

ALGORITHME : PSEUDO-CODE & ORGANIGRAMME

Sommaire

Définition du « Pseudo Code »	1
Exemple #1 : Calcul d'intérêt et de valeur acquise	1
Méthodologie	1
Formalisation de l'algorithme	2
Langage de description	2
Qualités d'un algorithme, d'un programme	3
Exemple #2 : Préparer une tasse de café soluble	4
Langage de description : Instructions	5
1° les instructions de lecture (d'entrée)	5
2° les instructions d'écriture (de sortie)	5
3° les instructions d'assignation (d'affectation)	5
Exemple #3 : Convertir un nombre de secondes en heure , minutes, secondes	6
Exemple #4 : Transformer une chaîne de caractère	6
Langage de description : Déclaration	7
Résumé sur le « Pseudo Code »	8
Définition d'un « Organigramme »	9
Convention d'écriture d'un organigramme	9
Les différentes structures d'organigramme	10
Transition linéaire	10
Transition alternative	11
Exemple : Fonctionnement d'une alarme de maison	12

Un algorithme peut se représenter de plusieurs manières :

- Le Pseudo Code
- L'organigramme

Définition du « Pseudo Code »

En programmation, le pseudo-code, également appelé LDA (pour **L**angage de **D**escription d'**A**lgorithmes) est une façon de décrire un algorithme en langage presque naturel, sans référence à un langage de programmation en particulier.

L'écriture en pseudo-code permet souvent de bien prendre toute la mesure de la difficulté de la mise en œuvre de l'algorithme, et de développer une démarche structurée dans la construction de celui-ci. En effet, son aspect descriptif permet de décrire avec plus ou moins de détail l'algorithme, permettant de ce fait de commencer par une vision très large et de passer outre temporairement certains aspects complexes, ce que n'offre pas la programmation directe.

Il n'existe pas de réelle convention pour le pseudo-code.

Exemple #1 : Calcul d'intérêt et de valeur acquise

Soit le problème suivant :

« Calcul de l'intérêt et de la valeur acquise par une somme placée à intérêt simple pendant un an »

L'énoncé du problème indique :

- Les données fournies : deux nombres représentant les valeurs de la somme placée et du taux d'intérêt.
- Les résultats désirés : deux nombres représentant l'intérêt fourni par la somme placée ainsi que la valeur obtenue après placement d'un an.

Il nous faut maintenant décrire les différentes étapes permettant de passer des données aux résultats. Nos connaissances générales nous permettent d'exprimer cette règle:

« Pour obtenir l'intérêt fourni par la somme, il suffit de multiplier la somme par le taux d'intérêt divisé par cent; la valeur acquise s'obtient en additionnant ce dernier montant et la somme initiale. »

Méthodologie

Dans cet exemple simple apparaissent les trois étapes qui caractérisent la résolution d'un problème sur ordinateur:

- 1) Comprendre la nature du problème posé et préciser les données fournies ("entrées" ou "input" en anglais).
- 2) Préciser les résultats que l'on désire obtenir ("sorties" ou "output" en anglais).
- 3) Déterminer le processus de transformation des données en résultats.

Ces trois étapes ne sont pas indépendantes et leur ordre peut être modifié.

Si les résultats fournis par l'ordinateur ne sont pas corrects, c'est qu'une erreur s'est glissée :

- soit dans l'analyse du problème
- soit dans la mise au point de l'algorithme
- soit dans sa traduction en langage de programmation car l'ordinateur ne fait qu'exécuter scrupuleusement les opérations demandées.

Formalisation de l'algorithme

En reprenant la méthodologie décrite ci-dessus, l'exemple précédent deviendrait:

- 1) Prendre connaissance de la somme initiale et du taux d'intérêt.
- 2) Multiplier la somme par le taux; diviser ce produit par 100; le quotient obtenu est l'intérêt de la somme.
- 3) Additionner ce montant et la somme initiale; cette somme est la valeur acquise.
- 4) Afficher les valeurs de l'intérêt et de la valeur acquise.

Il est évident, même sur cet exemple simple, qu'une telle formalisation risque de produire un texte long, difficile à comprendre et ne mettant pas clairement en évidence les différentes étapes du traitement.

Langage de description

Dans un langage de description, les actions sont généralement décrites par un symbole ou un verbe à l'infinitif choisi pour éviter les confusions. Ce langage est appelé pseudo-code.

