

ALGORITHMIQUE

"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)

TABLE DES MATIERES

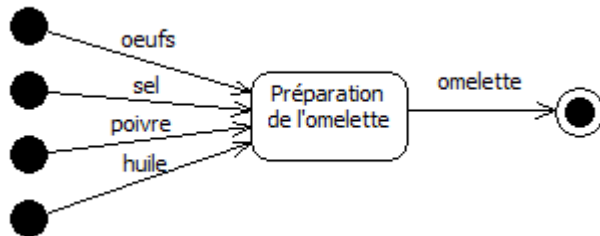
1 - Introduction.....	4
1.1 - Comment faire une omelette en cassant des œufs ?.....	5
1.2 - Objectif de ce support.....	7
1.3 - Bibliographie.....	8
1.4 - Références WEB.....	9
1.5 - ALGOBOX.....	10
1.6 - Etymologie et définitions.....	11
1.7 - Considérations générales.....	14
1.8 - Considérations sur la notion de langage.....	15
2 - L'analyse pour l'algorithme.....	16
2.1.1 - Principe de l'analyse descendante.....	17
2.1.2 - Exemple : établir une fiche de paie.....	18
2.1.3 - Autre exemple : le calcul de la variance.....	21
2.2 - Archétype d'un algorithme.....	22
2.3 - Le principe des Entrées/Sorties.....	23
3 - Les données, les types et les opérateurs.....	24
3.1 - Qu'est-ce qu'une donnée?.....	25
3.2 - Qu'est-ce qu'un type?.....	25
3.3 - Qu'est-ce qu'une constante?.....	25
3.4 - Qu'est-ce qu'une variable?.....	25
3.5 - Déclaration des variables.....	26
3.6 - Déclaration des constantes.....	27
3.7 - Les types.....	28
3.7.1 - Le type ENTIER.....	28
3.7.2 - Le type REEL.....	29
3.7.3 - Le type booléen (BOOLEEN).....	30
3.7.4 - Le type caractère (CAR).....	31
3.7.5 - Le type chaîne de caractères (CHAINE).....	32
3.7.6 - Le type tableau (TAB).....	33
3.7.7 - Le type enregistrement (ENREGISTREMENT).....	34
3.8 - Les opérateurs.....	35
3.8.1 - Les opérateurs arithmétiques.....	36
3.8.2 - Les opérateurs de comparaison.....	36
3.8.3 - Les opérateurs logiques.....	36
3.8.4 - Opérateurs divers.....	36
4 - Les actions élémentaires.....	37
4.1 - L'affectation.....	39
4.2 - La lecture.....	40
4.3 - L'écriture.....	41
4.4 - Découpage de l'algorithme.....	42
5 - Les trois principales structures des algorithmes.....	43
5.1 - Présentation.....	44
5.2 - La structure séquentielle.....	45
5.2.1 - Définition.....	45
5.2.2 - Exemple.....	47
5.2.3 - Exercice.....	47
5.3 - La structure alternative.....	48
5.3.1 - Définition.....	48
5.3.2 - L'alternative à deux branches.....	49
5.3.3 - L'alternative à n branches.....	60
5.3.4 - Exercices sur le SI.....	61

5.4 - La répétition (Le POUR, le TANTQUE, le JUSQU'A, etc).....	62
5.4.1 - Le POUR.....	63
5.4.2 - Le TANTQUE.....	66
5.4.3 - Le FAIRE ... TANTQUE.....	71
5.4.4 - Le FAIRE ... JUSQU'A.....	73
6 - Test des algorithmes.....	75
7 - Les fonctions et les procédures.....	77
7.1 - Introduction.....	78
7.2 - Fonction et procédure.....	79
7.2.1 - Fonction.....	79
7.2.2 - Procédure.....	79
7.2.3 - Classification : Procédures/Fonctions sans ou avec des paramètres.....	80
7.3 - Localisation - Portée.....	81
7.4 - Paramètres.....	83
7.4.1 - Présentation.....	83
7.4.2 - Le type de passage des paramètres.....	84
7.5 - Les fonctions : syntaxe.....	86
7.6 - Les procédures : syntaxe.....	90
7.7 - Démarche pour l'écriture d'une fonction.....	92
8 - Exemples d'algorithmes.....	93
8.1 - Algorithmes sur les numériques.....	94
8.1.1 - Multiplication sans le signe de la multiplication.....	94
8.1.2 - Elévation à la puissance P d'un entier N.....	97
8.1.3 - Exercices.....	99
8.1.4 - Calculatrice.....	100
8.2 - Algorithmes sur les tableaux.....	101
8.2.1 - Trouver la valeur MAX dans un tableau.....	101
8.2.2 - Rechercher une valeur dans un tableau.....	103
8.2.3 - Exercices.....	106
8.3 - Algorithme sur un tableau à 2 dimensions.....	107
8.3.1 - Somme des valeurs des éléments d'un tableau 2D.....	107
8.3.2 - Exercice.....	108
8.4 - Algorithmes sur les chaînes de caractères.....	109
8.4.1 - Calcul de la longueur d'une chaîne.....	109
8.4.2 - Elimination des espaces en début de phrase.....	111
8.4.3 - Exercices.....	114
9 - ANNEXES.....	115
9.1 - Quelques mots-clés utilisés.....	116
9.2 - Algorithmes et ordigrammes.....	117
9.2.1 - La structure séquentielle.....	118
9.2.2 - La structure conditionnelle (SI).....	119
9.2.3 - La structure conditionnelle (SI ... SINON).....	120
9.2.4 - La structure conditionnelle (AU CAS OU).....	121
9.2.5 - La structure itérative (POUR).....	122
9.2.6 - La structure itérative (TANTQUE).....	123
9.2.7 - La structure itérative (FAIRE TANT QUE).....	125
9.2.8 - La structure itérative (FAIRE JUSQU'A).....	126
9.2.9 - Appel d'une fonction ou d'une procédure.....	127
9.2.10 - Exercices.....	128

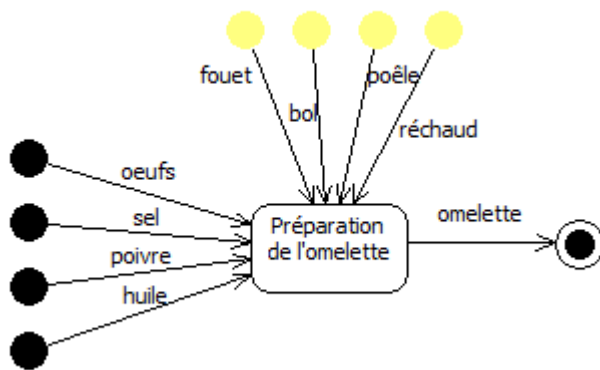
1 - INTRODUCTION

1.1 - COMMENT FAIRE UNE OMELETTE EN CASSANT DES ŒUFS ?

Il faut des œufs, du sel, du poivre et de l'huile d'olive (des ingrédients).



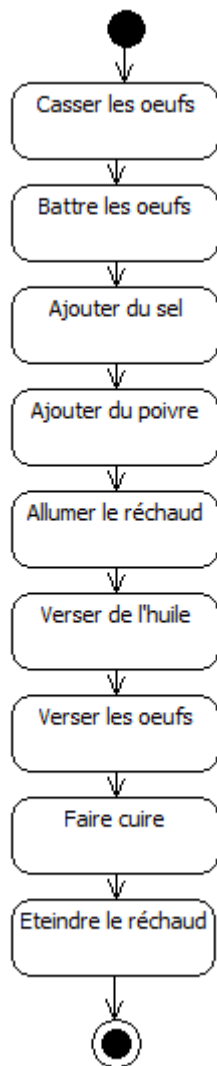
Mais il faut aussi un bol ou une assiette creuse, une fourchette ou un fouet, une poêle, un réchaud (des ustensiles).



Les ustensiles informatiques sont les variables et les instructions.

et aussi .../...

une recette (un mode opératoire, un ordonnancement).



En somme pour obtenir un OUTPUT il faut des INPUTS, des OUTILS et un MODE OPERATOIRE.

Dans un algorithme c'est la même chose : il y a des INPUTS, des OUTPUTS et un mode opératoire.

1.2 - OBJECTIF DE CE SUPPORT

Introduire à la programmation procédurale [impérative].

Le paradigme de la programmation procédurale est le suivant : on s'intéresse aux actions.

Les principaux chapitres sont les suivants :

- ✓ introduction,
- ✓ analyse descendante,
- ✓ données, types et opérateurs,
- ✓ actions élémentaires,
- ✓ structures algorithmiques,
- ✓ test des algorithmes.

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

L'introduction à la POO est abordée dans un autre document.

1.3 - 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.

"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.

1.4 - 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>

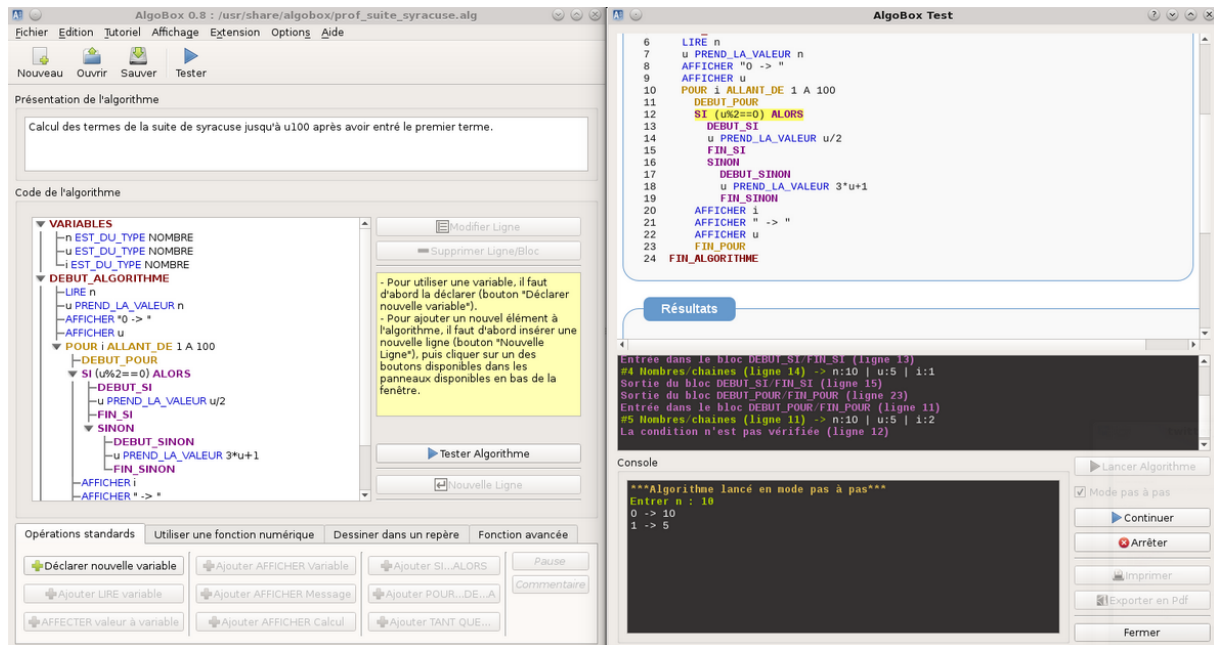
Le choix ...

Autres : cf votre ami Gogol !

1.5 - ALGOBOX

A découvrir !

<http://www.xmlmath.net/algobox/>

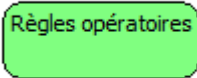


1.6 - ETYMOLOGIE ET DÉFINITIONS

Le mot algorithme vient du nom du mathématicien persan Al Khuwarizmi (latinisé au Moyen Âge en Algoritmi).

Première définition (source?) :

Ensemble de règles opératoires propres à un calcul.



Règles opératoires

Commentaires :

$\text{calcul} = \{\text{règles}\}$

La définition est succincte ; seul le processus est évoqué.

La notion de calcul est centrale.

Calculer la TVA à partir d'un prix HT est un calcul.

Stocker des données saisies via un formulaire dans une base de données, est-ce un calcul ?

Deuxième définition (Dictionnaire de l'Académie française – 9ème édition) :

Méthode de calcul qui indique la démarche à suivre pour résoudre une série de problèmes équivalents en appliquant dans un ordre précis une suite finie de règles.

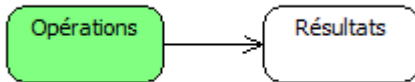
Par curiosité lisez la définition du Larousse, qui est quasiment identique, mais qui n'évoque pas le principe de la généricité.

Commentaires :

La définition évoque les notions de généricité, d'ordre, de finitude.

Troisième définition (AFNOR) :

Un algorithme est l'ensemble des règles opératoires et des procédés, définis en vue d'obtenir un résultat déterminé, au moyen d'un nombre fini d'opérations.

