ALGORITHMIQUE

“Voulez-vous apprendre les sciences avec facilité ? Commencez par apprendre votre langue.” Etienne de Condillac

"La plus grande attention doit être portée à la compréhension du problème, faute de quoi l'algorithme n'a aucune chance d'être correct". Denis Lapoire

"C'est toujours l'impatience de gagner qui fait perdre", Louis XIV cité dans "L'immortel" de FOG.

J'écoute et j'oublie.
Je lis et je retiens.
Je fais et j'apprends.
(Proverbe chinois)

"La langue est la meilleure et la pire des choses", Esope.

TABLE DES MATIERES

1 - INTRODUCTION

Penser aux gobelets (les variables) et aux petits bouts de papier, why not des mini postit, (les valeurs) ou bien encore des perles.

1.1 - Comment faire une omelette en cassant des œufs ?

Pour obtenir une omelette il faut des œufs, du sel, du poivre et de l'huile d'**olive** (des ingrédients).

Les ingrédients sont les **variables** ou les **paramètres**.

Mais il faut aussi un bol ou une assiette creuse, une fourchette ou un fouet, une poêle, un réchaud, etc (des ustensiles).

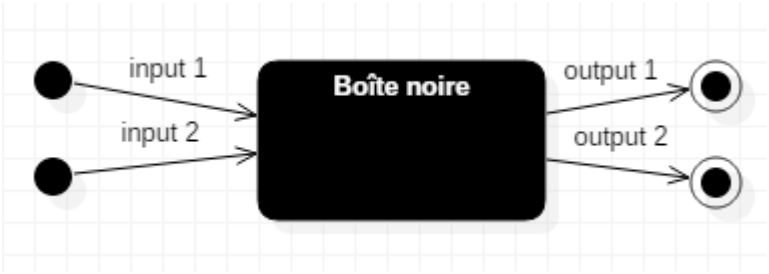
Les ustensiles informatiques sont les **instructions** et les **fonctions**.

Et aussi une recette (un mode opératoire, un **ordonnancement**).

En somme pour obtenir un PLAT il faut des INGREDIENTS, des USTENSILES et un MODE OPERATOIRE.

Dans un algorithme c'est la même chose : pour obtenir un ou plusieurs OUTPUTS il faut un ou plusieurs INPUTS, une série d'instructions et un mode opératoire.

La boîte noire ! Qu'il faudra blanchir !



1.2 - DÉFINITION

Tout algorithme est caractérisé par :

- ✔ des données en entrée,
- ✔ le contenu de chaque étape (opération),
- ✔ l'ordre de succession des différentes étapes,
- ✔ éventuellement l'existence de conditions déterminant l'exécution ou la non exécution de certaines étapes,
- ✔ des données en sortie.

Un algorithme est équivalent à R = f(D). Avec R pour résultats, D pour données et f() pour fonction.

1.3 - OBJECTIF DE CE SUPPORT

Introduire à la programmation impérative (structurée et procédurale).
Pensez que jusqu'à présent vous avez peut-être vu 2 ou 3 langages : HTML et CSS, programmation déclarative/descriptive et SQL, programmation « algébrique ».
Le paradigme de la programmation impérative est le suivant : on s'intéresse aux actions. Un algorithme, un programme, décrit les opérations en séquences d'instructions exécutées par un ordinateur pour modifier l'état du programme (l'état de la mémoire).

http://fr.wikipedia.org/wiki/Programmation_imp%C3%A9rative

Les instructions impératives de base sont :

- ✔ la déclaration des variables.
- ✔ l'affectation : assigner une valeur ou le résultat d'une expression à une variable,
- ✔ le branchement conditionnel : exécuter une opération si une condition prédéfinie est remplie,
- ✔ le branchement sans condition : appeler une procédure ou une fonction (un bloc d'instructions nommé),
- ✔ l'itération : exécuter plusieurs fois la même opération (synonymes : répétition, boucle).

Note :
chaque mot de cette page doit être « impérativement » :-) compris ! Pas nécessairement tout de suite puisque l'apprentissage de ces notions est l'objet de ce support de cours.
En revanche une fois lecture faite et pratique réalisées ces notions doivent êtes plus-que-parfaitement intégrées pour pouvoir suivre avec profit la suite.

Les principaux chapitres sont les suivants :

- ✔ introduction,
- ✔ analyse descendante (du macroscopique au microscopique),
- ✔ données, types et opérateurs,
- ✔ actions élémentaires (affectation, lecture, écriture),
- ✔ structures algorithmiques (séquentielle, conditionnelle, itérative),
- ✔ test des algorithmes.

Des notions avancées (récursivité, tris, graphes, arbres, complexité des algorithmes, …) sont abordées dans un autre document.

L'introduction à la POO (Programmation Orientée Objet) est abordée dans un autre document.

1.4 - BIBLIOGRAPHIE

"Mini manuel d'algorithmique et de programmation"
DUT, L1/L2, école d'ingénieurs - Cours + exos corrigés
Auteur : Vincent Granet
Editeur : Dunod
175 pages
EAN13 : 978-2-10-057350-9
20 euros.
Note : exemples en langage C.

"Algorithmique et programmation en Java"
Cours et exercices corrigés - IUT, IUP, Master, écoles d'ingénieurs
Auteur : Vincent Granet
Editeur : Dunod
396 pages
EAN13 : 978-2-10-054532-2
30 euros.
Note : difficile

"INFORMATIQUE - Premier pas en algorithmique"
De l'énoncé à la solution, approche par l'expérimentation - Exercices analysés, corrigés et commentés (niveau A)
Auteurs : Tartier Annie, Vailly Alain
Editeur : Ellipses
408 pages
ISBN : 9782340001251
25 euros
Note : pseudo-langage et de nombreux schémas.

1.5 - RÉFÉRENCES WEB

http://www.pise.info/algo/codage.htm
Simple et efficace.

http://www.ac-nancy-metz.fr/eco-gestion/eric_crepin/algo/exercices/presentation.htm
De nombreux exercices.

http://lwh.free.fr/
Simple et illustré.

http://lapoire.developpez.com/algorithmique/initiation/
Initiation pour scientifiques.

http://algo.developpez.com/cours/#intro
Le choix ...

Un site avec cours et exercices « en live » (pascal12345/ibidem)
http://www.codecademy.com/?locale_code=fr

Autres : votre ami Gogol !

1.6 - CONSIDÉRATIONS GÉNÉRALES

Le **langage utilisé** dans ce support **est artificiel**, propre à ce cours et non utilisable directement par un ordinateur. Il se rapproche du Pascal.

Les algorithmes sont aussi représentés par des algorigrammes.

Un algorithme doit être :

- ✔ correct,
- ✔ lisible,
- ✔ de faible complexité.

Correction : l'algorithme doit résoudre le problème posé.

Lisibilité :

Certaines règles permettent une écriture plus facile et donc une maintenance aisée.

- la **simplicité**.

- la première est d'utiliser des **désignations significatives**. Si une variable doit servir pour mémoriser un nom appelons-la nom et non pas N … il en est de même pour les noms des procédures et des fonctions.

- La deuxième règle est **d'insérer des commentaires** pour permettre une meilleure lecture des algorithmes. La ligne comportant un commentaire commencera par REM ou un astérisque (*).

- La troisième règle est l'**indentation** c'est-à-dire qu'après une instruction conditionnelle ou itérative on décalera de quelques colonnes pour commencer l'écriture de la ligne suivante.

Complexité : dans le temps et dans l'espace ; la solution est une optimisation de l'utilisation des ressources. Le calcul élaboré de la complexité des algorithmes est étudié dans le cours algorithmique avancée.
Pour l'instant considérons que dans l'espace le calcul de la complexité est la somme des octets des variables utilisées. En algorithmique c'est ?
L'unité de calcul pour le temps est l'affectation et le test.

L'écriture d'un algorithme nécessite du papier, un crayon, une gomme, du sens de l'observation, de la rigueur, de la précision, de l'intuition, …

1.7 - UNE PREMIÈRE TERNAIRE : CONCEPTION, RÉALISATION, TEST

La règle des 3 tiers !!!

1/3 du temps pour l'analyse et la conception,
1/3 du temps pour le codage, la réalisation,
1/3 du temps pour les tests.

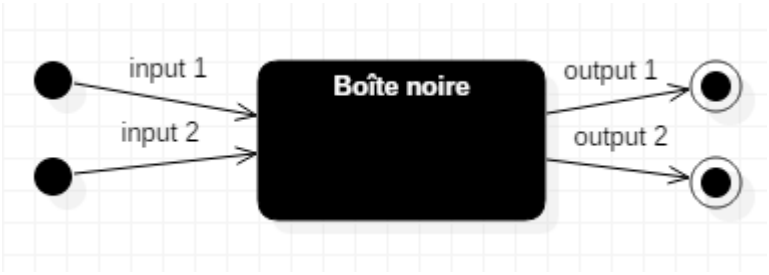
1.8 - Une autre ternaire : entrée(s), calcul(s), sortie(s)

Les ordinateurs servent à traiter et stocker des données.

La structure basique d'un algorithme, d'un programme, est la suivante :

INPUTS → BN → OUTPUTS

La boîte noire ! Qu'il faudra blanchir !



2 - L'ANALYSE POUR L'ALGORITHMIQUE

2.1.1 - Principe de l'analyse descendante

Du macroscopique au microscopique : l'art de la décomposition.

Pour passer de l'énoncé initial du problème à l'algorithme il faut analyser le problème.

Les principes de l'analyse sont les suivants :

Partir du **résultat final**. Le résultat final est ce que l'on connaît le mieux puisqu'il fait partie de l'énoncé du problème.

Définir le résultat final en introduisant des **résultats intermédiaires** si cela est nécessaire.
Ainsi on décompose le problème initial en problèmes plus simples.
Définir saura déterminer si le résultat est **une donnée** (non décomposable, non calculable) ou bien le résultat d'un calcul.

Définir chaque résultat intermédiaire, en introduisant éventuellement de nouveaux intermédiaires.

S'arrêter quand tous les résultats intermédiaires sont définis (non décomposables).

2.1.2 - Exemple : établir une fiche de paie

Le résultat attendu est le suivant : afficher le nom et le salaire net d'un salarié à partir d'un nom, d'un salaire brut et d'un taux de retenues.