En pseudo-code, notre exemple #1 devient:

- 1) écrire " Introduisez la somme initiale (en euros): "
- 2) lire somme_initiale
- 3) écrire " Introduisez le taux d'intérêt (ex: 3 pour 3%): "
- 4) lire taux
- 5) intérêt <-- somme_initiale * taux / 100
- 6) valeur_acquise <-- somme_initiale + intérêt
- 7) écrire " L'intérêt fourni est de " , intérêt , " euros"
- 8) écrire " La somme après un an sera de " ,valeur_acquise , " euros"

Nous pouvons remarquer deux verbes particuliers:

- lire qui correspond à la saisie, à l'introduction des données;
- écrire qui exécute l'affichage à l'écran ou l'impression des résultats.

Ces verbes sont soulignés pour indiquer qu'ils ont un sens particulier, qu'il est interdit de les utiliser dans un autre sens et qu'ils seront traduits pour être rendus compréhensibles par la machine.

Les valeurs manipulées dans cet algorithme sont des constantes (100) et des variables (somme_initiale, taux, intérêt, valeur_acquise). Il est pratique de choisir le nom des variables de manière à rappeler la signification de la valeur qu'elles représentent. Ce nom est souvent appelé identificateur de la variable. Les variables jouent le rôle de " tiroirs " dans lesquels on place une valeur durant l'exécution de l'algorithme.

Ainsi, « lire somme_initiale » signifie que l'on introduit dans le tiroir baptisé « somme_initiale » la valeur numérique entrée au clavier lors de l'exécution du programme.

Le contenu d'un de ces tiroirs peut être modifié en y plaçant le résultat d'un calcul. Cette instruction porte le nom d'assignation ou affectation et se représente par une flèche (<--).

Ainsi, « intérêt <-- somme_initiale * taux / 100 » signifie que l'on place dans le tiroir « intérêt » le résultat de l'opération figurant à droite de la flèche. Cette instruction se lit: « assigner à la variable intérêt la valeur de l'expression de droite ».

Les expressions symbolisant les calculs à effectuer sont représentées par des formules algébriques faisant intervenir les noms des variables, des symboles mathématiques ("+" pour l'addition, "-" pour la soustraction, "*" pour la multiplication, "/" pour la division, ...) et des constantes numériques.

La description d'une action et des objets qui y participent porte le nom d'instruction. L'ordre dans lequel les différentes opérations seront écrites indique l'ordre dans lequel elles seront exécutées: de haut en bas. Il s'agit d'une exécution séquentielle.

Qualités d'un algorithme, d'un programme

Tout programme fourni à l'ordinateur n'est que la traduction d'un algorithme mis au point pour résoudre un problème donné dans un langage de programmation.

Pour obtenir un bon programme, il faut partir d'un bon algorithme. Il doit, entre autres, posséder les qualités suivantes:

- Être clair, facile à comprendre par tous ceux qui le lisent (structure et documentation).
- Présenter la plus grande généralité possible pour répondre au plus grand nombre de cas possibles.
- Être d'une utilisation aisée même par ceux qui ne l'ont pas écrit et ce grâce aux messages apparaissant à l'écran qui indiqueront quelles sont les données à fournir et sous quelle forme elles doivent être introduites ainsi que les différentes actions attendues de la part de l'utilisateur.
- Être conçu de manière à limiter le nombre d'opérations à effectuer et la place occupée en mémoire.

Une des meilleures façons de rendre un algorithme clair et compréhensible est d'utiliser une programmation structurée n'utilisant qu'un petit nombre de structures indépendantes du langage de programmation utilisé.

Une technique d'élaboration d'un bon algorithme est appelée méthode descendante (top down). Elle consiste à considérer un problème dans son ensemble, à préciser les données fournies et les résultats à obtenir puis à décomposer le problème en plusieurs sous-problèmes plus simples qui seront traités séparément et éventuellement décomposés eux-mêmes de manière plus fine.

Exemple #2 : Préparer une tasse de café soluble

Imaginons un robot domestique à qui nous devons fournir un algorithme lui permettant de préparer une tasse de café soluble. Une première version de l'algorithme pourrait être:

- 1) Faire bouillir de l'eau
- 2) Mettre le café dans la tasse
- 3) Ajouter l'eau dans la tasse

Les étapes de cet algorithme ne sont probablement pas assez détaillées pour que le robot puisse les interpréter. Chaque étape doit donc être affinée en une suite d'étapes plus élémentaires, chacune étant spécifiée d'une manière plus détaillée que dans la première version.

Ainsi, l'étape « 1) Faire bouillir l'eau » peut être affiné en :

- 1.1) Remplir la bouilloire d'eau
- 1.2) Brancher la bouilloire sur le secteur
- 1.3) Attendre l'ébullition
- 1.4) Débrancher la bouilloire

De même,

« 2) Mettre le café dans la tasse » pourrait être affiné en :

- 2.1) Ouvrir le pot à café.
- 2.2) Prendre une cuillère à café.
- 2.3) Plonger la cuillère dans le pot.
- 2.4) Verser le contenu de la cuillère dans la tasse.
- 2.5) Fermer le pot à café.