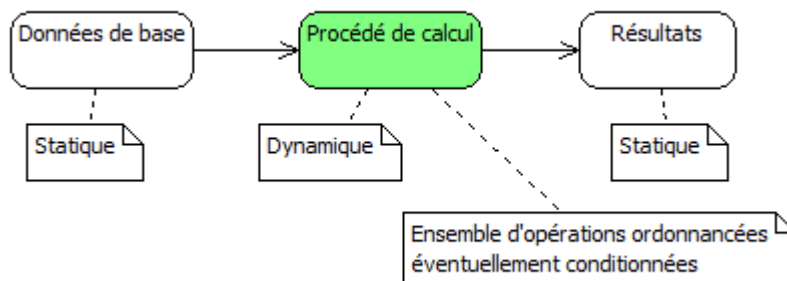


Commentaires :

La définition évoque explicitement la notion de résultat.
Elle contient une certaine redondance.

Quatrième définition (source ?) :

Un algorithme est un procédé de calcul; il décrit une succession d'opérations à exécuter dans un certain ordre et sous certaines conditions pour passer des données de base aux résultats. Il correspond à un processus, découpé en étapes dont l'enchaînement permet d'arriver à un but fixé.



Commentaires :

La définition évoque les données sources, les données cibles et le procédé pour passer des unes aux autres.

De plus elle évoque un ordonnancement et une exécution sous condition des opérations composant le procédé.

En conclusion provisoire :

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.

Au final un algorithme est équivalent à $R = f(D)$. Avec R pour résultats, D pour données et $f()$ pour fonction.

Remarques :

L'introduction des données dans la définition de l'algorithme modifie le point de vue.

Si l'on reprend la première définition où un algorithme est un ensemble d'opérations propre à un calcul l'accent est mis sur les opérations, sur les fonctionnalités de l'algorithme. Cela correspond bien au paradigme procédural.

L'introduction des données dans la définition ne s'apparente pas au paradigme objet où l'analyse est centrée sur les données. Cette information supplémentaire permet seulement d'aider à l'analyse descendante.

Voir aussi les fragments du "Discours de la méthode" de Descartes cités par Denis Lapoire dans son cours d'algorithmique (Disponible à <http://lapoire.developpez.com/algorithmique/initiation/>).

1.7 - 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 ordinogrammes (cf les annexes).

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, une lecture plus facile et donc et une maintenance aisée.

- 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 algo c'est ?

L'unité de calcul pour le temps est l'affectation et le test.

Cf un exemple au paragraphe 8.2.2 sur la recherche dans un tableau.

L'écriture d'un algorithme nécessite du papier, un crayon, une gomme, de la rigueur, de l'intuition, ...

1.8 - CONSIDÉRATIONS SUR LA NOTION DE LANGAGE

Un langage est caractérisé par :

- ✓ un lexique,
- ✓ une syntaxe,
- ✓ une sémantique.

En français :

les mots eau, vache et boit existent. Le mot vaca ne fait pas partie du lexique français.

La phrase "La vache boit de l'eau" est syntaxiquement correcte. La phrase "L'eau boit la vache" est syntaxiquement correcte. La phrase "Vache eau boit" est syntaxiquement incorrecte.

La phrase "La vache boit de l'eau" est sémantiquement correcte. La phrase "L'eau boit la vache" est sémantiquement incorrecte.

Il en est de même en informatique. Il en sera de même en UML, en Merise, ...

Un algorithme doit être lexicalement, syntaxiquement et sémantiquement correct.

2 - L'ANALYSE POUR L'ALGORITHMIQUE

2.1.1 - Principe de l'analyse descendante

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.

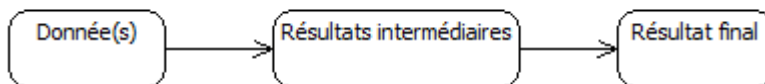
Ainsi on décompose le problème initial en problèmes plus simples.

Définir saura déterminer si le résultat est **une donnée** ou bien le résultat d'un calcul.

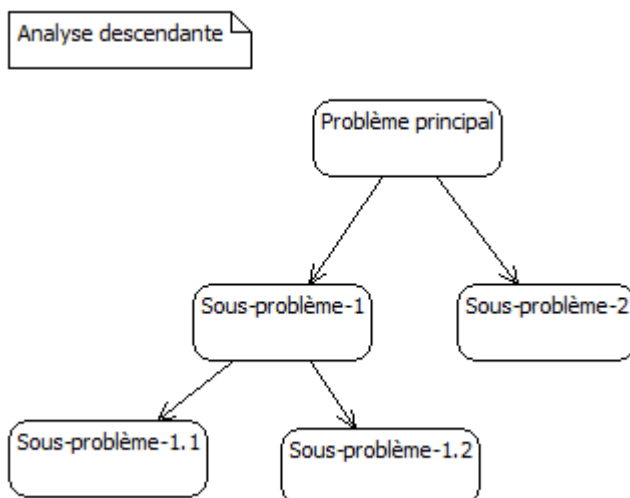
Définir chaque résultat intermédiaire, en introduisant éventuellement de nouveaux intermédiaires.

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

L'algorithme est l'ensemble ordonné des définitions obtenues.



Analyse descendante



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 :

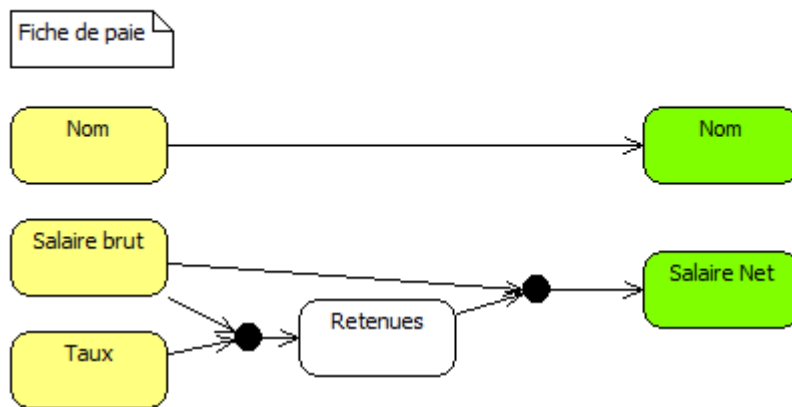
- ✓ nom
- ✓ salaire brut
- ✓ taux

Les données en sortie sont :

- ✓ nom
- ✓ salaire net

Les identificateurs et leur type :

- ✓ nom : CHAÎNE de caractères
- ✓ salaireBrut : REEL
- ✓ taux : REEL
- ✓ retenues : REEL
- ✓ salaireNet : REEL



Le nom est non décomposable

Les retenues sont obtenues par calcul en fonction du salaire brut et du taux de retenues

Le salaire net est obtenu en fonction du salaire brut et des retenues

L'algorithme :

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

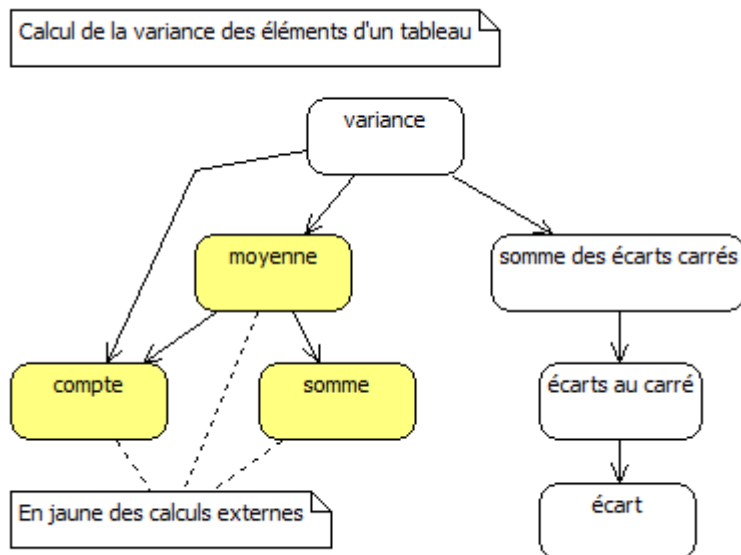
DEBUT
    ECRIRE "Saisir nom : "
    LIRE nom
    ECRIRE "Saisir salaire brut : "
    LIRE salaireBrut
    ECRIRE "Saisir taux : "
    LIRE taux

    retenues <- salaireBrut * taux
    salaireNet <- salaireBrut - retenues

    ECRIRE "Nom : " & nom
    ECRIRE "Salaire net : "
    ECRIRE salaireNet
FIN
```

2.1.3 - Autre exemple : le calcul de la variance

La variance est un indice de dispersion d'un ensemble de valeurs.
C'est la moyenne des carrés des écarts par rapport à la moyenne.



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.

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

```
VAR
    REM Déclaration des variables

DEBUT
    REM Instructions

FIN
```

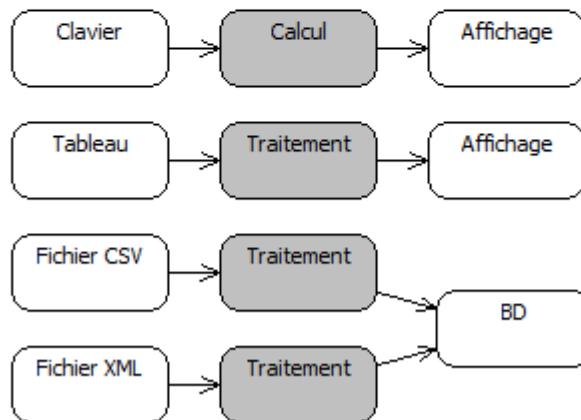
2.3 - LE PRINCIPE DES ENTRÉES/SORTIES

Ainsi que nous l'avons vu dans la définition d'un algorithme cette première ternaire est au centre de la démarche algorithmique : entrées, calculs ou traitements, sorties.

Générique



Spécifiques



3 - LES DONNÉES, LES TYPES ET LES OPÉRATEURS

3.1 - QU'EST-CE QU'UNE DONNÉE?

Fait ou notion représenté sous une forme conventionnelle convenant à un traitement par des moyens automatiques (ISO). C'est une information simple ou complexe. Par exemple le nom d'une personne ou l'ensemble des informations – nom, prénom, date de naissance, etc - concernant une personne.

3.2 - QU'EST-CE QU'UN TYPE?

Les programmes manipulent des informations de différentes natures : des nombres, des textes, des dates, ...

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

Lorsque l'ensemble des valeurs d'un type est un ensemble ordonné de valeurs non décomposables en plusieurs constituants, il s'agit d'un type élémentaire ou scalaire.

Les types scalaires sont : ENTIER (pour les numériques entiers), REEL (pour les numériques à partie décimale), CAR (pour les caractères), BOOLEEN.

Les autres types sont des types non scalaires ou composés : TAB pour les tableau, CHAINE pour les chaînes de caractères, ENREGISTREMENT pour les structures composites, OBJET (cf la POO), ...

3.3 - QU'EST-CE QU'UNE CONSTANTE?

Une constante est soit une valeur "brute" soit une zone mémoire qu'un programme va utiliser pour stocker temporairement une valeur qui ne changera pas durant l'exécution du programme.

Une constante est une valeur prise dans l'ensemble des valeurs du type. 28 est une constante de type ENTIER, 3,14 est une constante de type REEL, "A" est une constante de type CAR, "AZERTY" est une constante de type CHAINE, VRAI est une constante de type BOOLEEN. On peut distinguer les constantes littérales, comme précédemment, des constantes déclarées (cf plus bas la déclaration des constantes).

3.4 - QU'EST-CE QU'UNE VARIABLE?

Une variable est une zone mémoire qu'un programme va utiliser pour stocker temporairement une valeur.

Une variable est un triplet (identificateur, type, valeur). L'identificateur c'est son nom qui permet de la désigner tout au long de l'algorithme. Le type est fixe. En revanche la valeur variera au cours de l'exécution de l'algorithme.

3.5 - DÉCLARATION DES VARIABLES

Déclarer les variables c'est établir la liste des variables que l'algorithme va manipuler.
Pour chaque variable on précisera le type.

Il existe 2 conventions pour le nommage des variables : l'"underscorisation" ou la "camélisation".

Dans tous les cas les noms des variables doivent être sémantiquement significatifs.

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

La syntaxe est la suivante :

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

Si deux variables sont de même type on pourra séparer les identificateurs par une virgule.

```
    identificateur1, identificateur2 : TYPE-1
    identificateur3 : TYPE-2
```

Exemples :

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

3.6 - DÉCLARATION DES 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
```

3.7 - LES TYPES

3.7.1 - Le type ENTIER

Définition :

Le type ENTIER est l'ensemble des entiers. -155, 0, 65535 sont des entiers.

Les opérations sur les variables et constantes du type ENTIER sont les suivantes : addition, soustraction, multiplication, division entière, division réelle.