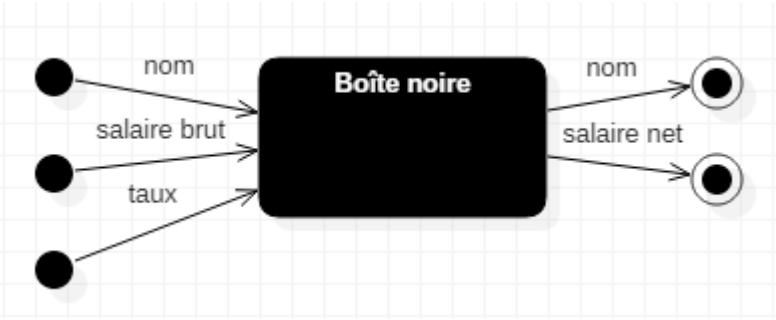
Le nom est une donnée.
Le salaire brut est une donnée.
Le taux est une donnée.
Les retenues sont calculées : salaire brut multiplié par un taux.
Le salaire net est calculé : salaire brut moins les retenues.

Les données en entrée sont :

- ✔ nom
- ✔ salaire brut
- ✔ taux

Les données en sortie sont :

- ✔ nom
- ✔ salaire net



Les identificateurs et leur type :

- ✔ nom : CHAINE
- ✔ salaireBrut : REEL
- ✔ taux : REEL
- ✔ retenues : REEL
- ✔ salaireNet : REEL

L'algorithme :

```
VAR
     nom : CHAINE
     salaireNet, salaireBrut, taux, retenues : REEL

DEBUT
     REM Saisies
     ECRIRE "Saisir nom : "
     LIRE nom
     ECRIRE "Saisir salaire brut : "
     LIRE salaireBrut
     ECRIRE "Saisir taux : "
     LIRE taux

     REM Traitement/Calcul
     retenues <- salaireBrut * taux
     salaireNet <- salaireBrut - retenues

     REM Emission des résultats
     ECRIRE "Nom : " & nom
     ECRIRE "Salaire net : "
     ECRIRE salaireNet
FIN
```

2.2 - ARCHÉTYPE D'UN ALGORITHME

Un algorithme sera décomposé en 2 grandes parties :

- ✓ une partie pour la déclaration des variables introduite par le mot-clé VAR,
- ✓ une partie pour la série des instructions, encadrée des mots-clés DEBUT et FIN. Cette deuxième partie est décomposée en 3 parties : récupération des données en entrée, traitement/calcul sur les données, restitution du résultat.

Les commentaires devront être présents. Ils seront précédés du mot-clé REM.

```
VAR
     REM Déclaration des variables


DEBUT

     REM Instructions pour la récupération des entrées, des saisies

     REM Instructions de calculs (traitements)

     REM Instructions pour la restitution du résultat

FIN
```

3 - LES DONNÉES, LES TYPES ET LES OPÉRATEURS

3.1 - Les variables

Une donnée est une information simple (une valeur) ou complexe (plusieurs valeurs dans une structure ou un objet).

Un type est un ensemble de valeurs et d'opérations sur ces valeurs.

Principaux types :

Type	Description	Opérations (liste non exhaustive)
ENTIER	Numérique sans partie décimale	Arithmétiques
REEL	Numérique avec une partie décimale	Arithmétiques
BOOLEEN	Une variable avec 2 valeurs possibles : VRAI ou FAUX	
CAR	L'ensemble des caractères	Concaténation
CHAINE	L'ensemble des textes possibles	Concaténation, extraction, ...
TAB	Un tableau : une structure homogène	
ENREGISTREMENT	Une structure hétérogène	

Tous les types permettent les opérations suivantes : affectation, comparaison.

Une variable est un espace mémoire.
Une variable est un triplet : nom, type, valeur.

La syntaxe est la suivante :

```
VAR
    identificateur1 : TYPE
    identificateur2 : TYPE
```

Exemple :

```
VAR
    idDeLaVille, idDuPays : ENTIER
    nomDelaVille : CHAINE
```

Il existe 2 conventions pour le nommage des variables : l'"underscorisation" (snake_case) ou la "camélisation" (camelCase).
Dans tous les cas les noms des variables doivent être sémantiquement significatifs.
Dans tous les cas les noms des variables ne comportent ni accents, ni espaces, ni caractères « exotiques », que des lettres non accentuées.

3.2 - LES CONSTANTES

Déclarer les constantes c'est établir la liste des constantes que l'algorithme va manipuler.
Pour chaque constante on précisera le type et la valeur.
Par convention les noms des constantes sont en majuscules et lorsque le nom est composé les mots sont séparés par un underscore (_) – tiret de soulignement -.

Cette partie déclarative se trouvera en tête de l'algorithme et sera précédée du mot CONST.

La syntaxe est la suivante :

```
CONST
      IDENTIFICATEUR_1 : type1 = valeur1
      IDENTIFICATEUR_2 : type2 = valeur2
```

Exemples :

```
CONST
     PI           : REEL   = 3,14
     ZERO         : ENTIER = 0
     TAUX_DE_TVA : REEL   = 19,6
     FRANCE : CHAINE = 'FR'
```

3.3 - LES PRINCIPAUX OPÉRATEURS

Catégorie	Opérateurs
Arithmétiques	+, -, *, : (division réelle), / (division entière), mod
Comparaison	=, <, <=, >, >= , <>
Logiques	ET, OU, NON, OUX
Autres	<- (affectation) , & (concaténation)

mod : le modulo est le reste de la division entière.

4 - ALGORITHMES ÉLÉMENTAIRES

4.1 - LES PRINCIPALES ACTIONS

- ✓ La lecture (récupération des INPUTS : d'un périphérique vers l'Unité Centrale),
- ✓ L'affectation (avec un calcul ou sans),
- ✓ L'appel d'une fonction pré-définie (du pseudo langage, du langage),
- ✓ L'appel d'une fonction définie (par vous, par un collègue, ...),
- ✓ L'écriture (traitement des OUTPUTS : de l'Unité Centrale vers un périphérique).

4.1.1 - L'affectation

Définition :

Cette opération consiste à affecter une valeur ou une expression à une variable. La valeur ou l'expression et la variable doivent être du même type.
On symbolise l'affectation par une flèche orientée à gauche.
La valeur (une constante ou la valeur d'une variable ou une expression) qui est à droite est affectée comme valeur à la variable qui est à gauche dans l'expression.

Exemples :

```
indice     <- 1
message    <- "Fichier inexistant"
compteur   <- compteur + 1
tableau[4] <- 30
test       <- faux
```

4.1.2 - La lecture

Définition :

L'ordre de lecture - LIRE - signifie qu'une donnée, située sur un périphérique d'entrée (clavier, souris, fichier, BD, …) est prête à être lue et que la valeur de cette donnée sera affectée à la variable.

Périphérique → Programme (mémoire vive).

Exemples :

```
LIRE nom
```

Le périphérique est le clavier; cela signifie qu'à la fin de la saisie la variable **nom** aura comme valeur la suite de caractères saisie au clavier.

```
LIRE fiche
```

Les données de l'enregistrement actif du fichier actif sont transférées dans la variable fiche.

Note : l'argument de LIRE ne peut être qu'une variable, jamais une constante.

4.1.3 - L'écriture

Définition :

L'ordre d'écriture - ECRIRE - est l'opération inverse.
La valeur de la variable sera affectée à un périphérique de sortie (écran, Fichier, BD, …).

Programme (mémoire vive) → Périphérique.

Si le périphérique est l'écran ECRIRE est synonyme d' AFFICHER.
Si la sortie est un fichier ou une BD et donc que le périphérique est un disque dur il s'agit bien d'une écriture.

Exemples :

```
ECRIRE nom
```

Le périphérique est l'écran et cela signifie afficher la valeur de la variable nom.

```
ECRIRE Saisir nom
```

Le périphérique est l'écran et cela signifie afficher le libellé « Saisir âge ».

```
ECRIRE fiche
```

Toutes les données de la variable fiche sont transférées de la mémoire vers le disque.

Note : l'argument de ECRIRE peut être une variable ou une constante ou une constante littérale.

4.2 - Découpage de l'algorithme

3 actions élémentaires incitent à découper l'algorithme en 3 parties :

traitement des entrées,
calculs,
traitement des sorties.

Nous retrouvons la première ternaire.

Note : les affectations de la partie « calcul » peuvent être des affectations simples (on affecte la valeur 1 à la variable i), des affectations de résultats d'expression (on affecte le résultat d'une addition à la variable somme), etc.

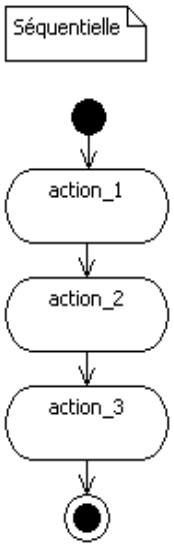
4.3 - EXEMPLE : L'ADDITION DE 2 ENTIERS

L'addition de 2 nombres

```
VAR
     nombre1,
     nombre2,
     resultat : ENTIER

DEBUT
     REM Saisies
     ECRIRE "Addition de deux nombres"

     ECRIRE "Saisir premier nombre : "
     LIRE nombre1

     ECRIRE "Saisir deuxième nombre : "
     LIRE nombre2

     REM Calcul/Traitement
     resultat <- nombre1 + nombre2

     REM Émission du résultat
     ECRIRE "Résultat : " & resultat
FIN
```

Boîte noire

Ordinogramme



4.4 - EXERCICE : LA PERMUTATION

La permutation de 2 chaînes de caractères !!! BN + Algorithme.

5 - LES TROIS PRINCIPALES STRUCTURES DES ALGORITHMES

5.1 - RÉSUMÉ

Il existe trois structures de base (encore une ternaire) en programmation :

- ✓ La structure séquentielle quand les actions sont exécutées dans le même ordre les unes après les autres,
- ✓ La structure conditionnelle quand l'action ou les actions sont exécutées si une condition est satisfaite,
- ✓ La structure itérative quand l'action ou les actions sont exécutées plusieurs fois.

5.2 - La structure alternative

5.2.1 - Définition

Dans de nombreux cas une action ou un bloc d'actions ne sera exécuté que sous une certaine condition. La condition pouvant être simple ou complexe.

Le choix peut être entre deux possibilités ou N possibilités.

Je veux aller à « Gare du Nord » à partir de « Nation ».
Que se passe-t-il SI la ligne 2 est en panne ?

5.2.2 - L'alternative à deux branches

5.2.2.1 **- Première Syntaxe** : SI ...

SI condition **ALORS**
    action-1
    [action-2]
FIN-SI

Si la condition est remplie les instructions 1 et 2 seront exécutées et si la condition n'est pas remplie aucune instruction ne sera exécutée.

