compteur**, dont la valeur augmente automatiquement de 1 à chaque tour. Cette variable permet en définitive de contrôler le nombre entier de tours. Cette variable est en d'autres termes un variable de contrôle d'itération, caractérisée par sa valeur initiale, sa valeur finale et son pas de variation.
- La **sortie de la boucle** s'effectue lorsque le nombre souhaité d'itérations est atteint, c'est-à-dire lorsque i prend la valeur N.

Exemple 1 (fonction) : Proposons un algorithme qui calcule le carré de tous les entiers compris entre 1 et 50.

```
VARIABLES
    nb EST_DU_TYPE NOMBRE
    cube EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
    POUR nb ALLANT_DE 1 A 50
        DEBUT_POUR
        AFFICHER "Le nombre "
        AFFICHER nb
        AFFICHER " a pour cube : "
        cube PREND_LA_VALEUR pow(nb,3)
        AFFICHER cube
        FIN_POUR
FIN_ALGORITHME
```

Affichage après lancement de l'algorithme avec AlgoBox :

```
***Algorithme lancé***
Le nombre 1 a pour cube : 1
Le nombre 2 a pour cube : 8
Le nombre 3 a pour cube : 27
Le nombre 4 a pour cube : 64
Le nombre 5 a pour cube : 125
Le nombre 6 a pour cube : 216
Le nombre 7 a pour cube : 343
Le nombre 8 a pour cube : 512
Le nombre 9 a pour cube : 729
Le nombre 10 a pour cube : 1000
...
Le nombre 42 a pour cube : 74088
Le nombre 43 a pour cube : 79507
Le nombre 44 a pour cube : 85184
Le nombre 45 a pour cube : 91125
Le nombre 46 a pour cube : 97336
Le nombre 47 a pour cube : 103823
Le nombre 48 a pour cube : 110592
Le nombre 49 a pour cube : 117649
Le nombre 50 a pour cube : 125000

***Algorithme terminé***
```

Remarque : A chaque tour de boucle, on affiche la locution « Le nombre » suivie de la valeur du nombre nb, suivie de la locution « a pour cube », suivie de la valeur du cube du nombre, à savoir la valeur de la variable cube.

Exemple 2 (numérique) : Ecrivons un algorithme permettant de calculer la somme des entiers de 1 à N, le nombre N étant renseigné par l'utilisateur.

```
VARIABLES
    somme EST_DU_TYPE NOMBRE
    i EST_DU_TYPE NOMBRE
    N EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
    somme PREND_LA_VALEUR 0
    LIRE N
    POUR i ALLANT_DE 1 A N
        DEBUT_POUR
        somme PREND_LA_VALEUR somme+i
        FIN_POUR
    AFFICHER "La somme est égale à : "
    AFFICHER somme
FIN_ALGORITHME
```

Affichage après lancement de l'algorithme avec AlgoBox avec N=30 :

```
***Algorithme lancé***
La somme est égale à 465
***Algorithme terminé***
```

Remarque : A chaque tour de boucle, la somme est affectée de la somme et du nombre.

Exemple 3 (probabilités) : Proposons un algorithme qui simule le lancer d'un dé cubique équilibré un certain nombre de fois et affiche le nombre d'apparitions de chaque face.

```
VARIABLES
    a EST_DU_TYPE NOMBRE
    i EST_DU_TYPE NOMBRE
    Compteur EST_DU_TYPE LISTE
    Texte EST_DU_TYPE CHAINE
    nb_lancers EST_DU_TYPE NOMBRE
```

Affichage après lancement de l'algorithme avec AlgoBox :

```
***Algorithme lancé***
Nombre de lancers désirés ?
Nombre de 1 : 16
Nombre de 2 : 15
Nombre de 3 : 10
Nombre de 4 : 17
Nombre de 5 : 18
Nombre de 6 : 24

***Algorithme terminé***
```

```
DEBUT_ALGORITHME
    AFFICHER "Nombre de lancers désirés ?"
    LIRE nb_lancers
    POUR i ALLANT_DE 1 A 6
        DEBUT_POUR
        Compteur[i] PREND_LA_VALEUR 0
        FIN_POUR
    POUR i ALLANT_DE 1 A nb_lancers
        DEBUT_POUR
        a PREND_LA_VALEUR 1+floor(random()*6)
        Compteur[a] PREND_LA_VALEUR Compteur[a]+1
        FIN_POUR
    POUR i ALLANT_DE 1 A 6
        DEBUT_POUR
        Texte PREND_LA_VALEUR "Nombre de "+i+" : "+Compteur[i]
        AFFICHER Texte
        FIN_POUR
FIN_ALGORITHME
```

Remarques :

- L'utilisateur est invité à renseigner le nombre de fois qu'il souhaite lancer le dé à 6 faces. La valeur de cette variable « nb_lancers » est lue.
- « floor(x) » est la commande qui permet de donner la partie entière de x.
- « random() » est la commande qui permet de choisir un nombre entre 0 et 1.
- Cet algorithme propose un nouveau type de variable, le type liste.