Exemple de déclaration :

```
| VAR  
|   code : ENTIER
```

3.7.2 - Le type REEL

Définition :

Le type REEL est l'ensemble des nombres réels. 3,14 et -0,90 sont des réels.

Les opérations sur les variables et les constantes du type REEL sont les suivantes : addition, soustraction, multiplication, division réelle.

Exemple de déclaration :

```
| VAR  
|     salaireNet : REEL
```

3.7.3 - Le type booléen (BOOLEEN)

Définition :

Le type Booléen est l'ensemble des valeurs logiques c'est-à-dire les deux constantes VRAI et FAUX.

Exemple de déclaration :

```
| VAR  
    test : BOOLEEN
```

3.7.4 - Le type caractère (CAR)

Définition :

Le type caractère (noté CAR) est un ensemble de caractères établi par une norme (ASCII par exemple). On trouve toutes les lettres de l'alphabet (minuscules et majuscules), les dix chiffres décimaux (0 à 9) et d'autres caractères (ponctuation, le caractère espace noté " ", ...).

Les constantes caractères seront notées entre guillemets. "A", "a", " ", "3", "*".

La principale opération sur les caractères est la concaténation pour former une chaîne de caractères.

Il est possible d'ajouter la fonction prédéfinie qui permet de récupérer le code d'un caractère et la fonction inverse qui permet de produire un caractère à partir d'un code. Par convention la référence sera la table ASCII étendue.

Exemple de déclaration :

```
| VAR  
|     lettre : CAR
```

3.7.5 - Le type chaîne de caractères (CHAINE)

Définition :

Le type CHAINE de caractères est très proche du type tableau.

Il s'agit d'un tableau de caractères mais il est possible de faire une opération de concaténation de chaînes ("addition" de deux ou plusieurs chaînes de caractères).

Le type CHAINE est l'ensemble des chaînes de caractères, mots ou ensemble de caractères, que l'on peut former avec des caractères de type CAR.

Les chaînes sont en algorithmique de longueur quelconque.

Les constantes de type CHAINE sont notées entre guillemets.

Une chaîne peut être vide et elle sera notée avec deux guillemets.

"AZERTY", "00002", "A", " " sont des constantes du type chaîne de caractères.

"" est une chaîne vide.

Les opérations sur les chaînes sont la concaténation, l'extraction de caractères.

On fera référence à un caractère de la chaîne en indiquant le nom de la chaîne suivi d'un indice entre crochets.

Par convention l'indice du premier caractère est 1.

Exemple de déclaration :

```
| VAR  
|     nom : CHAINE
```


3.7.6 - Le type tableau (TAB)

Définition :

Alors qu'une variable est associée à une seule valeur variant au cours de l'algorithme, un tableau a un identificateur, un type mais est associé à un ensemble de valeurs.

Il en découle que tous les éléments du tableau sont de même type.

Chaque valeur est repérable par le nom du tableau suivi d'un ou plusieurs indices entre crochets.

Les indices sont des entiers.

Lors de la déclaration d'un tableau on indique sa dimension entre crochets et le type des éléments. Comme paramètre d'une fonction ou d'une procédure on omettra la dimension.

Par convention l'indice du premier élément est 1 pour un tableau ordinal à 1 dimension ou 1,1 pour un tableau ordinal 2D ...

Exemple de déclaration :

```
VAR  
    tableau1 : TAB[10] ENTIER
```

Déclaration d'un tableau à une dimension (assimilable à un vecteur) pouvant contenir 10 entiers.

--	--	--	--	--	--	--	--	--	--

```
VAR  
    tableau2 : TAB[3,10] CAR
```

Déclaration d'un tableau à deux dimensions (une matrice) pouvant contenir 30 caractères sur 3 lignes et 10 colonnes.

Exemples d'accès à un élément du tableau :

tableau1[5] est le cinquième élément d'un tableau 1D.

tableau2[2,3] est le 3^{ème} élément de la 2^{ème} ligne d'un tableau 2D.

3.7.7 - Le type enregistrement (ENREGISTREMENT)

Définition :

Un enregistrement est caractérisé par un identificateur et plusieurs éléments de même type ou de types différents.

Exemple de déclaration :

```
| VAR  
    fiche : ENREGISTREMENT[nom, prenom, tel : CHAINE; age : ENTIER]
```

cf le support algorithmique_fichier.doc pour des exemples.

3.8 - LES OPÉRATEURS

Il existe plusieurs types d'opérateurs :

- ✓ Les opérateurs arithmétiques,
- ✓ Les opérateurs de comparaison,
- ✓ Les opérateurs logiques,
- ✓ Les opérateurs divers.

Note : d'autres opérateurs existent (binaires, ...) mais ils ne seront pas abordés dans ce support.

3.8.1 - Les opérateurs arithmétiques

Opérateur	Opération
+	L'addition
-	La soustraction
*	La multiplication
:	La division entière
/	La division réelle

3.8.2 - Les opérateurs de comparaison

Opérateur	Opération
=	Egal à
<	Inférieur à
<=	Inférieur ou égal à
>	Supérieur à
>=	Supérieur ou égal à
<>	Différent de

3.8.3 - Les opérateurs logiques

Opérateur	Opération
ET	ET logique
OU	OU logique inclusif
NON	Négation logique
XOR	OU logique exclusif

cf le paragraphe sur l'alternative avec des conditions complexes.

3.8.4 - Opérateurs divers

Opérateur	Opération
<-	L'affectation
&	La concaténation

4 - LES ACTIONS ÉLÉMENTAIRES

Les actions élémentaires sont :

- ✓ La lecture (récupération des INPUTS),
- ✓ L'affectation (avec un calcul ou sans),
- ✓ L'appel d'une fonction pré-définie,
- ✓ L'appel d'une fonction définie,
- ✓ L'écriture (Traitement des OUTPUTS).

4.1 - 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.

Exemples :

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

4.2 - LA LECTURE

Définition :

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

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.

4.3 - L'ÉCRITURE

Définition :

L'ordre d'écriture - ECRIRE - est l'opération inverse.

La valeur de la variable sera affectée à un périphérique de sortie (Ecran, Fichier, BD, ...).

Exemples :

| ECRIRE nom

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

| ECRIRE fiche

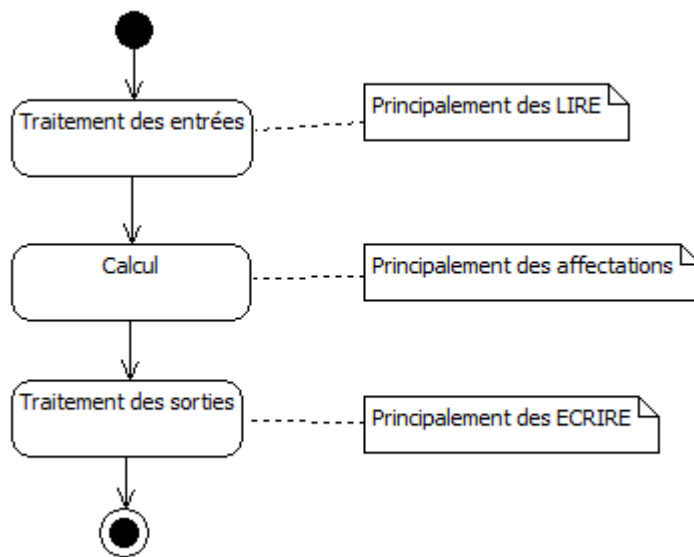
Toutes les données de la variable fiche sont transférées de la mémoire vers le disque.

4.4 - 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 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.

5 - LES TROIS PRINCIPALES STRUCTURES DES ALGORITHMES

5.1 - PRÉSENTATION

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.

Un algorithme est souvent un mélange de ces 3 structures.

Note

Séquence de la vie courante :

le matin je me lève, je prépare mon café, je m'assieds dans mon fauteuil, j'allume un Gitane, je prends un roman, je lis, ...

Conditionnelle de la vie courante :

si le livre que je lisais hier est terminé j'en prends un autre ...

Itération de la vie courante :

je tourne les pages du livre et j'en lis une autre tant que je n'ai pas terminé le livre !

5.2 - LA STRUCTURE SÉQUENTIELLE

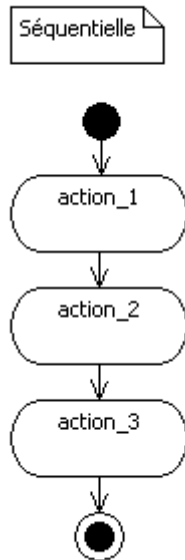
5.2.1 - Définition

Lorsque pour obtenir un résultat particulier il faut exécuter un certain nombre d'actions les unes après les autres nous sommes en présence d'une structure séquentielle.

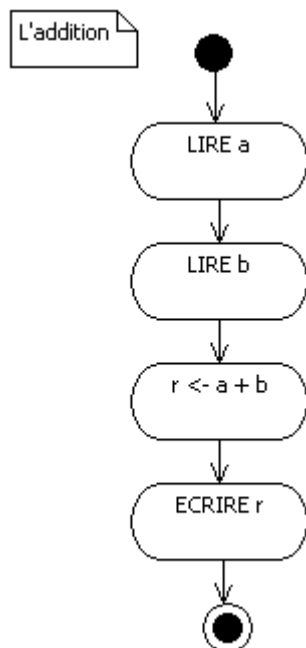
.../...

Algorithme :

Générique



Exemple



Note : l'utilisation des ordinogrammes peut – pourrait - dispenser du pseudo langage.

5.2.2 - Exemple

L'addition de 2 nombres

```
VAR
    nombre1,
    nombre2,
    resultat : ENTIER

DEBUT
    ECRIRE "Addition de deux nombres"

    ECRIRE "Saisir premier nombre : "
    LIRE nombre1

    ECRIRE "Saisir deuxième nombre : "
    LIRE nombre2

    resultat <- nombre1 + nombre2

    ECRIRE "Résultat : " & resultat
FIN
```

5.2.3 - Exercice

La permutation de 2 nombres !!! Algo et algorithme.

5.3 - LA STRUCTURE ALTERNATIVE

5.3.1 - 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.

5.3.2 - L'alternative à deux branches

5.3.2.1 - 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.

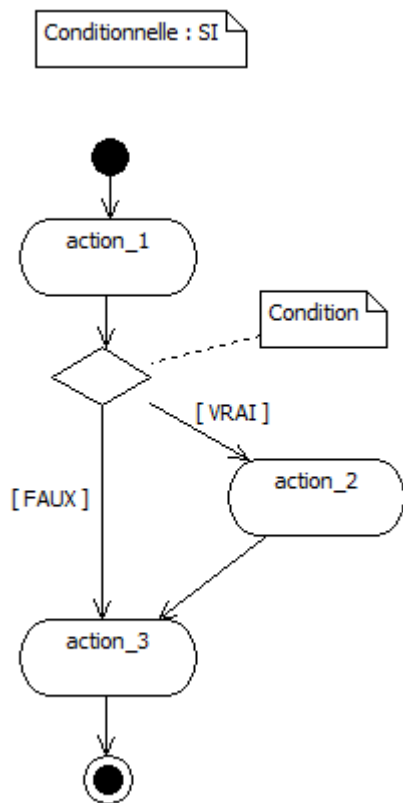
Par exemple si le total d'une facture est inférieur à 100 francs on ajoute 15 francs de frais de transport.

Exemple :

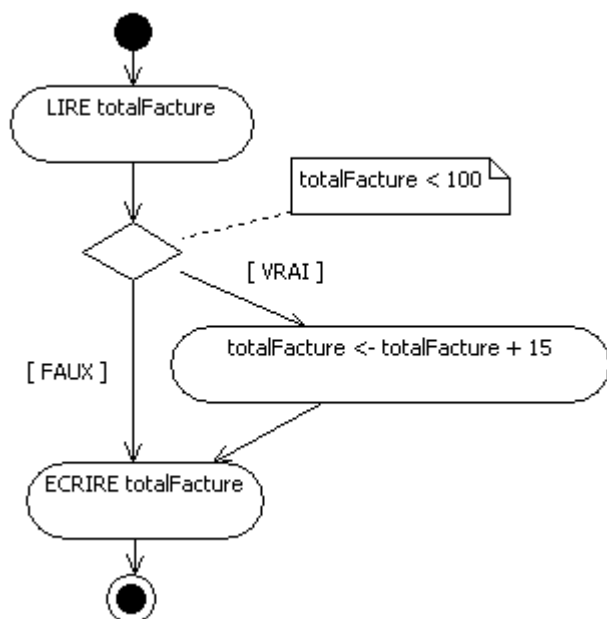
```
| SI totalFacture < 100 ALORS totalFacture <- totalFacture + 15
```

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

Générique



Exemple



5.3.2.2 - Deuxième Syntaxe : SI ... SINON ...