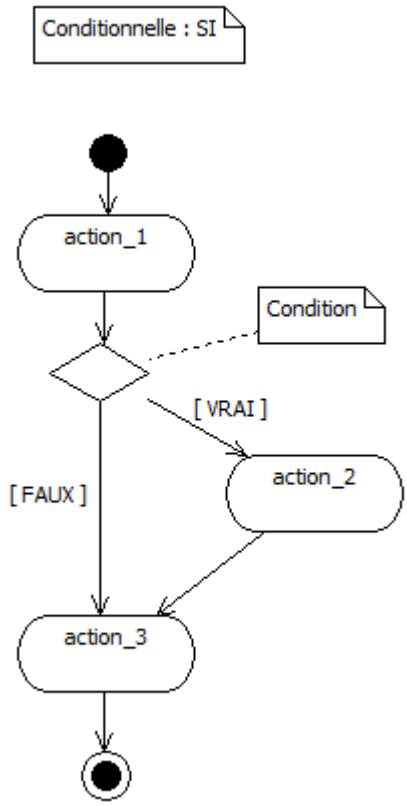
Par exemple si le total d'une facture est inférieur à 100 francs on ajoute 15 francs de frais de transport.

Exemple :

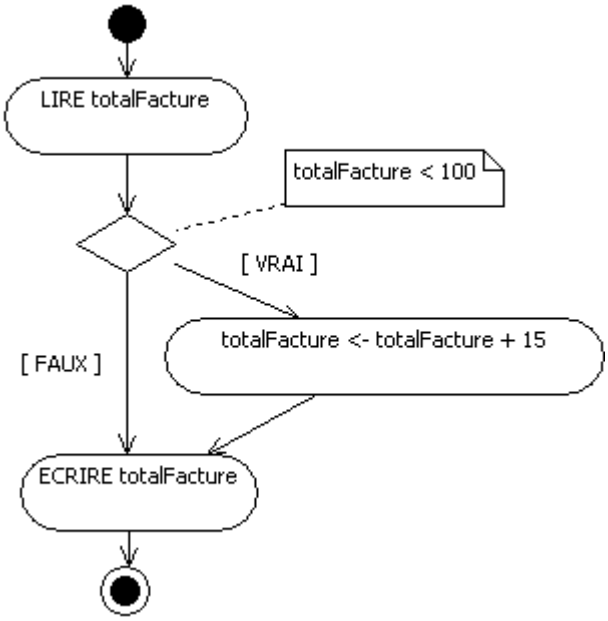
```
SI totalFacture < 100 ALORS
    totalFacture <- totalFacture + 15
FIN-SI
```

Remarque : dans le cas où il y a une seule instruction on peut omettre le FIN-SI.

5.2.2.2 - Algorigramme



Exemple : Si le total de la facture est inférieur à 100 on ajoute 15 francs de frais de port.



5.2.2.3 **- Deuxième Syntaxe** : SI ... SINON ...

SI condition **ALORS**
    instruction-1
    [instruction-2]
SINON
    instruction-3
    [instruction-4]
FIN-SI

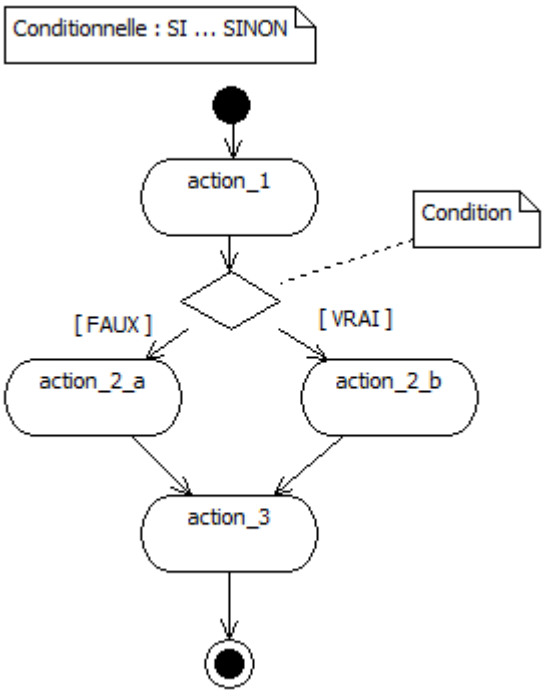
Exemple :

```
SI resultat >= 0 ALORS
     ECRIRE "Il s'agit d'un gain"
SINON
     ECRIRE "Il s'agit d'une perte"
FIN-SI
```

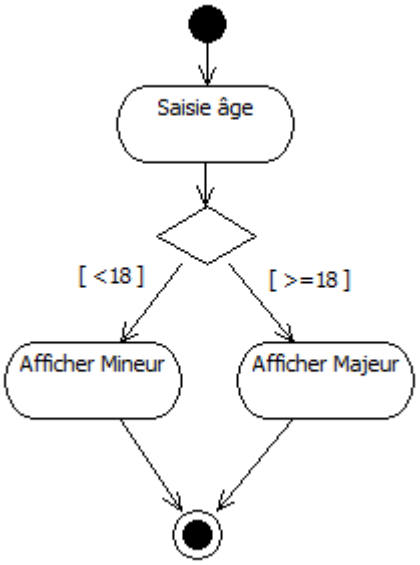
Remarque : dans le cas où il y a une seule instruction on peut omettre le FIN-SI.

.../...

5.2.2.4 - Algorigramme générique et spécifique



Exemple



Exercice : traduisez en pseudo-langage !

5.2.2.5 - Les conditions complexes

Définition :

Une condition complexe est la combinaison de plusieurs conditions simples.
Les opérateurs logiques - ET, OU inclusif, NON, OU exclusif - sont utilisés.

Si l'on doit combiner des conditions simples et des opérateurs logiques on utilisera des parenthèses.

C1	C2	ET (AND)	OU (OR)	NON (NOT) C1	OUX (XOR)
Vrai	Vrai	Vrai	Vrai	Faux	Faux
Vrai	Faux	Faux	Vrai	Faux	Vrai
Faux	Vrai	Faux	Vrai	Vrai	Vrai
Faux	Faux	Faux	Faux	Vrai	FAUX

ET : Le résultat est VRAI si les opérandes C1 et C2 sont VRAI.
OU : Le résultat est VRAI si un des opérandes C1 et C2 est VRAI.
NON : Le résultat est VRAI l'opérande C1 FAUX.
OUX : Le résultat est VRAI si un et un seul des opérandes C1 et C2 est VRAI.

Exemples :

SI le pseudo est OK **ET** le mot de passe est OK alors l'authentification est OK.
SI le pays est France **OU** le pays est FRANCE alors on affiche.
SI le pays n' **EST PAS** la France alors ...

Exemple : **ET logique**

Règles :

Si le total de la facture est inférieur à 1000, pas de remise.
Si le total de la facture est compris entre 1000 et 2000, la remise est de 5%.
Si le total de la facture est supérieur à 2000, la remise est de 10%.

```
VAR
     totalFacture : REEL

DEBUT
     REM Saisie
     ECRIRE "Saisir le total de la facture : "
     LIRE totalFacture

     REM Calcul/Traitement
     SI(totalFacture >= 1000) ET (totalFacture <= 2000) ALORS
          totalFacture <- totalFacture * 0,95
     FIN-SI

     SI totalFacture > 2000 ALORS
          totalFacture <- totalFacture * 0,90
     FIN-SI

     REM Emission du résultat
     ECRIRE "Total de la facture après une éventuelle remise : " &
totalFacture
FIN
```

Exemple : **OU logique INCLUSIF**

L'utilisateur peut faire une addition ou une soustraction et rien d'autre.

```
VAR
     x, y, r : ENTIER
     operateur : CAR
DEBUT
     ECRIRE "Saisir X : "
     LIRE x
     ECRIRE "Saisir opérateur (+ ou -) : "
     LIRE op
     ECRIRE "Saisir Y : "
     LIRE y

     SI (op = "+") OU (op = "-") ALORS
          SI op = "+" ALORS r <- x + y
          SINON r <- x - y
          ECRIRE "RESULTAT" & r
     SINON
          ECRIRE "ERREUR DE SAISIE DE L'OPERATEUR"
     FIN-SI
FIN
```

5.2.2.6 - SI imbriqués

A l'intérieur d'un SI il peut y avoir un autre SI

Syntaxes :

```
SI condition-1 ALORS
        SI condition-2 ALORS
                instruction-1
                [instruction-2]
        SINON
                instruction-2
                [instruction-3]
        FIN-SI
SINON
        instruction-4
        [instruction-5]
FIN-SI
```

Exemple :

Déterminer le signe du produit de 2 nombres entiers X et Y.

Quatre cas sont possibles :

- ✓ X est positif et Y est positif alors le Produit est positif
- ✓ X est négatif et Y est négatif alors le Produit est positif
- ✓ X est positif et Y est négatif alors le Produit est négatif
- ✓ X est négatif et Y est positif alors le Produit est négatif

```
VAR
     x,y : ENTIER
DEBUT
     ECRIRE "Saisir X : "
     LIRE x
     ECRIRE "Saisir Y : "
     LIRE y

     REM si x est négatif
     SI x < 0 ALORS
          REM si y est négatif alors le Produit est positif
          SI y < 0 ALORS
               ECRIRE "Le produit positif"
          REM sinon le Produit est négatif
          SINON
               ECRIRE "Le produit est négatif"
          FIN-SI

     REM si x est positif
     SINON
          REM si y est négatif alors le Produit est négatif
          SI y < 0 ALORS
               ECRIRE "Le produit est négatif"
          REM sinon le Produit est positif
          SINON
               ECRIRE "Le produit est positif"
          FIN-SI
     FIN-SI
FIN
```

5.2.3 - Exercices sur le SI

1) Contrôlez la saisie d'un code postal ; la longueur de la saisie est-elle correcte ? Trop court, trop long, « good » ? La fonction **longueur(chaîne)** qui permet de calculer la longueur d'une chaîne de caractères est pré-définie. Pour le contrôle de la qualité de la saisie, sont-ce des chiffres ? cf plus loin le traitement des chaînes de caractères.

2) Reprenez l'algorithme de la saisie de l'âge et gérez la non-saisie.

3) Ajoutez ensuite le fait que l'on accepte que les « Tintins » (entre 7 et 77 ans) que l'on divise en « Tintins mineurs » et « Tintins majeurs ». On affichera un message lorsque les candidats sont trop vieux et un autre quand ils sont trop jeunes.

BN et algorithmes (et éventuellement représentez-les sous forme d'algorigrammes).

4) Contrôle d'authentification. Le nom du user et le mot de passe sont obligatoires et doivent correspondre à des valeurs stockées dans des constantes.

5) Déterminer le signe du produit de 2 nombres entiers X et Y sans utiliser des SI imbriqués.

5.3 - La répétition (Le POUR, le TANTQUE, le JUSQU'A, etc)

Objectif :

Lorsqu'il est nécessaire d'exécuter un certain nombre de fois la même instruction ou le même groupe d'instructions on va écrire une boucle ou une itération.