« 3) Ajouter de l'eau dans la tasse » pourrait être affinée en :

- 3.1) Verser de l'eau dans la tasse jusqu'à ce que celle-ci soit pleine.

Certaines étapes étant encore trop complexes et sans doute incompréhensibles pour notre robot, il faut les affiner davantage.

Ainsi l'étape « 1.1) Remplir la bouilloire d'eau » peut nécessiter les affinements suivants:

- 1.1.1) Mettre la bouilloire sous le robinet
- 1.1.2) Ouvrir le robinet
- 1.1.3) Attendre que la bouilloire soit pleine
- 1.1.4) Fermer le robinet

Quand il procède à des affinements des différentes étapes, le concepteur d'un algorithme doit naturellement savoir où s'arrêter. Autrement dit, il doit savoir quand une étape constitue une primitive adéquate au point de ne pas avoir besoin d'affinement supplémentaire. Cela signifie évidemment qu'il doit connaître quelle sorte d'étape le processeur peut interpréter. Par exemple, le concepteur de l'algorithme précédent doit savoir que le robot peut interpréter "brancher la bouilloire" ce qui de ce fait n'exige pas d'affinement, mais qu'en revanche, il ne peut pas interpréter "remplir la bouilloire" et que dès lors un affinement devient nécessaire.

Langage de description : Instructions

Dans les algorithmes décrivant des calculs sur les quantités numériques, seront utilisées essentiellement les instructions que nous avons déjà étudiées.

1° les instructions de lecture (d'entrée)

lire variable

indique la saisie des données.

Exemples:

lire somme_initiale

lire taux

2° les instructions d'écriture (de sortie)

écrire expression

indique l'affichage d'un message et/ou du contenu d'une variable (ou du résultat d'un calcul).

Exemples:

écrire « Introduisez la somme initiale (en Euro): »

écrire « L'intérêt fourni est de « intérêt »

écrire intérêt

écrire a, b, (a+b)/2

3° les instructions d'assignation (d'affectation)

variable <-- expression

Exemples:

intérêt <-- somme-initiale * taux / 100

a <-- 0

i <-- i + 1

Les expressions sont des formules mathématiques symbolisant des opérations sur des variables et/ou des constantes numériques.

Les variables y sont représentées par un identificateur (un nom) comme en algèbre et les constantes sont des nombres écrits en chiffres.

Les opérations sur des nombres sont représentées par +, -, *, /.

D'autres fonctions mathématiques usuelles sont couramment utilisées: ln x, sin x, arctg x, [x] (signifie prendre la partie entière de x), a mod b (fournit le reste de la division de a par b), xy, loga x, ...

L'ordinateur peut également manipuler des variables contenant des chaînes de caractères alphanumériques pour les modifier, en extraire des sous-chaînes... Ces chaînes de caractères sont placées entre guillemets pour les distinguer des noms de variables. La concaténation (juxtaposition de 2 chaînes pour en former une nouvelle) est symbolisée par « + » séparant les 2 chaînes originelles. La fonction qui permet d'extraire une sous-chaîne est représentée par le nom de la variable avec en indice les positions des lettres à extraire.

Ainsi la sous-chaîne formée des caractères occupant les positions 2, 3, 4 dans la variable « prénom » sera symbolisée par:

« prénom2<--4 ».

Enfin, la fonction qui fournit la longueur (le nombre de caractères) de la chaîne contenue dans la variable prénom est symbolisée par |prénom|.

Exemple #3 : Convertir un nombre de secondes en heure , minutes, secondes.

Exprimer un nombre de secondes sous forme d'heures, minutes, secondes. La seule donnée fournie est le nombre total de secondes que nous appellerons « nsec ». Les résultats consistent en 3 nombres « h », « m », « s ».

Écrire " Introduisez le nombre de secondes"

lire nsec

s <-- nsec mod 60

m <-- (nsec / 60) mod 60

h <-- nsec / 3600

écrire nsec, "valent: ", h, "heure(s) ", m, "minute(s) et", s, "seconde(s)"

Exemple #4 : Transformer une chaîne de caractère

Donner la longueur du prénom et transformer un prénom et un nom en une chaîne contenant l'initiale du prénom séparée du nom par un point.