SI condition **ALORS**

instruction-1

instruction-2

SINON

instruction-3

instruction-4

FIN-SI

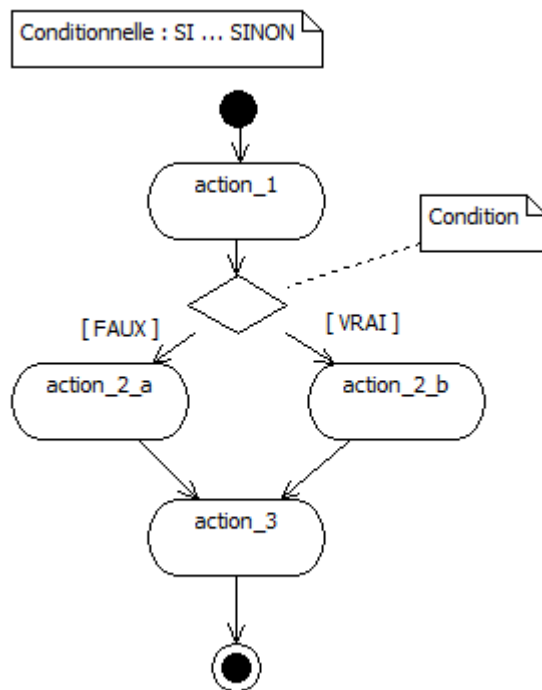
Exemple :

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

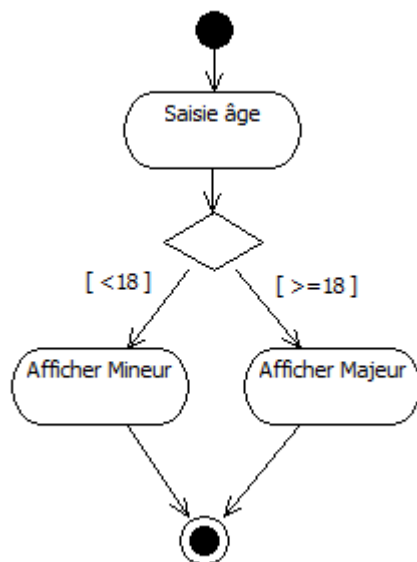
Remarque : dans le cas où il y a une seule instruction on peut omettre le FIN-SI.

.../...

Générique



Exemple



5.3.2.3 - Les conditions complexes

Définition :

La 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	OU	NON C1	OUX
Vrai	Vrai	Vrai	Vrai	Faux	Faux
Vrai	Faux	Faux	Vrai	Faux	Vrai
Faux	Vrai	Faux	Vrai	Vrai	Vrai
Faux	Faux	Faux	Faux	Vrai	Vrai

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.

Autres 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

La condition est remplie si les deux énoncés sont vrais.

Règles :

Si total facture est inférieur à 1000 : pas de remise

Si total facture est entre 1000 et 2000 : remise de 5%

Si total facture est supérieur à 2000 : remise de 10%

```
VAR
    totalFacture : REEL

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

    SI (totalFacture >= 1000) ET (totalFacture <= 2000)
        ALORS totalFacture <- totalFacture * 0.95

    SI totalFacture > 2000 ALORS totalFacture <- totalFacture *
0.90

    ECRIRE "Total de la facture après une éventuelle remise : " &
totalFacture
FIN
```

Exemple : OU logique INCLUSIF

La condition est remplie si un des énoncés est VRAI.

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

    SI op = "+" ALORS r <- x + y
    SINON SI op = "-" ALORS r <- x - y

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

5.3.2.4 - Le cas du XOR (OU EXCLUSIF)



5.3.2.5 - 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
    SINON instruction-2
SINON instruction-3
```

ou aussi

```
SINON
    SI condition-3 ALORS instruction-3
    SINON instruction-4
```

Exemple :

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

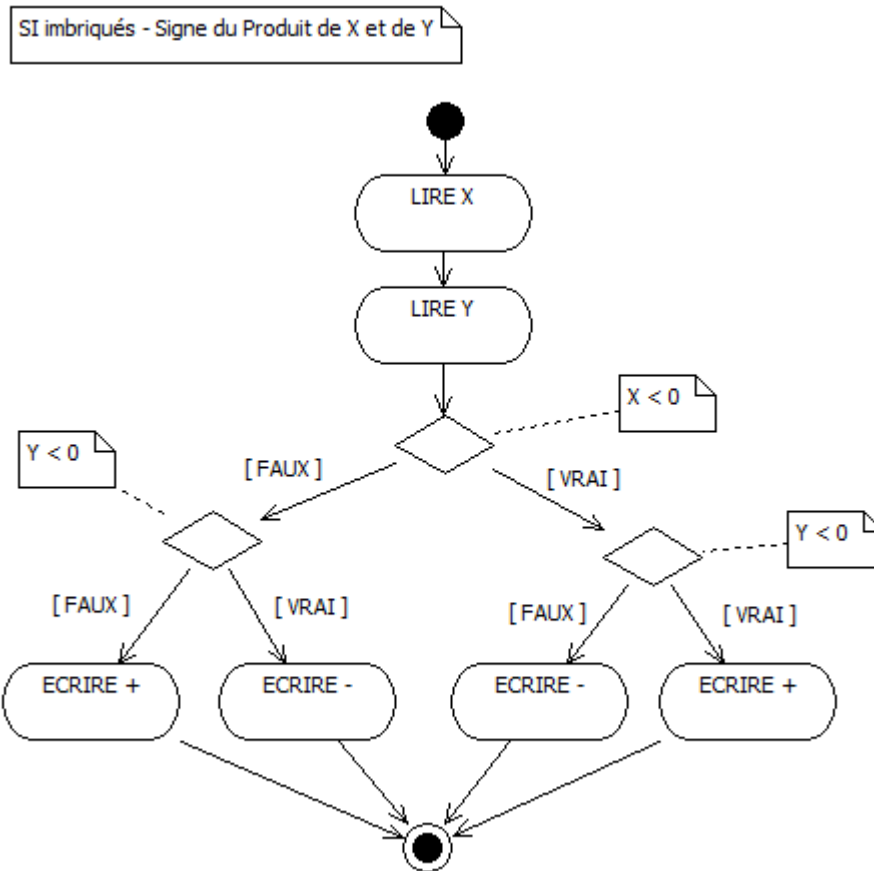
Quatre cas sont possibles :

- ✓ X est positif et Y est positif alors P est positif
- ✓ X est négatif et Y est négatif alors P est positif
- ✓ X est positif et Y est négatif alors P est négatif
- ✓ X est négatif et Y est positif alors P 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 p est positif
        SI y < 0 ALORS ECRIRE "P est positif"
        REM sinon p est négatif
        SINON ECRIRE "P est négatif"
    REM si p est positif
    SINON
        REM si y est négatif alors p est négatif
        SI y < 0 ALORS ECRIRE "P est négatif"
        REM sinon p est positif
        SINON ECRIRE "P est positif"
    FIN-SI
FIN
```

Ordinogramme ...



5.3.3 - L'alternative à n branches

Définition :

Au lieu de considérer seulement deux cas associés aux valeurs VRAI et FAUX d'une expression booléenne, il peut être nécessaire de représenter un choix entre n possibilités. C'est avec la directive "au cas où" que l'on exprime cette alternative à n branches.

Syntaxe :

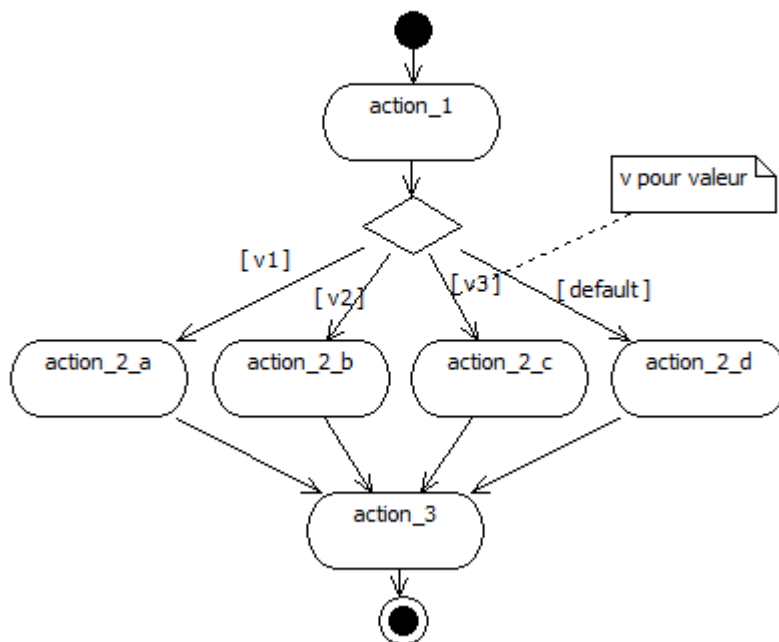
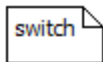
```

AU CAS OU
  condition-1 : instruction-1
  condition-2 : instruction-2
  condition-n : instruction-n
  SINON instruction m
FIN-CAS

```

A chaque condition peut correspondre une suite d'instructions; dans ce cas on encadrera cette série d'instructions de DEBUT et de FIN

De plus une série de cas peuvent correspondre aux mêmes instructions ; on utilisera alors SINON comme pour le SI.



5.3.4 - Exercices sur le SI

Ecrire les algorithmes et représentez-les sous forme d'algorigrammes.

1 - Contrôlez le fait que la saisie de l'âge soit réaliste. Affichez le message en conséquence.
Version light et version hard !

2 - Contrôle d'authentification. Le nom du user et le mot de passe sont obligatoires et doivent correspondre à des valeurs stockées.

3 – Contrôlez la saisie d'un code postal. 5 chiffres, etc (mais seulement sur la longueur – la fonction longueur(chaine) est pré-définie - , cf plus loin pour le contrôle de chaque chiffre).

5.4 - LA RÉPÉTITION (LE POUR, LE TANQUE, 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.

5.4.1 - Le POUR

Définition :

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

Syntaxe :

```
POUR compteur VARIANT DE début A fin PAR PAS DE pas FAIRE
    instructions
FIN-POUR
```

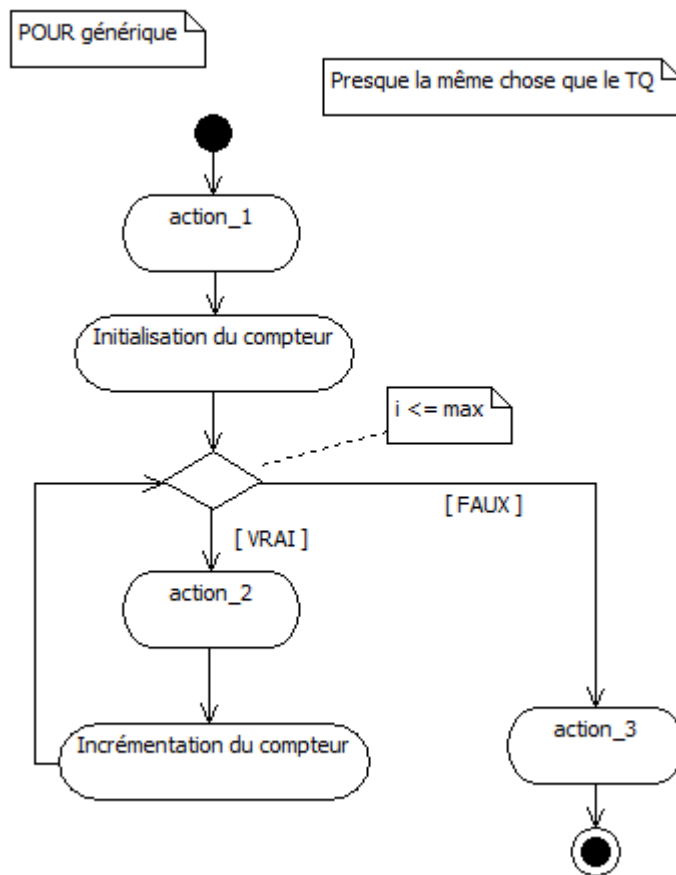
début correspond à l'initialisation du compteur,
fin à la valeur à comparer pour la condition,
et pas à l'incrémentation du compteur.
début, fin et pas peuvent être des constantes ou des variables.