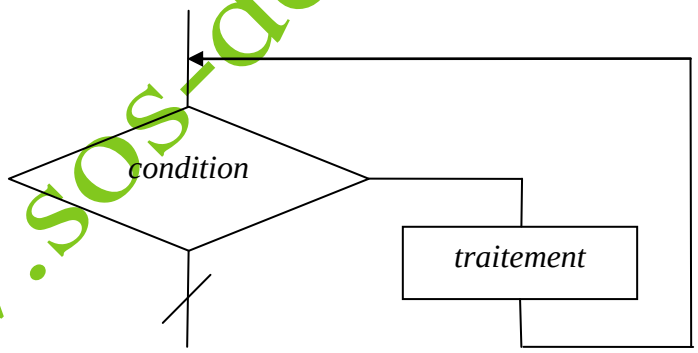
2) La structure TANT QUE …, FAIRE

Parfois, pour réaliser une tâche, on doit effectuer plusieurs fois les mêmes instructions, sans que le nombre de fois soit déterminé à l'avance. On utilise alors une **BOUCLE CONDITIONNELLE**. Dans cette structure, le même traitement est effectué tant qu'une condition reste valide ; la boucle s'arrête quand celle-ci n'est plus remplie. Cette **structure répétitive** est ainsi formulée :

Tant que *condition* (on répète un nombre inconnu de fois le même traitement, autant de fois que la condition est vérifiée)

    Faire *traitement* (instructions à effectuer)

FinTant



Algorigramme

Remarques :

- Le nombre de répétitions dépendra de la *condition*.
- Si la condition n'est pas vérifiée au début, alors le *traitement 1* ne sera pas exécuté du tout.
- Si la *condition* est vérifiée au début et si la *condition* n'est pas susceptible d'être modifiée lors du *traitement 1*, alors le *traitement 1* sera indéfiniment exécuté et l'utilisateur sera contraint d'arrêter le programme. Dans ce cas, il s'agit d'une erreur majeure car un programme ne doit pas boucler indéfiniment mais au contraire s'arrêter automatiquement une fois que la *condition* cesse d'être vérifiée.

Exemple 1 (fonction) : Proposons un algorithme avec AlgoBox qui permet d'afficher tous les nombres entiers naturels dont le cube est inférieur à 5000.

```
VARIABLES
    nb EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
    nb PREND_LA_VALEUR 0
    AFFICHER "La liste des entiers naturels dont le cube est inférieur à 5000 est : "
    TANT_QUE (pow(nb,3)<=5000) FAIRE
        DEBUT_TANT_QUE
        AFFICHER nb
        AFFICHER " "
        nb PREND_LA_VALEUR nb+1
        FIN_TANT_QUE
FIN_ALGORITHME
```

Remarques :

- On commence par initialiser nb.
- Au début du premier passage dans la boucle, nb vaut donc 0. A la fin du premier passage et au début du second, nb vaut nb+1, c'est-à-dire 1. Dès que $nb^3>5000$, la boucle s'arrête. Il ne faut pas oublier l'instruction « nb prend la valeur nb+1 », faute de quoi la boucle est infinie.

Affichage après lancement de l'algorithme avec AlgoBox :

```
***Algorithme lancé***
La liste des entiers naturels dont le cube est inférieur à 5000 est :
0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17
***Algorithme terminé***
```

Exemple 2 (numérique) : Ecrivons un algorithme avec AlgoBox permettant de calculer le quotient de deux entiers positifs dans une division euclidienne.

```
VARIABLES
    dividende EST_DU_TYPE NOMBRE
    diviseur EST_DU_TYPE NOMBRE
    quotient EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
    AFFICHER "Saisir le dividende : "
    LIRE dividende
    AFFICHER dividende
    AFFICHER "Saisir le diviseur : "
    LIRE diviseur
    AFFICHER diviseur
    TANT_QUE (dividende-diviseur*(quotient+1)>=0) FAIRE
        DEBUT_TANT_QUE
        quotient PREND_LA_VALEUR quotient+1
        FIN_TANT_QUE
    AFFICHER "Le quotient est : "
    AFFICHER quotient
FIN_ALGORITHME
```

Remarques :