5.3.1 - Le POUR

5.3.1.1 - Définition

Le POUR permet d'itérer. La boucle est gérée par une variable de contrôle de type ENTIER.
D'écriture simple il est limité.

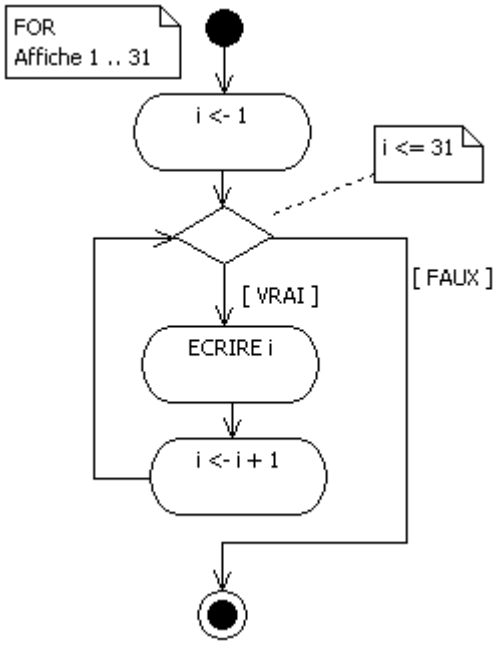
5.3.1.2 - Syntaxe

```
POUR compteur VARIANT DE début A fin [PAR PAS DE pas] FAIRE
        instruction(s)
FIN-POUR
```

début correspond à l'initialisation du compteur,
fin à la valeur à comparer pour la condition
et pas à l'incrémentation du compteur (facultatif si le pas est de 1)
début, fin et pas peuvent être des constantes ou des variables.

5.3.1.3 - Algorigrammes

Exemple : afficher 1 à 31



5.3.1.4 - Exemples

```
REM --- Afficher de 1 à 31

VAR
     compteur : ENTIER

DEBUT
     POUR compteur VARIANT DE 1 A 31 FAIRE
          ECRIRE compteur
     FIN-POUR
FIN
```

```
REM --- Somme des 5 premiers entiers positifs

VAR
     compteur, somme : ENTIER

DEBUT
     somme <- 0

     POUR compteur VARIANT DE 1 A 5 FAIRE
          somme <- somme + compteur
     FIN-POUR

     ECRIRE "Somme des 5 premiers Entiers positifs : "
     ECRIRE somme
FIN
```

5.3.1.5 - Exercices POUR

1 – Les 12 mois de l'année (en numérique : 1,2,3, ...).

2 – Les années 1900 à 2050 (1900-1901-1902- …-2048-2049-2050).

Voir aussi les exercices sur les tableaux !

Passer aux tableaux ! Et voir le TantQue plus tard (par exemple pour la recherche optimisée).

5.3.2 - Le TANTQUE

5.3.2.1 - Syntaxe

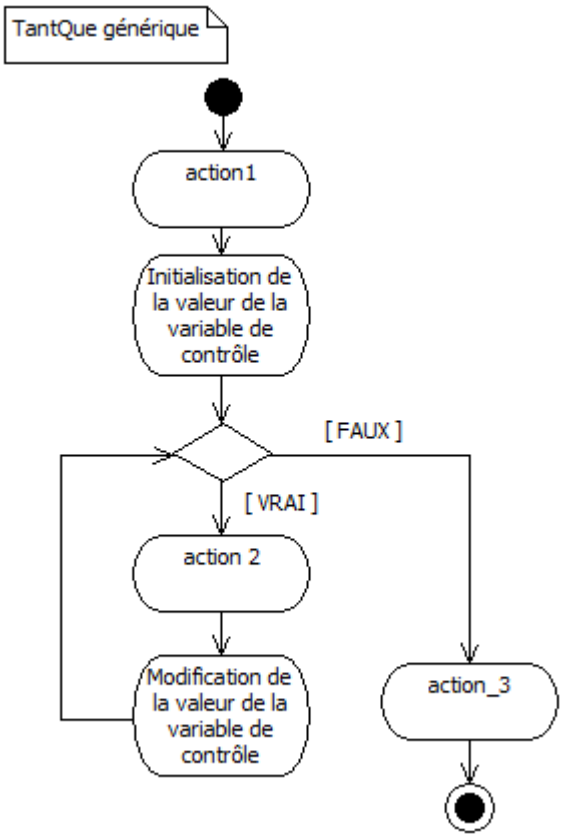
```
TANTQUE condition FAIRE
  instruction-1
  [instruction-2]
FIN-TANTQUE
```

L'utilisation de la boucle TANTQUE nécessite une attention particulière. Pour que l'instruction soit exécutée au moins une fois il faut que la condition soit VRAI au moins une fois.
De plus, c'est le plus important, il faut sortir de la boucle et donc éviter une boucle infinie fatale. Il faut donc une instruction, à l'intérieur de la boucle, qui fasse basculer la variable de contrôle à une valeur qui rende la condition FAUSSE, que ce soit par incrémentation, par modification, …

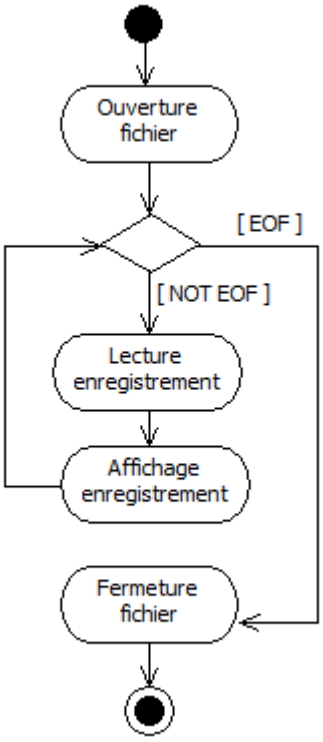
Le TANTQUE à la différence du POUR permet d'avoir une variable de contrôle qui soit d'un autre type qu'un entier. De plus elle permet de gérer aisément des conditions complexes (multiples).

5.3.2.2 - Algorigrammes

Générique



Exemple


Ouverture fichier
[EOF]
[NOT EOF]
Lecture enregistrement
Affichage enregistrement
Fermeture fichier

5.3.2.3 - Premier exemple : boucle d'affichage

```
VAR
     i : ENTIER

DEBUT
     i <- 1
     TANTQUE i <= 31 FAIRE
          ECRIRE i
          i <- i + 1
     FIN-TANTQUE
FIN
```

5.3.2.4 - Deuxième exemple : calcul de la somme de n entiers saisis au clavier

On demande à l'utilisateur de saisir au préalable le nombre d'entiers qu'il va saisir.

```
VAR
     n, valeur, somme : ENTIER

DEBUT
     somme <- 0

     ECRIRE "Combien de nombres à saisir ?"
     LIRE n

     TANTQUE n > 0 FAIRE
          ECRIRE "Saisir un nombre : "
          LIRE valeur
          somme <- somme + valeur
          n <- n - 1
     FIN-TANTQUE

     ECRIRE "La somme des valeurs : "
     ECRIRE somme
FIN
```

Étapes	n	valeur	somme
Au début	null	null	0
Après la première saisie (nombre de valeurs à saisir, 3 par exemple)	3	null	0
Après la première valeur saisie (3 par exemple)	2	3	3
Après la deuxième valeur saisie (5 par exemple)	1	5	8
Après la troisième valeur saisie (7 par exemple)	0	7	15
Après la sortie de la boucle	0	7	15

5.3.2.5 - Exercices TANK

Les mêmes que pour le POUR.

6 - TEST DES ALGORITHMES

6.1 - PRINCIPES

Tester un algorithme est aussi important que d'analyser le problème et que de le concevoir et l'implémenter.

On peut considérer qu'il faut :

- ✔ 1/3 du temps pour l'analyse et la conception,
- ✔ 1/3 du temps pour l'encodage,
- ✔ 1/3 du temps pour la conception des tests et les tests.

Il est possible de considérer les tests d'un algorithme de 2 points de vue :

- ✔ l'algorithme remplit-il la fonction définie ?
- ✔ l'algorithme se déroule-t-il comme prévu ?

Dans le premier cas il faut le tester avec plusieurs valeurs de test.

Dans le deuxième cas, il s'agit plutôt d'une procédure de débogage, il faut suivre pas à pas les valeurs de toutes les variables comme avec un débogueur !

6.2 - EXEMPLE

Calcul de la somme des N premiers entiers positifs avec un TANTQUE.

```
FONCTION somme(n : ENTIER) : ENTIER
VAR
     compteur, somme : ENTIER

DEBUT
     somme <- 0

     compteur <- 1
     TANTQUE compteur <= n FAIRE
           somme <- somme + compteur
           compteur <- compteur + 1
     FIN-TANTQUE

     RETOURNER somme
FIN-FONCTION



DEBUT
     ECRIRE "Somme : "
     ECRIRE somme(5)
FIN
```

Passage	n	Compteur	Somme	Condition	Complexité
Avant la boucle	5	1	0	VRAI	2
1	5	2	1	VRAI	3
2	5	3	3	VRAI	3
3	5	4	6	VRAI	3
4	5	5	10	VRAI	3
5	5	6	15	FAUX	3
Après la boucle	5		15	FAUX	17

Complexité : chaque instruction d'affectation ou de test coûte une unité de traitement.

7 - LES TABLEAUX

7.1 - GÉNÉRALITÉS

Un tableau est une variable à laquelle on associe plusieurs valeurs.

Les tableaux peuvent être classés en plusieurs catégories :

- ✔ les tableaux ordinaux,
- ✔ les tableaux à clés.

Les tableaux ordinaux peuvent être à 1D, 2D, 3D, …
Un tableau 1D est un vecteur, un tableau 2D est une matrice, …

Un tableau à clés est aussi appelé table de hachage, un tableau associatif, …

Il est possible de gérer des tableaux de tableaux.
Un tableau 2D est d'ailleurs un tableau de tableaux.
Mais il est aussi possible de gérer un tableau ordinal de tableaux associatifs.

Par convention arbitraire le premier indice du tableau est 0. La plupart des langages l'implémentent ainsi. Certains langages commence à 1.
La fonction prédéfinie **compte**(tableau) renvoie le nombre d'éléments d'un tableau ordinal 1D.
Un élément d'un tableau ordinal est noté nomDuTableau[indice].

L'affectation d'une valeur dans un tableau à clés est réalisée ainsi : tableau["clé"] <- valeur
L'inverse pour la récupération d'une valeur en fonction d'une clé : valeur <- tableau["clé"]

7.2 - TABLEAUX ORDINAUX

7.2.1 - Les tableaux ordinaux 1D

Un tableau ordinal contient une série de valeurs de même type (scalaire ou pas).
Un élément d'un tableau est repérable via un indice de type Entier. Le premier indice est 0.
Un tableau peut être parcouru par une boucle.