Exemple :

```
REM --- Somme des 5 premiers entiers positifs
VAR
    ctr, somme : ENTIER
DEBUT
    somme <- 0
    POUR ctr VARIANT DE 1 A 5 PAR PAS DE 1 FAIRE
        somme <- somme + ctr
    FIN-POUR
    ECRIRE "Somme des 5 premiers Entiers positifs : "
    ECRIRE somme
FIN
```

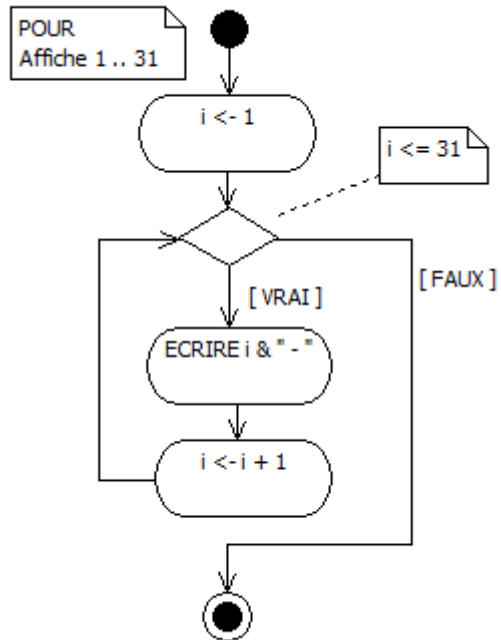
.../...

Générique



Exemples

Affiche 1 - 2 - 3 - 4 - 5 ... - 31



5.4.2 - Le TANTQUE

5.4.2.1 - 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.

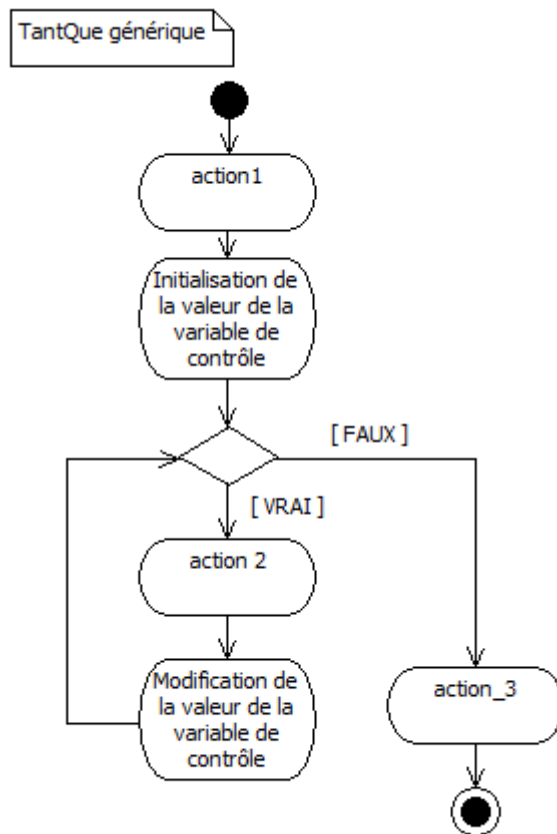
De plus, c'est le plus important, il faut sortir de la boucle et donc éviter une boucle infinie fatale. Il faut donc une instruction, à l'intérieur de la boucle, qui fasse basculer la variable de contrôle à une valeur qui rende la condition FAUSSE, que ce soit par incrémentation, par modification, ...

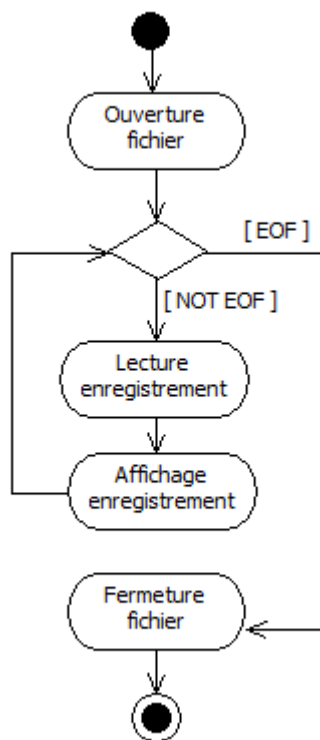
Le TANTQUE à la différence du POUR permet d'avoir une variable de contrôle qui soit d'un autre type qu'un entier. De plus elle permet de gérer aisément des conditions complexes (multiples).

.../...

5.4.2.2 - Ordinogrammes

Générique



Exemple

5.4.2.3 - Premier exemple : initialisation d'un tableau

Tous les éléments d'un tableau comprenant 10 éléments sont initialisés avec des 0.

```
VAR
    I : ENTIER
    T : TAB[10] ENTIER

DEBUT
    I <- 1
    TANTQUE I <= 10 FAIRE
        T[I] <- 0
        I <- I + 1
    FIN-TANTQUE
FIN
```

5.4.2.4 - 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

```

Etapes	N	Valeur	Somme
Au début	0	0	0
Après la première saisie (nombre de valeurs à saisir)	3	0	0
Après la première valeur saisie (3)	3	3	3
Après la deuxième valeur saisie (5)	2	5	8
Après la troisième valeur saisie (7)	1	7	15
Après la sortie de la boucle	0	7	15

5.4.3 - Le FAIRE ... TANTQUE

Syntaxe :

```
FAIRE
  instruction-1
  instruction-2
  instruction-n
TANTQUE condition
```

Cela veut dire que l'instruction sera exécutée tant que la condition est VRAI.

A la différence de la boucle TANT QUE ici l'instruction ou les instructions seront exécutées au moins une fois puisque le test fait après le premier passage.

cf l'algorithme dans les annexes.

Exemple : somme de n entiers saisis au clavier

```
VAR
    n, valeur, somme : ENTIER

DEBUT
    somme <- 0

    ECRIRE "Combien de saisies ?"
    LIRE n

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

    ECRIRE "Somme : "
    ECRIRE somme
FIN
```


5.4.4 - Le FAIRE ... JUSQU'A

Syntaxe :

```
FAIRE
  instruction-1
  instruction-2
  instruction-n
JUSQU'A condition
```

Cela veut dire que l'instruction sera exécutée jusqu'à ce que la condition soit VRAI. Donc tant que la condition est FAUSSE on reste dans la boucle.

A la différence de la boucle TANT QUE ici l'instruction ou les instructions seront exécutées au moins une fois puisque le test fait après le premier passage.

cf l'algorithme dans les annexes.

Exemple : somme de n entiers saisis au clavier

```
VAR
    n, valeur, somme : ENTIER
DEBUT
    somme <- 0

    ECRIRE "Combien de saisies ?"
    LIRE n

    FAIRE
        ECRIRE "Saisir une valeur : "
        LIRE valeur
        somme <- somme + valeur
        n <- n - 1
    JUSQU'A n = 0

    ECRIRE "Somme : "
    ECRIRE somme
FIN
```

6 - TEST DES ALGORITHMES

Pour tester un algorithme il faut suivre pas à pas les valeurs de toutes les variables comme avec un débogueur !

Exemple :

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

```

FONCTION somme (n : ENTIER) : ENTIER
VAR
    ctr, somme : ENTIER

DEBUT
    somme <- 0

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

    REOURNER somme
FIN-FONCTION

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

```

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

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

7 - LES FONCTIONS ET LES PROCÉDURES

7.1 - INTRODUCTION

Principes de la programmation structurée :

La programmation structurée n'est pas une méthode. Elle est plutôt un état d'esprit qui doit permettre :

- ✓ De contrôler l'écriture des programmes.
- ✓ D'intégrer facilement les parties des programmes écrites par plusieurs programmeurs.
- ✓ De documenter sans difficultés les programmes.
- ✓ De mieux vérifier les programmes.
- ✓ De relire les programmes sans difficultés.
- ✓ De faciliter la maintenance.
- ✓ De créer des bibliothèques.

Pour cela il faut une analyse et une programmation de haut en bas dite descendante, 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.

7.2 - 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, 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.

7.2.1 - Fonction

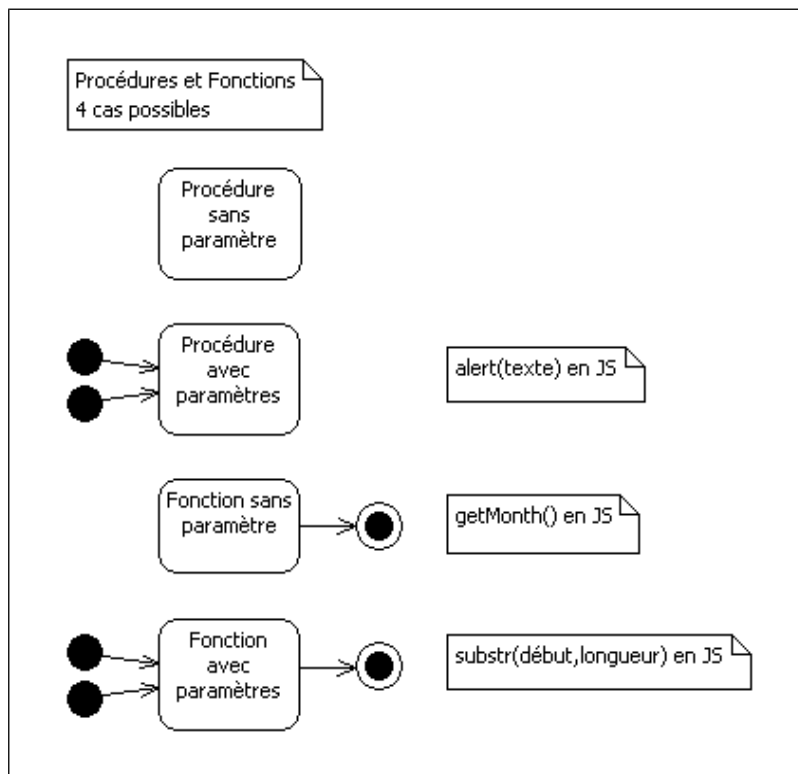
Si une suite d'instructions, caractérisée par un nom, utilisant éventuellement des données en entrée, fournit une ou des données en sortie et peut servir dans des expressions il s'agit d'une **FONCTION**.

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

7.2.2 - 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**.

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



Exemples en PHP :

```
phpinfo()
define(« NOM_DE_CONSTANTE », valeur_de_la_constant)
getDate()
strToUpper(« chaîne »)
```

Exemples en Java :

```
???
main(String[] args)
toUpperCase()
substring(début, fin)
```

Exemples en SQL :

```
Aucune procédure,
aucune procédure,
st = NOW()
st = UPPER(st).
```


7.3 - 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.

Exemple de déclaration de procédure utilisant des variables globales avec déclaration de variables locales.

```
VAR
    X, Y : ENTIER

PROCEDURE permutation()
VAR
    P : ENTIER

DEBUT
    P <- X

    X <- Y
    Y <- P
FIN-PROCEDURE

PROCEDURE principale()
DEBUT
    X <- 3
    Y <- 5
    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 n'est pas recommandé d'écrire ainsi (cf les paramètres).

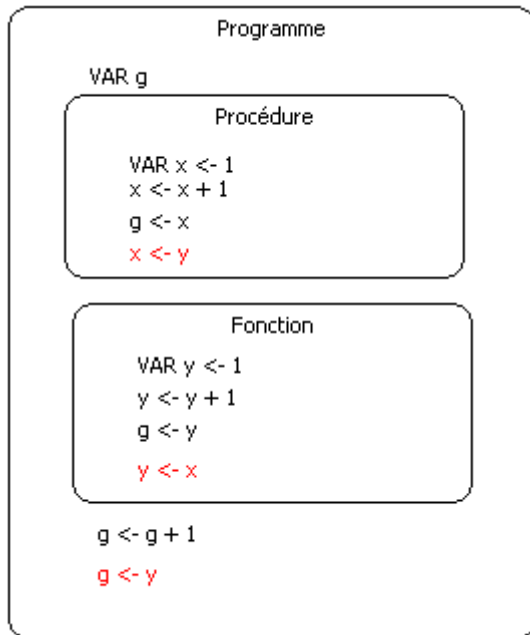
Le domaine de validité des objets locaux est le code la procédure et ceci implique :

- ✓ 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.



7.4 - PARAMÈTRES

7.4.1 - 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 plus 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 il est indiqué dans l'en-tête 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.

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.

7.4.2 - Le type de passage des paramètres

Il existe deux sortes de substitution de paramètres.

7.4.2.1 - 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. Donc la fonction utilise les valeurs réelles au moment de l'appel et les valeurs des paramètres réels au retour de la fonction restent inchangées même si les valeurs des paramètres ont changé. Ce sont les valeurs des paramètres formels qui ont changé. Lors de l'appel des **constantes** ou des **variables** peuvent être utilisées comme argument.

7.4.2.2 - 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 utilisés comme argument.

7.4.2.3 - Choix du type de passage

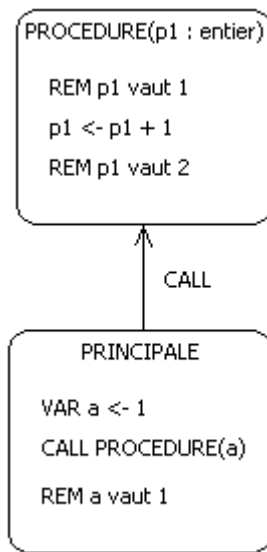
Le choix du type de substitution se fera selon les critères suivants :

- 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.