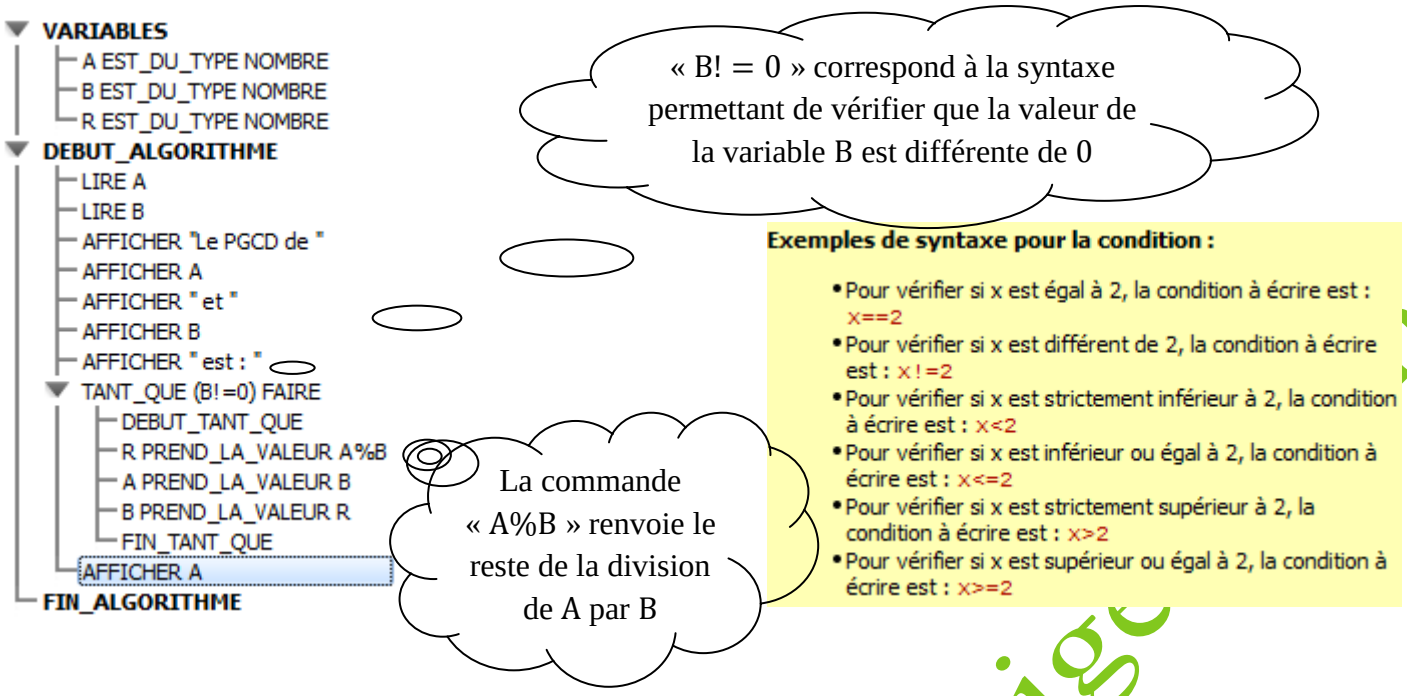
- A deux entiers naturels *dividende* et *diviseur*, avec diviseur non nul, la division euclidienne associe un *quotient* et un *reste*, tous deux entiers naturels vérifiant :

$$\begin{cases} dividende = diviseur \times quotient + reste \\ reste < diviseur \end{cases}$$

- Cet algorithme n'exclut pas la saisie de valeurs négatives, qui invalideraient le résultat final. Un message informant l'utilisateur serait à prévoir pour qu'il ne saisisse que des valeurs entières positives.

Exemple 3 (numérique) : Ecrivons un algorithme avec AlgoBox permettant de déterminer le PGCD de deux nombres entiers.

Rappel : En arithmétique, le plus grand commun diviseur (PGCD) de deux nombres entiers naturels est le plus grand entier naturel qui divise simultanément ces deux entiers.



3) La structure REPETER … JUSQU'A …

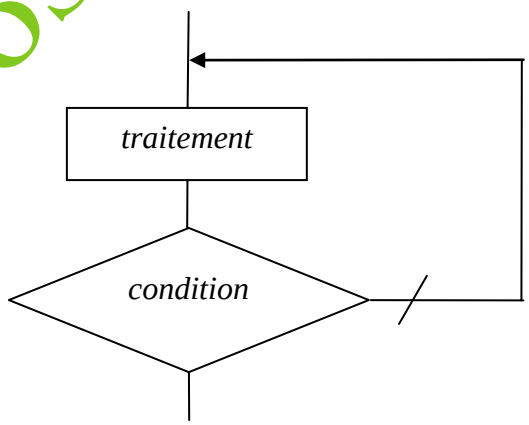
Une variante de structure répétitive avec **BOUCLE CONDITIONNELLE** consiste à répéter un traitement jusqu'à ce qu'une certaine condition soit vérifiée. On la traduit par l'instruction :

Répète (on répète un nombre inconnu de fois le même traitement, autant de fois que la condition est vérifiée)

traitement (instructions à effectuer)