Représentation d'un tableau 1D (Les jours de la semaine)

Lundi	Mardi	Mercredi	Jeudi	Vendredi	Samedi	Dimanche

Représentation d'un tableau 1D avec ses indices.

0	1	2	3	4	5	6
Lundi	Mardi	Mercredi	Jeudi	Vendredi	Samedi	Dimanche

7.2.2 - Exemples avec des tableaux ordinaux 1D

7.2.2.1 - Initialisation d'un tableau

Exemple de tableau :

3	5	9	-1	1

Initialisation statique

```
VAR
     tableau : TAB[5] ENTIER = [3,5,9,-1,1]
DEBUT

FIN
```

Initialisation « dynamique »

Lors de la déclaration

Après l'initialisation

3	5	9	-1	1

```
VAR
     tableau : TAB[5] ENTIER

DEBUT
     tableau[0] <- 3
     tableau[1] <- 5
     tableau[2] <- 9
     tableau[3] <- -1
     tableau[4] <- 1
FIN
```

Initialisation d'un tableau de 10 éléments avec des 0.

Avec un POUR

```
VAR
     i : ENTIER
     t : TAB[10] ENTIER
DEBUT
     POUR i VARIANT DE 0 A 9 FAIRE
          t[i] <- 0
     FIN-POUR
FIN
```

Avec un TANK !

```
VAR
     i : ENTIER
     t : TAB[10] ENTIER
DEBUT
     i <- 0
     TANTQUE i < 10 FAIRE
          t[i] <- 0
          i <- i + 1
     FIN-TANTQUE
FIN
```

7.2.2.2 - Affichage des éléments d'un tableau

Avec un POUR

```
VAR
     tableau : TAB[5] ENTIER = [3,5,9,-1,1]
     indiceMax : ENTIER
     compteur : ENTIER

DEBUT
     indiceMax <- compte(tableau) - 1

     POUR compteur VARIANT DE 0 A indiceMax FAIRE
          ECRIRE tableau[compteur]
     FIN-POUR
FIN
```

Avec un TANTQUE

```
VAR
     tableau : TAB[5] ENTIER = [3,5,9,-1,1]
     nElements : ENTIER
     compteur : ENTIER

DEBUT
     nElements <- compte(tableau)
     compteur <- 0
     TANTQUE compteur < nElements FAIRE
          ECRIRE tableau[compteur]
          compteur <- compteur + 1
     FIN-TANTQUE
FIN
```

7.2.2.3 - Somme des valeurs des éléments d'un tableau

Exemple de tableau :

3	5	9	-1	1

tableau : TAB[] ENTIER → BN → somme : ENTIER

```
VAR
     tableau : TAB[5] ENTIER = [3,5,9,-1,1]
     indiceMax, somme : ENTIER
     compteur : ENTIER

DEBUT
     indiceMax <- compte(tableau) - 1
     somme <- 0

     POUR compteur VARIANT DE 0 A indiceMax FAIRE
          somme <- somme + tableau[compteur]
     FIN-POUR

     ECRIRE "Somme : " & somme
FIN
```

Tableau : [3,5,9,-1,1]

Passage	Compteur	indiceMax	tableau[compteur]	Somme	Condition
Avant la boucle	null	4	null	0	VRAI
1	0	4	3	3	VRAI
2	1	4	5	8	VRAI
3	2	4	9	17	VRAI
4	3	4	-1	16	VRAI
5	4	4	1	17	VRAI
Après la boucle	5	4		17	FAUX

7.2.3 - Exercices sur les tableaux ordinaux 1D

7.2.3.1 - La moyenne des valeurs des éléments d'un tableau

7.2.3.2 - La valeur MIN des valeurs des éléments d'un tableau

7.2.3.3 - La valeur MAX des valeurs des éléments d'un tableau

7.2.3.4 - Recherche de la position d'une valeur dans un tableau à valeurs uniques

Avant de faire les exercices qui suivent : somme des valeurs des éléments d'un tableau avec un TANTQUE.

7.2.3.5 - Recherche optimisée de la position d'une valeur dans un tableau à valeurs uniques

7.2.3.6 - Recherche de la ou des positions d'une valeur dans un tableau à valeurs non uniques

7.2.3.7 - Recherche des positions de plusieurs valeurs dans un tableau à valeurs uniques

7.2.4 - Les tableaux ordinaux 2D

Exemple : températures min et max à Paris au cours d'une semaine.

Lundi	Mardi	Mercredi	Jeudi	Vendredi	Samedi	Dimanche
10	13	10	11	14	12	15
20	20	21	20	24	25	25

	0	1	2	3	4	5	6
0	10	13	10	11	14	12	15
1	20	20	21	20	24	25	25

Initialisation

```
VAR
     tableau : TAB[2,7] ENTIER
DEBUT
     tableau[0,0] <- 10
     tableau[0,1] <- 13
     tableau[0,2] <- 10
     ...
     tableau[1,0] <- 20
     tableau[1,1] <- 20
     tableau[1,2] <- 21
     ...
FIN
```

Boucle

```
VAR
     tableau : TAB[2,7] ENTIER
DEBUT
     POUR i VARIANT DE 0 A 1 FAIRE
          POUR j VARIANT DE 0 A 6 FAIRE
               tableau[i, j] <- 0
          FIN-POUR
     FIN-POUR
FIN
```

7.3 - Les tableaux à clés

Un tableau à clés (table de hachage, tableau associatif, ...) est un tableau à « 2 colonnes », une colonne pour les « index » (les clés), une colonne pour les valeurs.

Représentation :

Clé	Valeur
Paris	75
Lyon	69
Marseille	13
Toulouse	31
Bordeaux	33

Une valeur est « récupérable » par la clé.

8 - LES FONCTIONS ET LES PROCÉDURES

8.1 - RÉSUMÉ

Les fonctions et les procédures sont des blocs de codes caractérisés par un nom.
Elles sont réutilisables.
Une procédure ne renvoie pas de résultat.
Une fonction renvoie un résultat.
Une procédure ou une fonction peut posséder un ou plusieurs ou aucun paramètres.
Les variables déclarées à l'intérieur d'une procédure ou d'une fonction sont locales à la fonction et ne sont utilisables que dans ce bloc.

8.2 - INTRODUCTION

Principes de la programmation structurée et procédurale :

La programmation structurée et procédurale n'est pas une méthode. C'est un paradigme (*).
Ou plutôt un état d'esprit.

La programmation structurée – dans les années '70 - s'oppose à l'usage du GOTO (principalement dans le langage COBOL) et s'appuie principalement sur le principe suivant : une entrée – une sortie. La volonté est de sortir de la « programmation spaghetti ».

La programmation procédurale – dans les années '70 et '80 s'oppose à la réalisation de programmes mono-bloc composés de « nombreuses » lignes. Un code doit être décomposé en blocs, chaque bloc doit être composé d'un certain nombre de lignes.
Le nombre maximum de lignes d'un bloc de code a évolué au cours du temps. D'abord le format « de lecture » faisait loi. 60 lignes à l'époque des listings, 20 lignes à l'époque où les écrans permettaient d'afficher ce nombre de lignes, environ 10 lignes actuellement – en 2015, cf les « warnings » de certaines IDE - . Le principe semble être lié à l'empan du lecteur et du support qu'il utilise. L'objectif final étant une approximation asymptotique au regard de la simplicité, tant de l'écriture que de la lecture. Le formatage « tweet » imposera-t-il sa norme ?

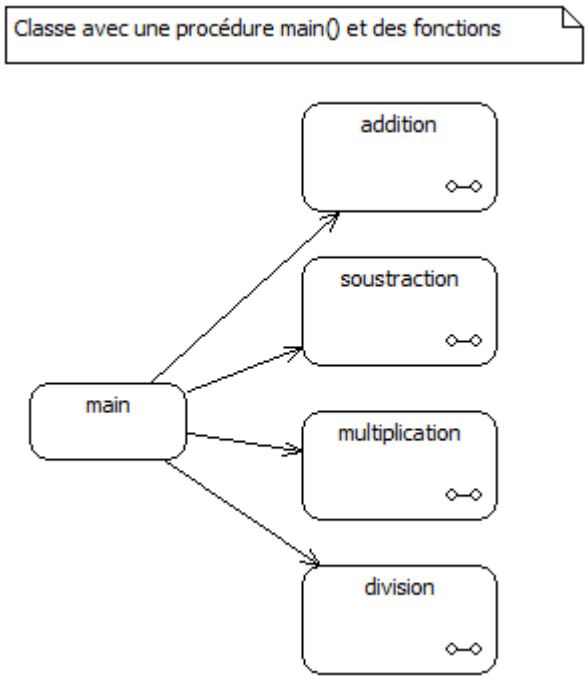
L'objectif de ces « types » de programmation est :

- ✔ de produire du code réutilisable,
- ✔ de contrôler l'écriture des programmes,
- ✔ d'intégrer facilement les parties des programmes écrites par plusieurs programmeurs,
- ✔ de documenter facilement les programmes,
- ✔ de mieux vérifier les programmes, de mieux les tester,
- ✔ de relire les programmes sans difficultés, et donc de faciliter la maintenance,
- ✔ de créer des bibliothèques.

Pour cela il faut une analyse et une programmation de haut en bas dite descendante (partir du résultat, le décomposer, …) , ce que nous avons vu précédemment.
Le programme constitue un ensemble qui pourra être divisé en blocs, eux-mêmes seront encore subdivisés pour arriver à des blocs de taille raisonnable contenant une suite d'instructions.
Chaque bloc doit correspondre à un type de tâches à effectuer au cours du programme, et donc réaliser une fonctionnalité particulière.

Note :
(*) en épistémologie (**) un paradigme est une conception théorique dominante ayant cours à une certaine époque dans une communauté scientifique donnée, qui fonde les types d'explication envisageables, et les types de faits à découvrir dans une science donnée.
En technologie (!) un paradigme est le choix des problèmes à étudier et des techniques à utiliser.

(**) épistémologie : étude de la connaissance scientifique en général ; certains disent que c'est la science de la science. Selon le TDLi – Trésor de la Langue Française Informatisé - cette définition est vieillie ! Synonymes : méthodologie, philosophie, théorie.
Selon le Larousse c'est la discipline qui prend la connaissance scientifique pour objet.



8.3 - Fonction et procédure

Il arrive fréquemment qu'une suite d'instructions doive être répétée plusieurs fois dans un même programme ou dans plusieurs. Pour éviter lors de l'écriture du programme de réécrire plusieurs fois cette suite d'instructions (CTRL/C suivi de CTRL/V !!!), il est possible de programmer et d'écrire des **sous-programmes**. On donnera un nom à la suite d'instructions et on fera référence à ce nom chaque fois que cette suite d'instructions doit être exécutée.