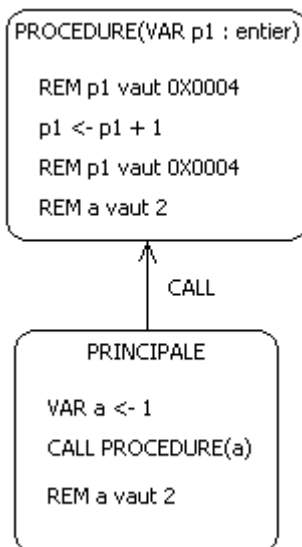
Jusqu'à présent ce qui a été vu était le premier type de substitution.

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

Passage par valeur



Passage par référence



7.5 - 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 l'instruction ou les instructions à exécuter encadrées des mots réservés DEBUT et FIN-FONCTION.

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 de la fonction**

C'est une expression.

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

IF(nom_de_la_fonction()=...)...

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
```

7.6 - LES PROCÉDURES : SYNTAXE

Comme cela a été dit une procédure est une fonction qui ne renvoie pas de résultat. Donc elle n'est pas typée et ne comprend aucune instruction RETOURNE. Une procédure est appelée avec l'instruction FAIRE et ne peut pas être utilisée dans une expression.

Syntaxes :

Ecriture de la procédure

```
PROCEDURE nomDeLaProcédure([parametre1 : TYPE[, parametre2 : TYPE]])  
  
VAR  
  
    identificateur-1 : TYPE  
    identificateur-2 : TYPE  
    identificateur-n : TYPE  
  
DEBUT  
    instruction-1  
    instruction-2  
  
    instruction-n  
FIN-PROCEDURE
```

Appel de la 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 nomDeLaProcédure([liste d'arguments])
```

Exemple

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

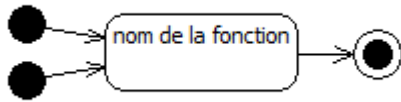
DEBUT
    p <- a
    a <- b
    b <- p
FIN-PROCEDURE

*----- 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
```

7.7 - DÉMARCHE POUR L'ÉCRITURE D'UNE FONCTION

Le schéma de la boîte noire.



Étapes	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

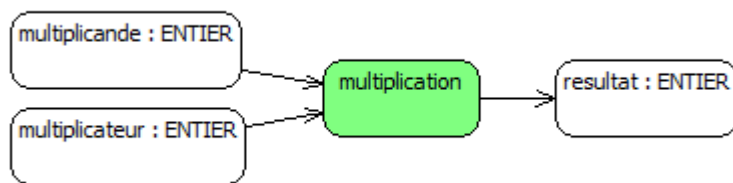
8 - EXEMPLES D'ALGORITHMES

8.1 - ALGORITHMES SUR LES NUMÉRIQUES

8.1.1 - Multiplication sans le signe de la multiplication

La multiplication est une suite d'additions.

Par exemple : $2 \times 3 = 2+2+2$.



```

FONCTION multiplication(multiplicande, multiplicateur : ENTIER) :
ENTIER

VAR
    ctr, resultat : ENTIER

DEBUT
    ctr <- 1
    resultat <- 0

    TANTQUE ctr <= multiplicateur FAIRE
        resultat <- resultat + nombre
        ctr <- ctr + 1
    FIN-TANTQUE

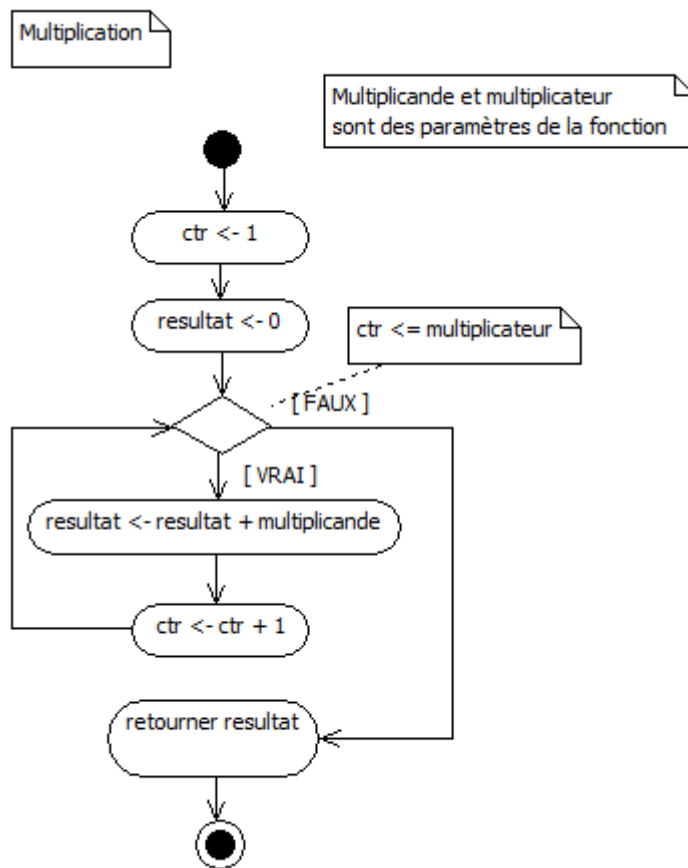
    RETOURNER resultat
FIN-FONCTION

VAR
    multiplicande, multiplicateur : ENTIER

DEBUT
    ECRIRE "Saisir un nombre : "
    LIRE multiplicande
    ECRIRE "Saisir un multiplicateur : "
    LIRE multiplicateur
    ECRIRE "Résultat : "
    ECRIRE multiplication(multiplicande, multiplicateur)
FIN

```

Passage	Multiplicande	Multiplicateur	Ctr	Résultat
Avant la boucle	2	3	1	0
1er passage	2	3	2	2
2ème passage	2	3	3	4
3ème passage	2	3	4	6
Après la boucle	2	3	4	6

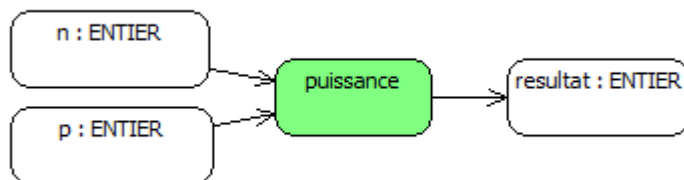


8.1.2 - Elévation à la puissance P d'un entier N

N est supérieur à 0 et P est positif.

Pour obtenir le résultat il faut multiplier P fois l'entier par lui-même.

Par convention un entier à la puissance 0 est égal à 1 et N puissance 1 est égal à N.



```

FONCTION puissance(n, p: ENTIER) : ENTIER
VAR
    ctr, resultat : ENTIER
DEBUT
    ctr <- 0
    resultat <- 1

    TANTQUE ctr < p FAIRE
        resultat <- resultat * n
        ctr <- ctr + 1
    FIN-TANTQUE

    RETOURNER resultat
FIN-FONCTION

VAR
    n, p: ENTIER
DEBUT
    ECRIRE "Saisir le nombre : "
    LIRE n
    ECRIRE "Saisir la puissance : "
    LIRE p
    ECRIRE "Résultat : "
    ECRIRE puissance(n, p)
FIN

```

2³

Passage	N	Puissance	Ctr	Résultat
Avant la boucle	2	3	0	1
1	2	3	1	2
2	2	3	2	4
3	2	3	3	8
Après la boucle	2	3	3	8

2¹

Passage	N	Puissance	Ctr	Résultat
Avant la boucle	2	1	0	1
1	2	1	1	2
Après la boucle	2	1	1	2

2⁰

Passage	N	Puissance	Ctr	Résultat
Avant la boucle	2	0	0	1
Après la boucle	2	0	0	1

8.1.3 - Exercices

1 - Calculez la factorielle d'un nombre.

2 - La division sans le signe de la division.

3 – Les nombres premiers de 1 à 100.

8.1.4 - Calculette

Il s'agit de simuler une calculette qui permet de faire les quatre opérations arithmétiques. Il est possible de mélanger les opérations.

La règle est la suivante : on saisit d'abord la première opérande puis l'opérateur puis la seconde opérande etc. On termine la saisie par le signe '='.

```

VAR
    rep, op : CAR
    x, res  : ENTIER

DEBUT
    rep <- 'O'
    TANTQUE rep = 'O' FAIRE
    DEBUT
        op <- '+'
        res <- 0
        TANTQUE op <> '=' FAIRE
            LIRE x
            AU CAS OU
                op = '+' : res <- res + x
                op = '-' : res <- res - x
                op = '*' : res <- res * x
                op = ':' : res <- res : x
            FIN-CAS
            LIRE op
            TANTQUE op <> '+' ET op <> '-' ET op <> ':' ET op <>
            '*' ET op <> '=' FAIRE
                LIRE op
            FIN-TANTQUE
        FIN-TANTQUE
        ECRIRE "Resultat : " + res
        ECRIRE "Continuer (O/N) : "
        LIRE rep
        TANTQUE rep <> 'O' ET rep <> 'N' FAIRE
            LIRE rep
        FIN-TANTQUE
    FIN-TANTQUE
FIN

```

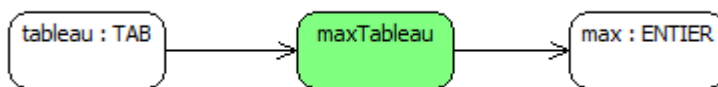
8.2 - ALGORITHMES SUR LES TABLEAUX

8.2.1 - Trouver la valeur MAX dans un tableau

On affecte la valeur du premier élément du tableau à une variable nommée max.

On parcourt le tableau du deuxième élément à la fin avec une boucle et on teste pour savoir si la valeur actuelle est supérieur à la valeur du max stocké. Si c'est le cas on affecte la valeur courante à la variable max.

On considère que l'on dispose d'une fonction nommé `compte()` qui renvoie le nombre d'éléments d'un tableau 1D.



```

FONCTION maxTableau(tableau : TAB[] ENTIER) : ENTIER

VAR
    nElements, ctr, max : ENTIER

DEBUT
    nElements <- compte(tableau)
    max <- tableau[1]
    ctr <- 2

    TANTQUE ctr <= nElements FAIRE
        SI tableau[ctr] > max ALORS
            max <- tableau[ctr]
        FIN-SI
        ctr <- ctr + 1
    FIN-TANTQUE

    RETOURNER max
FIN-FONCTION

VAR

tableau : TAB[7] ENTIER = [3,5,7,11,1,4,8]

DEBUT
    ECRIRE "MAX : "
    ECRIRE maxTableau(tableau)
FIN

```

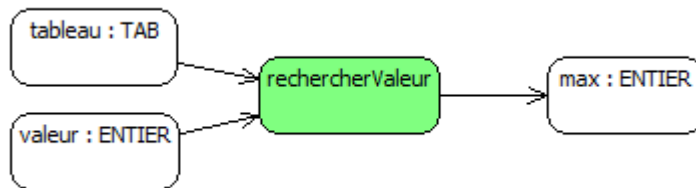
Passage	tableau	compte(tableau)	ctr	tableau[ctr]	max
Avant la boucle	[3,5,7,11,1,4,8]		2		3
1	idem	7	2	5	5
2	idem	7	3	7	7
3	idem	7	4	11	11
4	idem	7	5	1	11
5	idem	7	6	4	11
6	idem	7	7	8	11
Après la boucle	idem	7	8		11

8.2.2 - Rechercher une valeur dans un tableau

On affecte -1 au résultat.

On saisit la valeur recherchée.

On parcourt le tableau du début à la fin avec une boucle et on teste pour savoir si la valeur actuelle est égale à la valeur recherchée. Si c'est le cas on affecte la position courante au résultat.