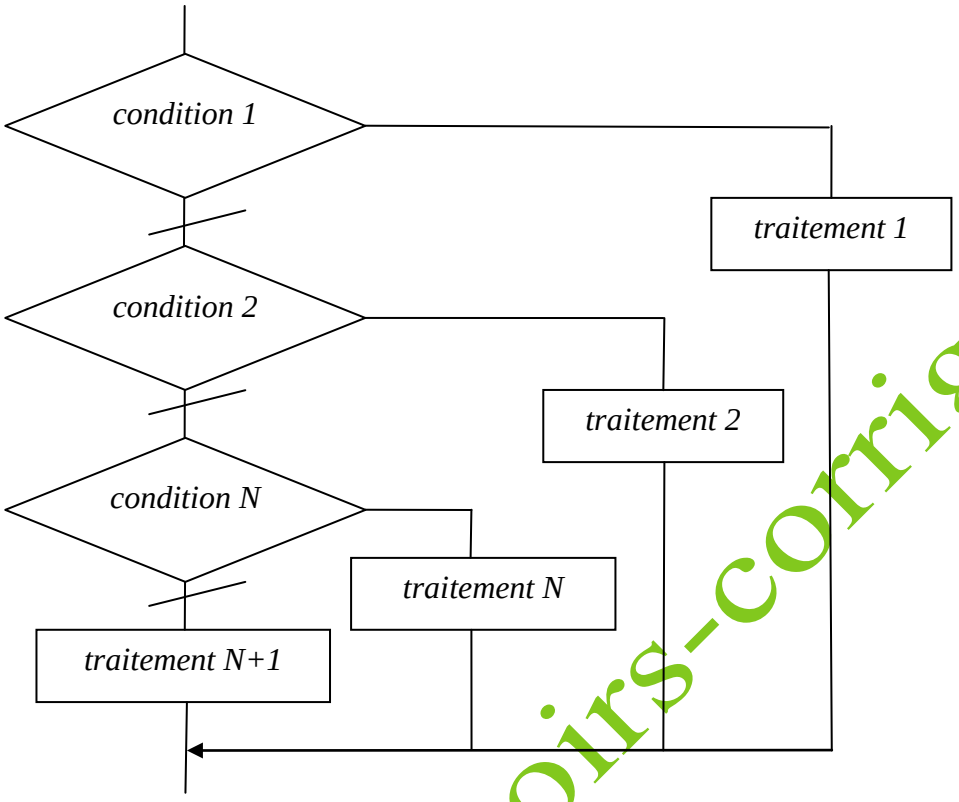
Jusqu'à *condition*

Remarque : Dans ce type d'instruction, le test est effectué à la fin de la boucle, si bien que le *traitement* est exécuté au moins une fois, que la *condition* soit ou non vérifiée au début.



Algorigramme

La structure de choix permet, en fonction de plusieurs conditions de type booléen, d'effectuer des traitements spécifiques.



Algorigramme

Exemple (numérique) : Proposons un algorithme qui choisit de manière aléatoire un nombre entier compris entre 1 et 6 et précise si ce nombre est pair ou impair, premier ou pas.

```
VARIABLES
    nb EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
    AFFICHER "Le programme va afficher un nombre entre 1 et 6, aléatoirement : "
    nb PREND_LA_VALEUR floor(6*random()+1)
    AFFICHER nb
    SI (nb==1) ALORS
        DEBUT_SI
        AFFICHER "Ce nombre est pair mais n'est pas premier."
        FIN_SI
    SI (nb==2) ALORS
        DEBUT_SI
        AFFICHER "Ce nombre est premier et pair."
        FIN_SI
    SI (nb==3 OU nb==5) ALORS
        DEBUT_SI
        AFFICHER "Ce nombre est premier et impair."
        FIN_SI
    SI (nb==4 ou nb==6) ALORS
        DEBUT_SI
        AFFICHER "Ce nombre est pair mais n'est pas premier."
        FIN_SI
FIN_ALGORITHME
```

Rappel :

Un nombre entier est premier si, et seulement si, il a pour uniques diviseurs 1 et lui-même. Les nombres 2, 3, 5 et 7 sont les seuls nombres premiers inférieurs à 10.

Remarque : 2 est le seul nombre premier pair.

```
***Algorithme lancé***
Le programme va afficher un nombre entre 1 et 6, aléatoirement : 4
Ce nombre est pair mais n'est pas premier.
***Algorithme terminé***
```

```
***Algorithme lancé***
Le programme va afficher un nombre entre 1 et 6, aléatoirement : 2
Ce nombre est premier et pair.
***Algorithme terminé***
```

```
***Algorithme lancé***
Le programme va afficher un nombre entre 1 et 6, aléatoirement : 3
Ce nombre est premier et impair.
***Algorithme terminé***
```