8.3.1 - Fonction

Si une suite d'instructions, caractérisée par un **nom**, utilisant **éventuellement des données en entrée**, fournit **une** donnée en sortie et peut être utilisée **dans des expressions** (affectation ou tests) il s'agit d'une **FONCTION**.

Note : les langages fonctionnels F#, Lisp, ..., implémentant le paradigme fonctionnel n'utilisent que ce type de code.

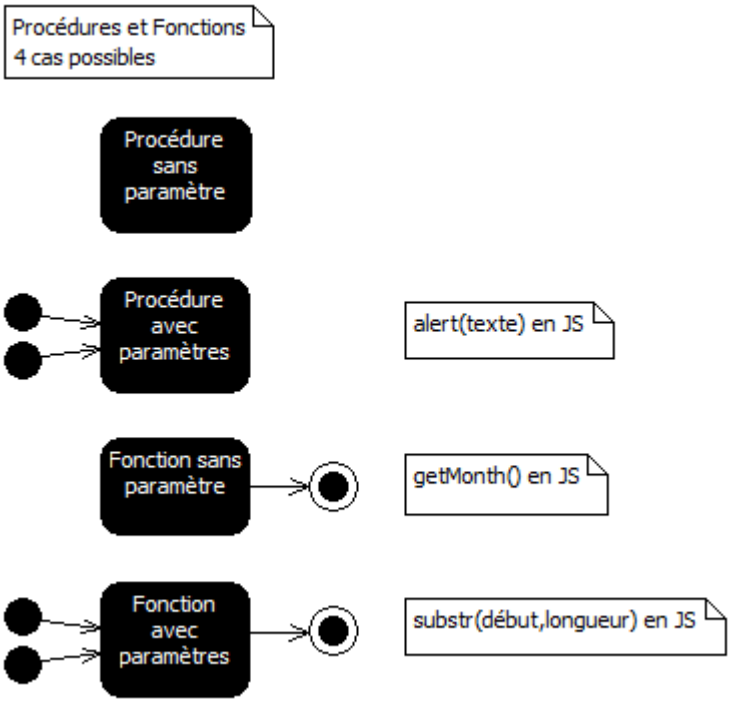
Rappel : en mathématique une fonction est définie ainsi : une fonction d'un ensemble A vers un ensemble B est une loi qui associe à chaque élément de A un unique élément de B.

8.3.2 - Procédure

Si une suite d'instructions, caractérisée par un **nom**, utilisant **éventuellement des données en entrée**, ne renvoie pas de résultat, il s'agit d'une **PROCEDURE**.

Notes :
Une fonction s'apparente à un échange, à un « jeu » de question-réponse (donne-moi la longueur du mot « fonction » ou de la phrase « Etude des fonctions »).
Une procédure s'apparente à un ordre à exécuter (affiche le contenu de l'enregistrement).

8.3.3 - Classification : Procédures/Fonctions sans ou avec des paramètres



	Procédure	Procédure paramétrée	Fonction	Fonction paramétrée
JAVA	initComponents() de Swing	main(String[] args)	toUpperCase()	substring(début, fin)
PHP	phpinfo()	define("NOM_DE_CONSTANTE", valeur) include("fichier")	getDate()	strToUpper("chaîne")
SQL	DESCRIBE		NOW()	UPPER(st)
PL/SQL			NOW()	UPPER(st)
JavaScript	window.close()	alert("message")	toUpperCase()	confirm("message")

8.4 - LOCALISATION - PORTÉE

Définition

Si un objet (une constante, une variable, une procédure, une fonction ou un type) n'a de signification qu'à l'intérieur d'une partie du programme, cet objet est local.

Les objets qui n'ont de signification que locale sont déclarés au début du corps de la fonction ou de la procédure.

Exemple de déclaration de procédure utilisant des variables globales (X,Y) avec déclaration d'une variable locale (P).

```
VAR
   x, y : ENTIER

PROCEDURE permutation()
VAR
   P : ENTIER

DEBUT
     P <- x
     x <- y
     y <- P
     ECRIRE x
     ECRIRE y
FIN-PROCEDURE

PROCEDURE principale()
DEBUT
   x <- 3
   y <- 5
   ECRIRE x
   ECRIRE y
   FAIRE permutation()
FIN-PROCEDURE
```

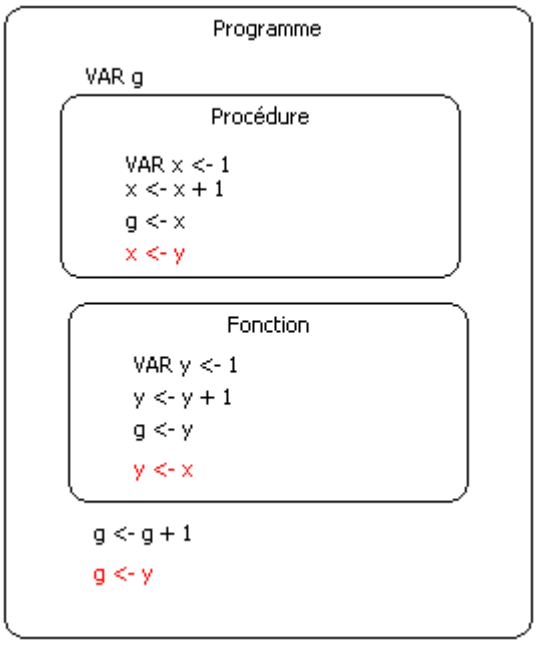
Dans le corps de la procédure (ou de la fonction) un objet local (ici p) est utilisé ainsi que 2 objets non locaux. Ces objets sont dits globaux; ce sont des VARIABLES GLOBALES qui sont déclarées dans le programme principal. Elles peuvent être utilisées dans la procédure et dans son environnement. Il est déconseillé, autant que faire se peut, d'utiliser des variables globales (cf les paramètres).

Le domaine de validité des objets locaux est le code de la procédure et ceci implique :

- ✓ Qu'avant l'appel de la procédure ces objets n'occupent pas de place mémoire,
- ✓ Que pendant l'appel de la procédure une place mémoire correspondant à leur type leur est réservée,
- ✓ Que cet espace mémoire est libéré à la fin de l'exécution des instructions de la procédure. Cet espace mémoire étant libéré il peut être alloué à de nouveaux objets. Si cette procédure est à nouveau appelée les valeurs des variables locales sont, comme lors du premier appel, non définies.

L'avantage de la localisation est que l'on n'a pas à se préoccuper de l'identification des variables locales car il n'y a pas de conflits avec des objets globaux de même nom étant donnée l'utilisation locale qui est faite de ces espaces mémoire.

En résumé l'espace d'utilisation et la durée de vie de la variable locale sont restreints à l'espace utilisé par la fonction et à la durée de son exécution.



8.5 - PARAMÈTRES

8.5.1 - Présentation

Si la suite d'instructions est appelée plusieurs fois dans le programme avec des opérandes différents il est préférable d'utiliser des paramètres qui vont rendre la fonction ou la procédure **autonome**. Les paramètres sont placés dans l'en-tête de la fonction et les identificateurs de ces paramètres sont locaux à la fonction.
De plus il est toujours préférable de ne pas travailler avec des variables globales et donc de travailler avec des paramètres. C'est le principe d'**autonomie de la boîte noire**.

Ils sont appelés **PARAMETRES FORMELS**. Les objets qui sont substitués aux paramètres formels sont appelés **paramètres effectifs** ou **PARAMETRES REELS** ou encore **ARGUMENTS**. Ces paramètres sont fournis par le programme appelant lors de l'appel de la procédure.

Le type du paramètre réel est déterminé par le type du paramètre formel comme indiqué dans l'en-tête de la fonction ou de la procédure.

Il doit y avoir **bijection** entre la liste des paramètres réels et la liste des paramètres formels. C'est-à-dire que l'ordre de l'appel doit être identique à celui de la déclaration dans l'en-tête et que le nombre d'arguments doit être identique au nombre de paramètres. Les types des arguments doivent aussi être identiques au types des paramètres.

La liste des paramètres sera écrite à la suite de l'en-tête et à la suite de l'appel. Le type de chaque paramètre doit être précisé dans l'en-tête; le tout est écrit entre parenthèses.

8.5.2 - Le type de passage des paramètres

Il existe deux sortes de substitution de paramètres.

8.5.2.1 - Par valeur

La substitution la plus commune est la **SUBSTITUTION PAR VALEUR**. On évalue le paramètre réel et on copie la valeur dans le paramètre formel correspondant.
Lors de l'appel des constantes ou des variables peuvent être utilisées comme argument.

8.5.2.2 - Par référence

Une autre substitution est la **SUBSTITUTION PAR REFERENCE ou PAR ADRESSE**. Le paramètre réel est la variable. On substitue la variable identifiée à son correspondant formel.
Il s'agit donc d'une substitution de variable ou substitution par référence ou passage d'adresse.
Cette substitution sert si un paramètre représente un "résultat" de la fonction.
Lors de l'appel **seules des variables** peuvent être utilisées comme argument.

8.5.2.3 - Choix du type de passage

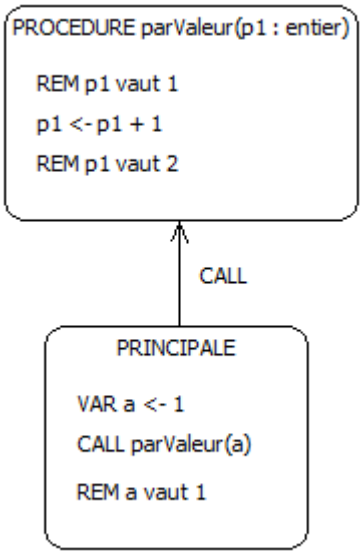
Le choix du type de substitution se fera selon les critères suivants :

1 Si un paramètre est un argument mais non pas un résultat de la fonction c'est une substitution par valeur qui est appropriée.
2 Si un paramètre joue le rôle de résultat de la fonction la substitution par référence est alors appropriée.

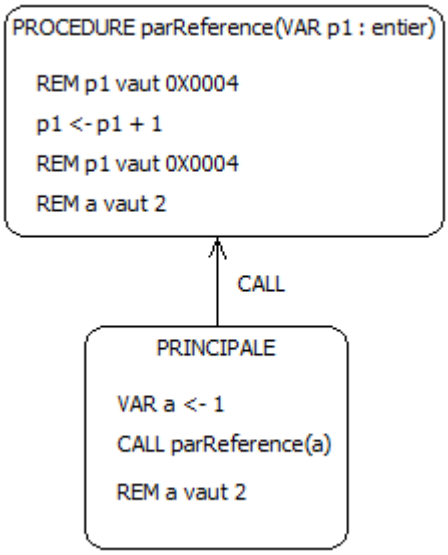
Jusqu'à présent ce qui a été vu était le premier type de substitution.