```

FONCTION rechercherValeur(tableau : TAB[] ENTIER, valeurRecherchee :
ENTIER) : ENTIER
VAR
    nElements, ctr, rang : ENTIER

DEBUT
    nElements <- compte(tableau)
    rang <- -1
    ctr <- 1

    TANTQUE ctr <= nElements FAIRE
        SI tableau[ctr] = valeurRecherchee ALORS
            rang <- ctr
        FIN-SI
        ctr <- ctr + 1
    FIN-TANTQUE

    RETOURNER rang

FIN-FONCTION

VAR
    tableau : TAB[5] ENTIER = [3,5,7,11,1]
DEBUT
    ECRIRE "Valeur recherchée?"
    LIRE valeurRecherchee
    ECRIRE "RANG : "
    ECRIRE rechercherValeur(tableau, valeurRecherchee)
FIN

```

Passage	tableau	compte(tableau)	ctr	tableau[ctr]	rang
Avant la boucle	[3,5,7,11,1]		1		-1
1	idem	5	2	5	-1
2	idem	5	3	7	-1
3	idem	5	4	11	3
4	idem	5	5	1	3
Après la boucle	idem		6		3

Note : amélioration possible si l'on recherche la première occurrence ou si les valeurs sont uniques dans le tableau. La complexité passera de $O(n)$ à $O(n) / 2$.

Coût espace : 10 pour le tableau + 6 pour les variables.

Coût temps :

3 pour les 3 premières affectation,

10 pour le TQ,

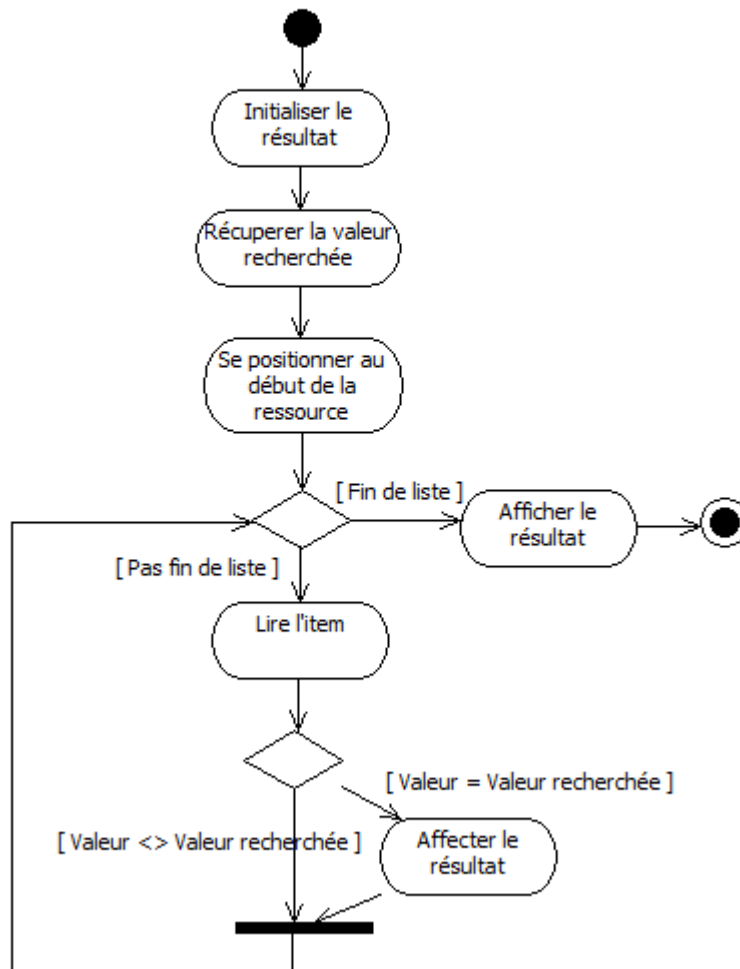
10 pour le SI,

10 pour l'incrément,

1 pour l'affectation interne au SI DONC 34 au total.



Recherche 1
Rechercher une valeur dans une ressource
(tableau, fichier, BD, ...)



Note : lire item signifie incrémenter un compteur s'il s'agit d'un tableau, lire l'enregistrement s'il s'agit d'un fichier, fetch record s'il s'agit d'une table, ...

8.2.3 - Exercices

1 - Modifiez l'algorithme pour l'améliorer en performance si on recherche seulement la première occurrence.

2 – Ecrivez les fonctions statistiques de base sur un tableau d'entiers (compte, min, **max (déjà vu)**, somme, moyenne, variance, écart-type).

8.3 - ALGORITHME SUR UN TABLEAU À 2 DIMENSIONS

8.3.1 - Somme des valeurs des éléments d'un tableau 2D

Remarques : imaginez la nécessité de calculer la somme des unités vendues par vendeur et par produit.

La matrice aura en ligne les vendeurs et en colonne les produits et aux intersections les quantités vendues.

```

FONCTION somme2D(t : TAB[] ENTIER) : ENTIER
VAR
    nLignes, nColonnes, I, J, somme : ENTIER
DEBUT
    somme <- 0
    I <- 1
    nLignes <- compte(t)
    nColonnes <- compte(t[I])

    TANTQUE I <= nLignes FAIRE
        J <- 1
        TANTQUE J <= nColonnes FAIRE
            somme <- somme + t[I,J]
            J <- J+1
        FIN-TANTQUE
        I <- I + 1
    FIN-TANTQUE

    RETOURNER somme
FIN

VAR
    tab2D : TAB[3,2] ENTIER
DEBUT
    tab2D <- [ [1,2], [10,20], [10,200] ]
    ECRIRE "Somme : "
    ECRIRE somme2D(tab2D)
FIN

```

8.3.2 - Exercice

Ecrivez l'algorithme qui permet de rechercher une valeur dans un tableau 2D.

8.4 - ALGORITHMES SUR LES CHÂÎNES DE CARACTÈRES

8.4.1 - Calcul de la longueur d'une chaîne

On fait saisir la chaîne de caractères par l'utilisateur.

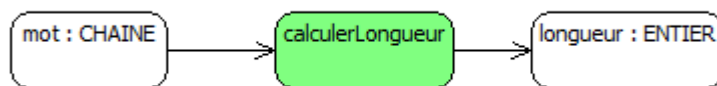
Par convention on établit qu'une chaîne de caractères est terminée par le caractère "\$".

Donc on concatène un '\$' à la chaîne à la chaîne saisie.

On examine la chaîne, caractère par caractère, et lorsque l'on rencontrera le "\$" on sortira de la boucle et l'on aura ainsi obtenu la longueur.

Il faut une variable pour la chaîne de caractères, et une variable pour l'indice.

Remarques : un seul paramètre est à passer, c'est la chaîne de caractères; et le résultat est la longueur c'est-à-dire un entier.



```

FONCTION calculerLongueur(texte : CHAINE) : ENTIER
VAR
    ctr : ENTIER
DEBUT
    texte <- texte & '$'
    ctr <- 1
    TANT QUE texte[ctr] <> '$' FAIRE
        ctr <- ctr + 1
    FIN-TQ
    ctr <- ctr - 1
    RETOURNER ctr
FIN-FONCTION

VAR
    mot : CHAINE

DEBUT
    ECRIRE "Saisissez un mot : "
    LIRE mot
    ECRIRE calculerLongueur(mot)
FIN

```

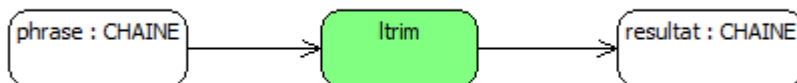
Passage	texte	ctr	texte[ctr]
Avant la boucle	Tintin\$	1	
1	Tintin\$	1	T
2	Tintin\$	2	i
3	Tintin\$	3	n
4	Tintin\$	4	t
5	Tintin\$	5	i
6	Tintin\$	6	n
7	Tintin\$	7	\$
Après la boucle	Tintin\$	6	\$

8.4.2 - Elimination des espaces en début de phrase

Admettons la phrase suivante " C'est un algo 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. Ensuite avec une deuxième boucle on décalera les caractères.



REM Elimination des espaces en début de phrase

FONCTION **ltrim**(texte : CHAINE) : CHAINE

VAR

 i,j : ENTIER

DEBUT

 texte <- texte & "\$"

 i <- 1

 j <- 1

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

 TANTQUE texte[i] <> "\$" FAIRE

 texte[j] <- texte[i]

 i <- i + 1

 j <- j + 1

 FIN-TANTQUE

 REM Pour déplacer le \$

 texte[j] <- texte[i]

 RETOURNER texte

FIN-FONCTION

VAR

DEBUT

 Ecrire "Saisir une phrase : "

 Lire phrase

 Ecrire "Résultat" & **ltrim**(phrase)

FIN

Test : La phrase saisie " il est"

i	j	texte
1	1	" il est\$"
2	1	" il est\$"
3	1	" il est\$"
3	1	"i il est\$"
4	2	"ilil est\$"
5	3	"il l est\$"
6	4	"il e est\$"
7	5	"il eest\$"
8	6	"il estst\$"
9	7	"il est\$t\$"

8.4.3 - Exercices

Ecrivez la fonction `gauche()` qui extrait les n premiers caractères d'une chaîne.

Ecrivez la fonction `extraire()` qui extrait n caractères d'une chaîne.

Ecrivez la fonction `trimInterne()` qui effectue un « TRIM » interne.

Note : dans tous les cas c'est à vous de définir les paramètres et leur type ainsi que le type de la fonction.

9 - ANNEXES

9.1 - QUELQUES MOTS-CLÉS UTILISÉS



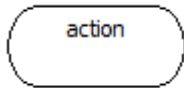

Ils sont en majuscules dans les algorithmes.

Mot-clé	Description
VAR	Déclaration de variables
CONST	Déclaration de constantes
ENTIER, REEL, BOOLEEN, CAR	Les types scalaires
CHAINE, TAB, ENREGISTREMENT	Les types non scalaires
LIRE	Lire une donnée
ECRIRE	Ecrire une donnée
DEBUT	Début d'un bloc d'instructions
FIN	Fin d'un bloc d'instructions
SI condition ALORS	Instruction de contrôle conditionnelle
SINON	
FIN-SI	
TANTQUE	Instruction de contrôle itérative
FIN-TANTQUE	
POUR	Instruction de contrôle itérative
FIN-POUR	

9.2 - ALGORITHMES ET ORDINOGRAMMES

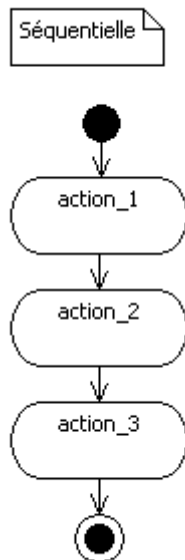
Les algorithmes peuvent être représentés sous forme graphique grâce aux ordinoigrammes.

Les représentations graphiques ont été réalisées avec StarUML. Ce sont des diagrammes d'activité UML.

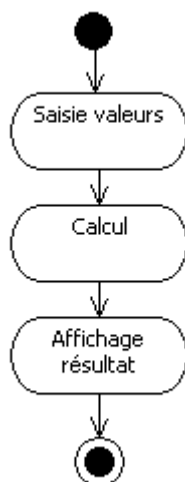
Symbole	Description
	Début
	Fin
	Action
	Condition

9.2.1 - La structure séquentielle

Générique

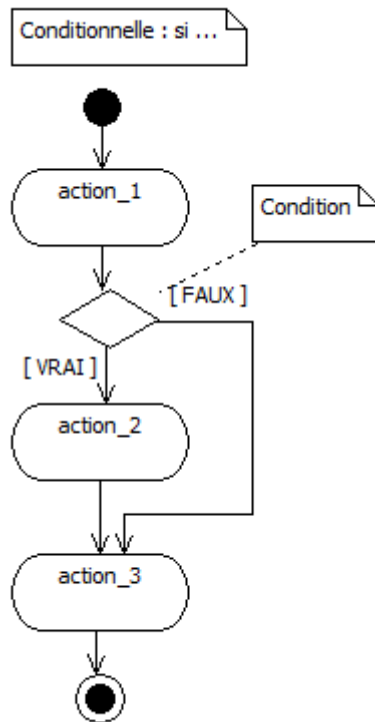


Exemple

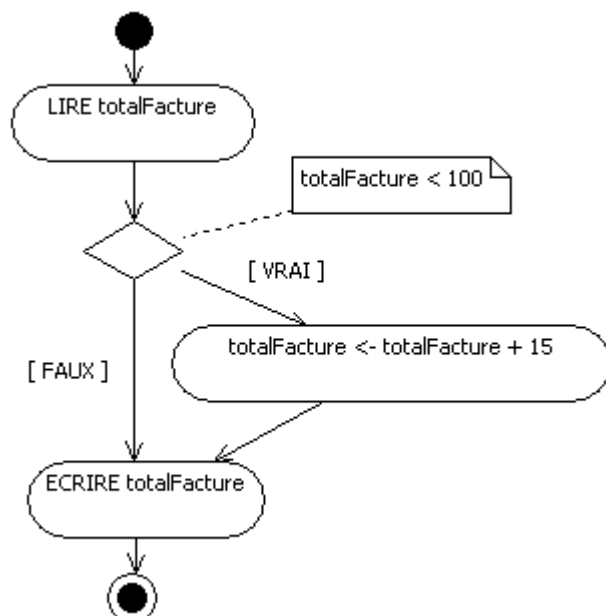


9.2.2 - La structure conditionnelle (SI)

Générique

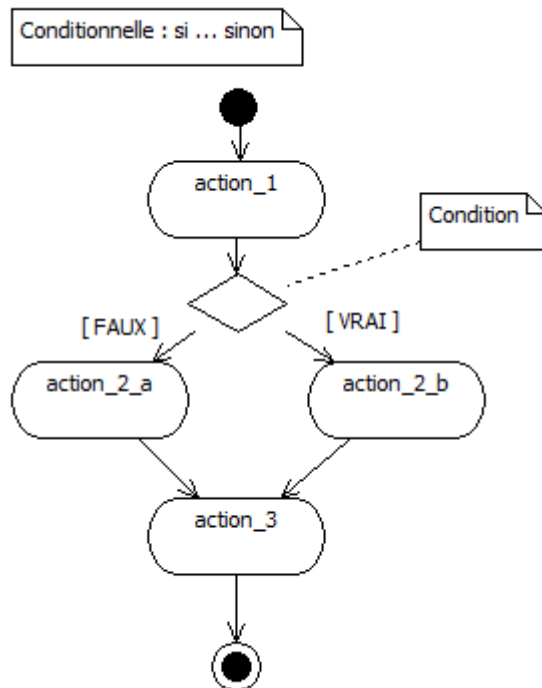


Exemple

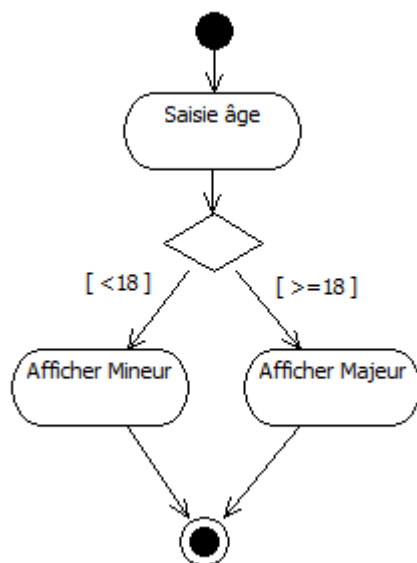


9.2.3 - La structure conditionnelle (SI ... SINON)

Générique

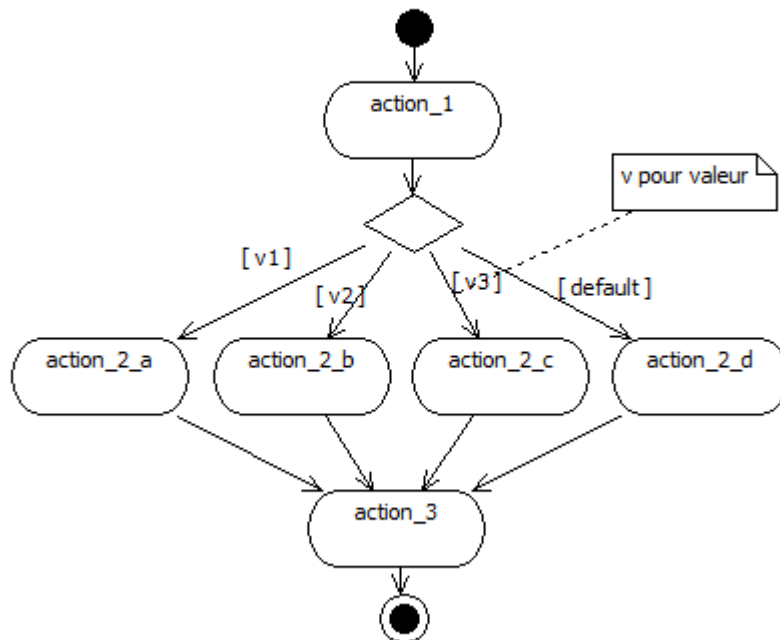


Exemple



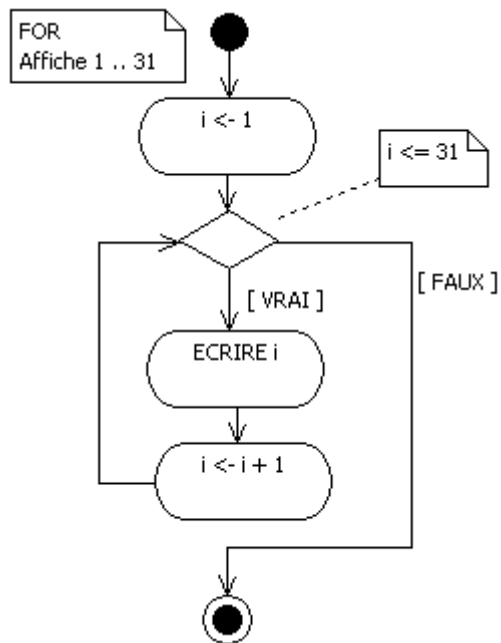
9.2.4 - La structure conditionnelle (AU CAS OU)

switch

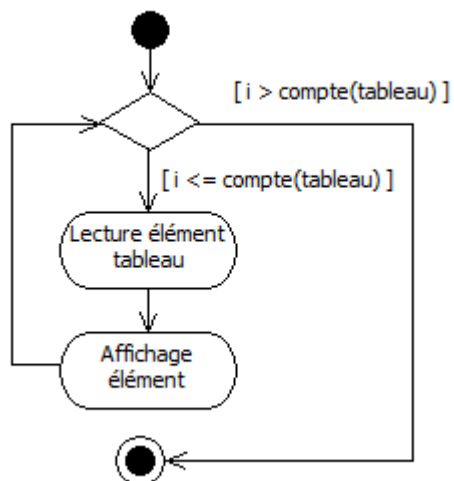


9.2.5 - La structure itérative (POUR)

Générique

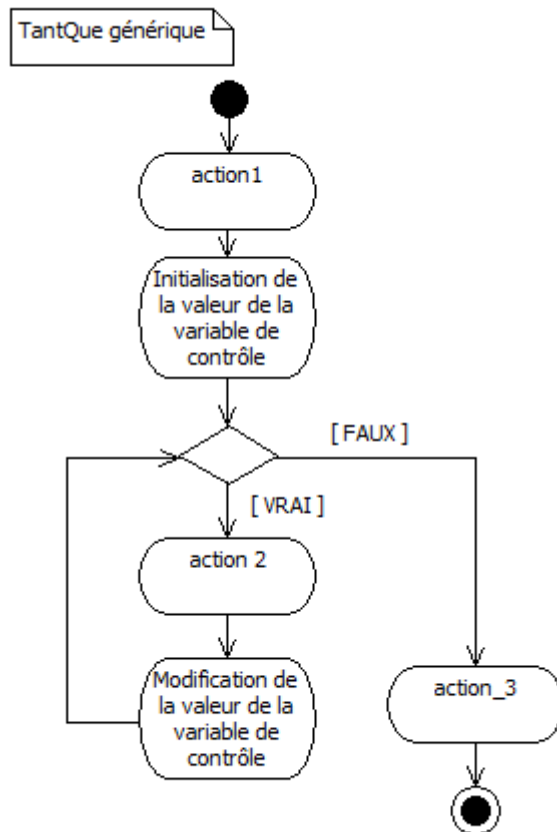


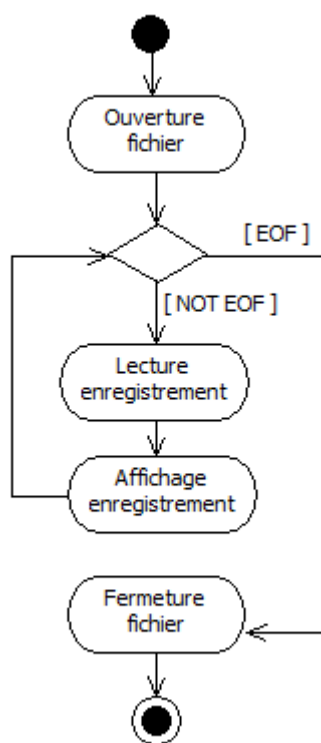
Exemple



9.2.6 - La structure itérative (TANTQUE)

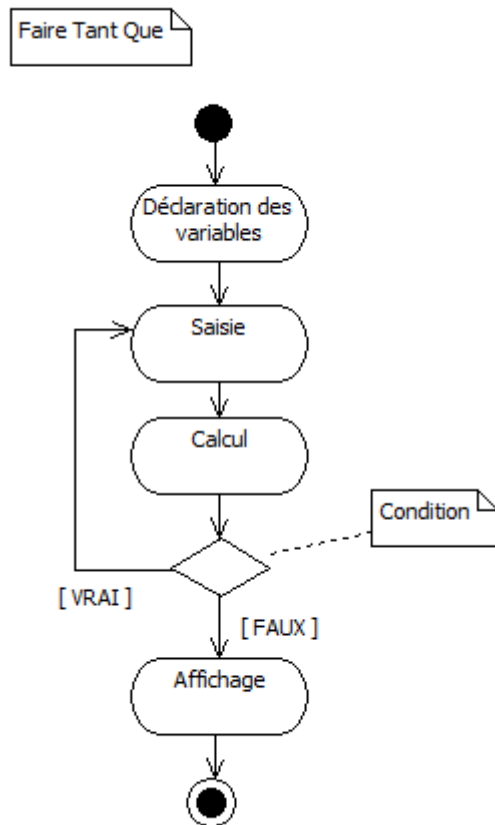
Générique



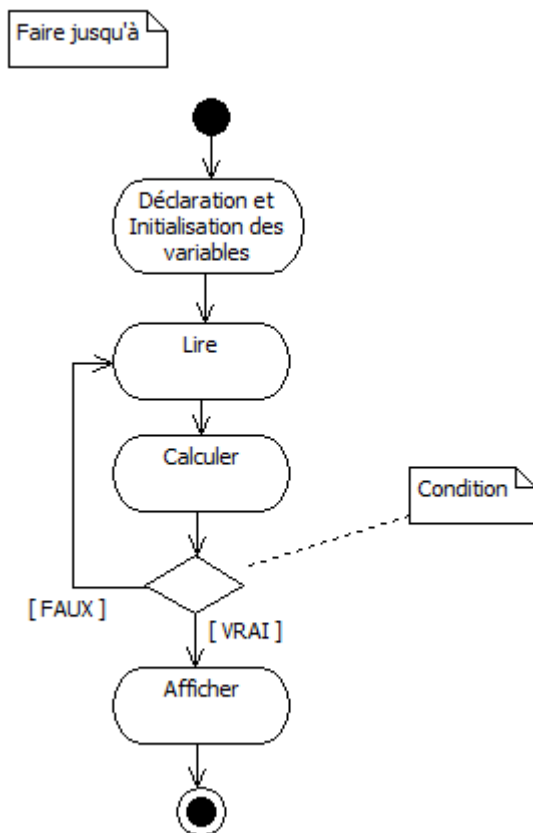
Exemple

9.2.7 - La structure itérative (FAIRE TANT QUE)

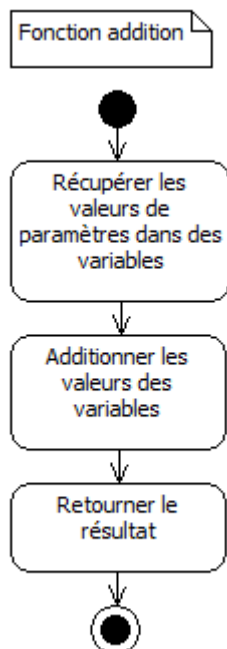
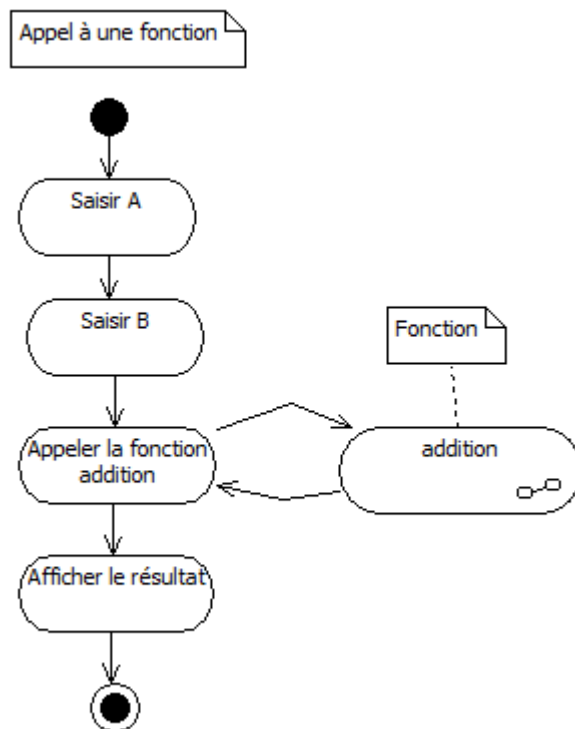
Générique



9.2.8 - La structure itérative (FAIRE JUSQU'A)



9.2.9 - Appel d'une fonction ou d'une procédure



9.2.10 - Exercices

Reprenez tous les exemples et exercices et représentez-les avec des ordinogrammes ! Si cela n'a pas déjà été fait.