La règle d'écriture pour le deuxième type est la suivante : dans l'en-tête de la fonction on ajoutera **VAR** devant le paramètre formel qui sert de paramètre à la substitution.

Passage par valeur



Passage par référence



8.6 - LES FONCTIONS : SYNTAXE

Le code d'une fonction est formée de **deux parties** :

l'en-tête (4 parties) : l'en-tête contient le mot réservé FONCTION qui correspond à la nature du bloc suivi de l'identificateur de la fonction puis de parenthèses et éventuellement d'une liste de paramètres formels typés. Enfin le type de la fonction est précisé.

le corps : le corps contient les éventuelles déclarations de variables locales précédées du mot clé VAR, l'instruction ou les instructions à exécuter encadrées des mots réservés DEBUT et FIN-FONCTION. Parmi ces instructions, la dernière en principe, l'instruction RETOURNER permet de renvoyer le résultat au code appelant.

Il existe des variables qui ne sont utilisées que dans une suite d'énoncés. Ces variables n'ont de signification qu'à l'intérieur de ce bloc. La clarté des énoncés sera d'autant plus grande si ces variables n'apparaissent que là où elles doivent être utilisées. La partie déclarative du programme principale sera d'autant moins longue et d'autant plus facile à lire et à examiner.

Ces variables ont un domaine d'activité et de validité limité ainsi qu'une durée de vie limitée; elles sont locales à la fonction, ce sont des **VARIABLES LOCALES**.

Il arrive aussi qu'une suite d'instructions ou d'énoncés apparaisse plusieurs fois dans un programme sous une forme structurellement identique, mais différente au niveau des valeurs. Dans ce dernier cas, les diverses apparitions de la fonction peuvent être homogénéisées par la substitution systématique d'identificateurs ou d'expressions. Il s'agit du cas où la suite d'instructions est transformée en un schéma de fonction abstrait. Les entités qu'il faut alors préciser pour chaque appel sont les **paramètres formels** de la fonction (cf. le paragraphe suivant sur les paramètres).

Syntaxe

```
FONCTION nomDelaFonction([parametre1 : TYPE[, parametre2 : TYPE]]) : TYPE

VAR
  identificateur-1 : TYPE
  identificateur-2 : TYPE
  identificateur-n : TYPE

DEBUT
  instruction-1
  instruction-2

  instruction-n-1
  * --- la dernière instruction est l'affectation d'une valeur ou du résultat d'un calcul
  RETOURNER expression
FIN-FONCTION
```

Syntaxe d'appel d'une fonction

C'est une expression.

```
variable <- nomDeLaFonction([liste d'arguments])
```

```
SI nomDeLaFonction([liste d'arguments]) = …
```

Exemple 1 : la fonction addition(a,b)

```
REM ----------- Addition

FONCTION addition(a,b : ENTIER) : ENTIER
VAR
   resultat : ENTIER

DEBUT
   resultat <- a + b
   RETOURNER resultat
FIN-FONCTION



REM ----------- Procédure principale
VAR
   x,y : ENTIER

DEBUT
   ECRIRE "Saisir X : "
   LIRE x
   ECRIRE "Saisir Y : "
   LIRE y
   ECRIRE "Résultat : "
   ECRIRE addition(x,y)
FIN
```

Exemple 2 : appel d'une fonction dans un test

```
* ----------------- Exemple 2 : calcul du salaire net
FONCTION net(x : REEL) : REEL
VAR
   taux_1, taux_2, seuil_1, seuil_2, resultat : REEL
DEBUT
   taux_1   <- 0.7
   taux_2   <- 0.8
   seuil_1  <- 5000.0
   seuil_2  <- 9000.0

   SI (x > seuil_1) ET (x < seuil_2)
      ALORS resultat <- x * taux_1
   SINON SI x >= seuil_2
      ALORS resultat <- x * taux_2
      SINON resultat <- x
   RETOURNER resultat
FIN-FONCTION



REM ----------- Procédure principale
VAR
   revenuBrut   : REEL
   plafond      : REEL
   revenuFinal  : REEL

DEBUT
   plafond <- 9000.0
   ECRIRE "Saisir le Revenu Brut : "
   LIRE revenuBrut
   SI net(revenuBrut) > plafond
      ALORS revenuFinal <- net(revenuBrut) * 0.9
      SINON revenuFinal <- net(revenuBrut)
   ECRIRE revenuFinal
FIN
```

8.7 - Les procédures : syntaxe

Comme cela a été dit une procédure est une fonction qui ne renvoie pas de résultat.
L'en-tête – la signature - est composé de 3 parties : le mot clé PROCEDURE, le nom de la procédure, des parenthèses et éventuellement des paramètres typés.
Le corps de la procédure commence par DEBUT et se termine par FIN-PROCEDURE, suit éventuellement la déclaration des variables locales précédée du mot clé VAR, et ensuite la liste des instructions à exécuter.
Donc elle n'est pas typée et ne comprend aucune instruction RETOURNER.

Une procédure est appelée avec l'instruction FAIRE et ne peut pas être utilisée dans une expression.

Syntaxes :

Ecriture de la procédure

```
PROCEDURE nomDeLaProcedure([parametre1 : TYPE[, parametre2 : TYPE]])

VAR
  identificateur-1 : TYPE
  identificateur-2 : TYPE
  identificateur-n : TYPE

DEBUT
   instruction-1
   instruction-2

   instruction-n
FIN-PROCEDURE
```

Syntaxe d'appel d'une procédure

A chaque fois qu'il est nécessaire d'exécuter cette série d'instructions dans le programme, il suffira de la remplacer par l'appel de la procédure de la façon suivante :

Instruction ***FAIRE*** suivie du nom de la procédure :

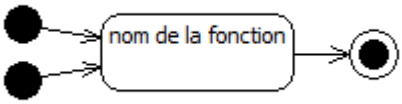
```
FAIRE nomDeLaProcedure([liste d'arguments])
```

Exemple

```
* --- Programme Passage de paramètres par référence
PROCEDURE permutation(VAR a,b : ENTIER);
VAR
     p: ENTIER

DEBUT
     p <- a
     a <- b
     b <- p
FIN-PROCEDURE



*----- Procédure principale
VAR
  x,y : ENTIER

DEBUT
     ECRIRE 'Saisir X : '
     LIRE x
     ECRIRE 'Saisir Y : '
     LIRE y
     permutation(x,y)
     ECRIRE 'Résultat de la permutation : '
     ECRIRE "X: "
     ECRIRE x
     ECRIRE "Y: "
     ECRIRE y
FIN
```

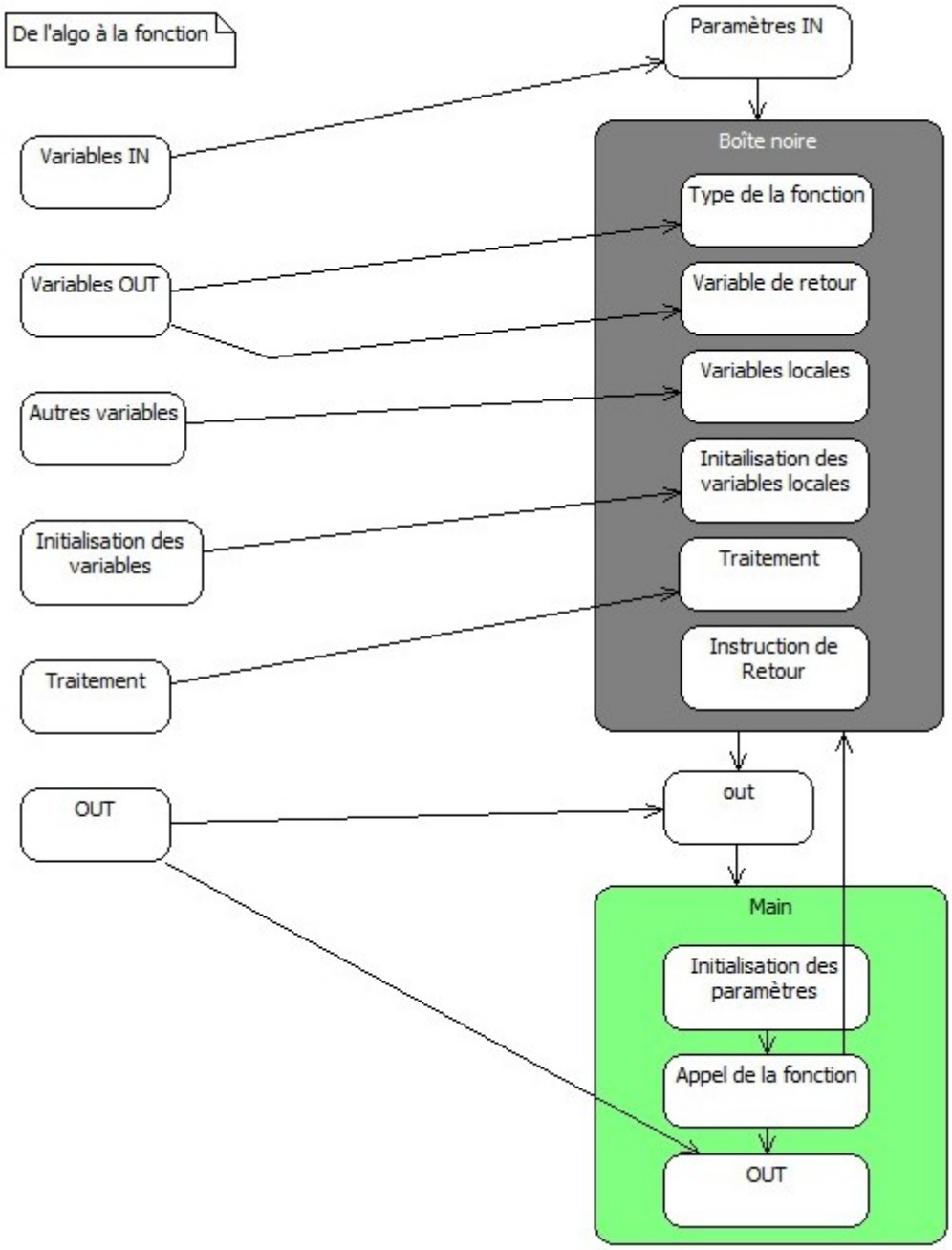
8.8 - Démarche pour l'écriture d'une fonction

Le schéma de la boîte noire.



Etapes	Exemple
Objectif ?	Convertir des secondes en minutes:secondes. Par exemple de 75 à 1:15
Nom ?	secondes2minutes
Paramètres et leur type ?	secondes : numérique
Type de la fonction ?	chaîne
Corps de la fonction ?	Diviser, récupérer le quotient, récupérer le reste, compléter éventuellement, concaténer.
Variables locales et leur type ?	Une variable pour le diviseur, le dividende, le quotient, le reste et le résultat, … des entiers et une chaîne de caractères

Note : pour une procédure c'est la même démarche sans le type et le retour.


De l'algo à la fonction
Paramètres IN
Variables IN
Boîte noire
Type de la fonction
Variables OUT
Variable de retour
Variables locales
Autres variables
Initailisation des variables locales
Initialisation des variables
Traitement
Instruction de Retour
Traitement
OUT
out
Main
Initialisation des paramètres
Appel de la fonction
OUT

8.9 - EXERCICES SUR LES FONCTIONS

8.9.1 - Première série

Transformer un énoncé en une BN (Boîte Noire).
Écrire la signature de la fonction.

L'énoncé doit être lu à l'envers pour formaliser la boîte noire.

Exemples :

Je veux la somme des valeurs des éléments d'un tableau.
Tableau : REEL[] → BN : somme → somme : REEL

Je veux les n premiers caractères d'une chaîne de caractères.
Chaîne : CHAINE, n : ENTIER → BN : gauche → caractères : CHAINE

Énoncés :

Je veux afficher en diagonale une chaîne de caractères.

Je veux afficher à l'envers une chaîne de caractères.

Je veux le nombre d'éléments d'un tableau.

Je veux la moyenne des valeurs des éléments d'un tableau.

Je veux une date au format US à partir d'une date au format FR.

Je veux la position d'une valeur numérique dans un tableau.

Je veux enlever les espaces en début et en fin de chaîne (trim()).

Je veux savoir si un mot est un palindrome.

Je veux savoir si une saisie est un numérique.

Je veux savoir si une saisie est un code postal français.

Je veux connaître la fréquence de chaque lettre de l'alphabet dans un texte.

8.9.2 - Deuxième série

Pour tous les exercices qui suivent :

- ✔ la boîte noire,
- ✔ la signature,
- ✔ la fonction.

8.9.2.1 - La fonction max des éléments d'un tableau

8.9.2.2 - La fonction min des éléments d'un tableau

8.9.2.3 - La fonction recherche d'un élément dans un tableau

9 - LES STRINGS

9.1 - Définition et Principales opérations sur les chaînes de caractères

9.1.1 - Définition

Une chaîne de caractères est un ensemble de caractères. Il faut les considérer comme des tableaux de caractères. Le premier caractère est d'indice 0.
Les chaînes de caractères sont placées entre guillemets doubles droits (").

9.1.2 - Fonctions pré-définies

Fonction	Description
longueur(chaîne)	Calcule la longueur d'une chaîne de caractères : ENTIER
extract(chaîne, position)	Extrait un caractère : CAR
extract(chaîne, début, longueur)	Extrait une sous-chaîne : CHAINE
concat(chaine1, chaine2)	Concatène 2 chaînes de caractères : CHAINE
&	Concatène 2 chaînes de caractères : CHAINE
majuscules(chaîne)	Met en majuscules une chaîne de caractères
minuscules(chaîne)	Met en minuscules une chaîne de caractères

9.1.3 - Exemples

9.1.3.1 - Affichage d'une chaîne de caractères caractère par caractère à la verticale

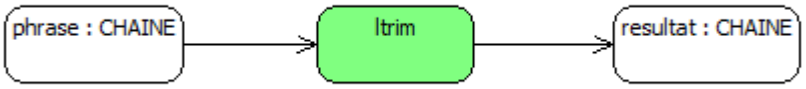
RC : Retour chariot et saut de ligne.

```
REM Affichage à la verticale

FONCTION verticale(texte : CHAINE) : CHAINE

VAR
     i, nCaracteres : ENTIER
     resultante : CHAINE

DEBUT
     nCaracteres <- longueur(texte)
     i <- 0
     TANTQUE i < nCaracteres FAIRE
          resultante <- resultante & texte[i] & RC
          i <- i + 1
     FIN-TANTQUE

     RETOURNER texte
FIN-FONCTION

REM Procédure principale : appel de la fonction

VAR
     phrase : CHAINE
DEBUT
     ECRIRE "Saisir une phrase : "
     LIRE phrase
     ECRIRE "Résultat" & verticale(phrase)
FIN
```

9.1.3.2 - Élimination des espaces en début de phrase

Admettons la phrase suivante " C'est un algorithme facile à écrire".
Il y a trois espaces en trop - saisis involontairement par l'opérateur. L'algorithme doit pouvoir les éliminer. En revanche il faut laisser les espaces entre les mots puisqu'ils sont significatifs.

Après que la phrase ait été saisie - dans une chaîne de caractères - on analysera les caractères pour savoir s'il s'agit d'un caractère espace - on utilisera une boucle et on ne fait rien d'autre que d'avancer. Ensuite avec une deuxième boucle on concaténera les caractères dans une chaîne résultante.



```
REM Elimination des espaces en début de phrase

FONCTION ltrim(texte : CHAINE) : CHAINE

VAR
     i, nLongueur : ENTIER
     resultante : CHAINE

DEBUT
     i <- 0
     nLongueur <- longueur(texte)
     resultante <- ""

     TANTQUE texte[i] = " " FAIRE
          i <- i + 1
     FIN-TANTQUE

     TANTQUE i < nLongueur FAIRE
          resultante <- resultante & texte[i]
          i <- i + 1
     FIN-TANTQUE

     RETOURNER texte
FIN-FONCTION


REM Procédure principale : appel de la fonction

VAR
     phrase : CHAINE

DEBUT
     ECRIRE "Saisir une phrase : "
     LIRE phrase
     ECRIRE "Résultat" & ltrim(phrase)
FIN
```

Test : La phrase saisie " il est"

Texte	N	I	OUT
" il est"	8	0	""
" il est"	8	1	""
" il est"	8	2	"i"
" il est"	8	3	"il"
" il est"	8	4	"il "
" il est"	8	5	"il e"
" il est"	8	6	"il es"
" il est"	8	7	"il est"
" il est"	8	8	"il est"

9.2 - Exercices sur les Strings

Exercices … que des fonctions !!!

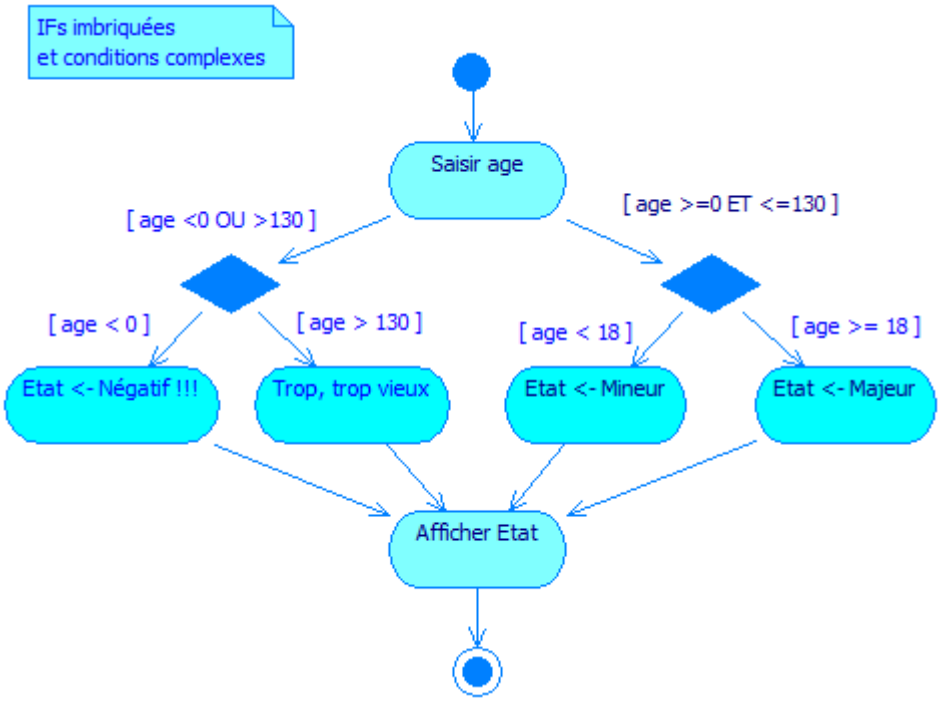
Niveau 1 : trop facile !
Niveau 2 : facile
Niveau 3 : moyen
Niveau 4 : difficile
Niveau 5 : très difficile

Algorithme à faire	Description	Niveau
Contrôle d'une saisie numérique « en chaîne »	Chiffre par chiffre (estNum)	2
Contrôle du CP	En réutilisant le précédent	2
Contrôle de l'âge	Version complète ! Et c'est une chaîne qui est en input. L'âge est compris entre 0 et 130 et on distingue les majeurs des mineurs. cf int = Integer.valueOf(String)	2
MajPremier	Met la première lettre en majuscule et les autres en minuscule (UCFirst)	1
NomPropre	Met en majuscule la première lettre de tous les mots d'un texte (UCWords ou presque)	3
Left	Récupérer n caractères à gauche	1
Right	Récupérer n caractères à droite	2
LTrim	Élimine les espaces à gauche	2
Rtrim	Élimine les espaces à droite	3
Trim	Élimine les espaces à gauche et à droite	3
RechercherCaractere	Rechercher un caractère dans toute la chaîne. Caractère est présent une seule fois ; Version optimisée.	1
RechercherCaractere	Rechercher un caractère dans toute la chaîne. Le même caractère peut être présent plusieurs fois.	3
RemplacerCaractere	Remplacer un caractère par un autre.	2
Underscore2Camel	Camélise : nom_du_client ---> nomDuClient	2 ou 3
Camel2Underscore	NomDuClient ---> nom_du_client	3
Implode	Tableau ordinal de chaînes ---> Chaîne avec séparateur	1
Explode (split)	Chaîne ---> ArrayList	3
Fréquences des lettres dans un texte	Première possibilité : avec un tableau statique des lettres non accentuées (26 * 2). Deuxième possibilité : avec un tableau dynamique	3
Compter le nombre de mots dans un texte		4
Date FR 2 date US	jj/mm/aaaa -> yyyy-mm-dd	1 avec split()
Date US 2 date FR	yyyy-mm-dd -> jj/mm/aaaa	1 avec split()

10 - ANNEXES

10.1 - IF ET ARBORESCENCE

Algorigramme



Arbre