écrire "Quel est votre prénom?"

lire prenom

écrire "et votre nom?"

lire nom

pr <-- prenom1

lpr <-- |prenom|

identification<-- pr + "." + nom

écrire "Votre prénom de", lpr, "lettres a été abrégé et votre identification est : ", identification

Langage de description : Déclaration

Il est aussi nécessaire de préciser ce que les variables utilisées contiendront comme type de données. Il peut s'agir de nombres entiers, de nombres réels, de chaînes de caractères... Il faut faire précéder la description de l'algorithme par une partie dite déclarative où l'on regroupe les caractéristiques des variables manipulées.

La partie déclarative est placée en tête de l'algorithme et regroupe une ou plusieurs indications de la forme:

entier variables

ou

réel variables

L'algorithme complété de l'exemple #3 devient:

entier nsec

entier h

entier m

entier s

écrire "Introduisez le nombre de secondes:"

lire nsec

s <-- nsec mod 60

m <-- (nsec / 60) mod 60

h <-- nsec / 3600

écrire nsec, "valent: ", h, "heure(s)", m, "minute(s) et", s, "seconde(s)"

L'algorithme complété de l'exemple #4 devient:

entier lpr

chaîne prenom, nom, identification

écrire "Quel est votre prénom?"

lire prenom

écrire "et votre nom?"

lire nom

pr <-- prenom1

lpr <-- |prenom|

identification <-- pr + "." + nom

écrire "Votre prénom de", lpr, "lettres a été abrégé et votre identification est : ", identification

Résumé sur le « Pseudo Code »

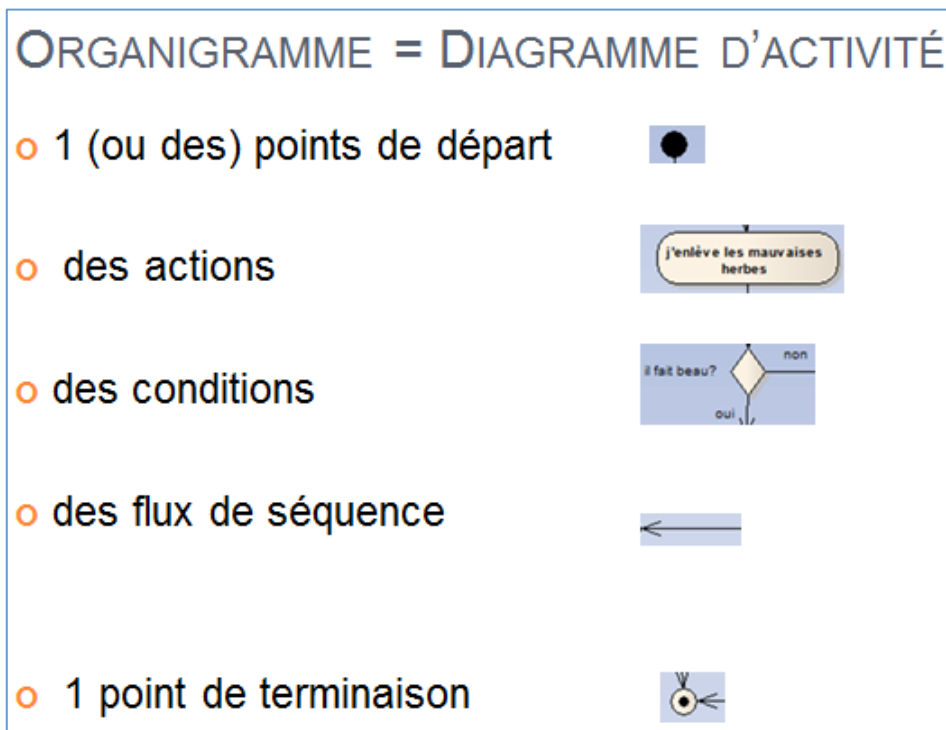
Pour l'échange de données entre le programme et l'utilisateur (ou le disque du PC), 2 mots sont utilisés:

- 1) **lire** : pour recevoir de l'info du monde extérieur:
 - a. **lire** N où N est le nom de la variable qui va recevoir l'information fournie par l'utilisateur
 - b. **lire** N sur Fichier où N est le nom de la variable qui va recevoir l'information récupérée dans le fichier Fichier.
- 2) **écrire** pour fournir de l'info au monde extérieur:
 - a. **écrire** "Bonjour tout le monde." où la partie entre guillemets est le message à afficher à l'écran.
 - b. **écrire** N où N est le nom de la variable qui contient l'information à écrire.
 - c. **écrire** N sur Fichier où N est le nom de la variable qui contient l'information à écrire sur le fichier Fichier.
- 3) Lorsque le programme travaille, on utilise l'assignation **<--** pour symboliser la mémorisation dans une variable :
 - a. $N \leftarrow N+2$
 - b. $St \leftarrow \text{"Hello"}$

Définition d'un « Organigramme »

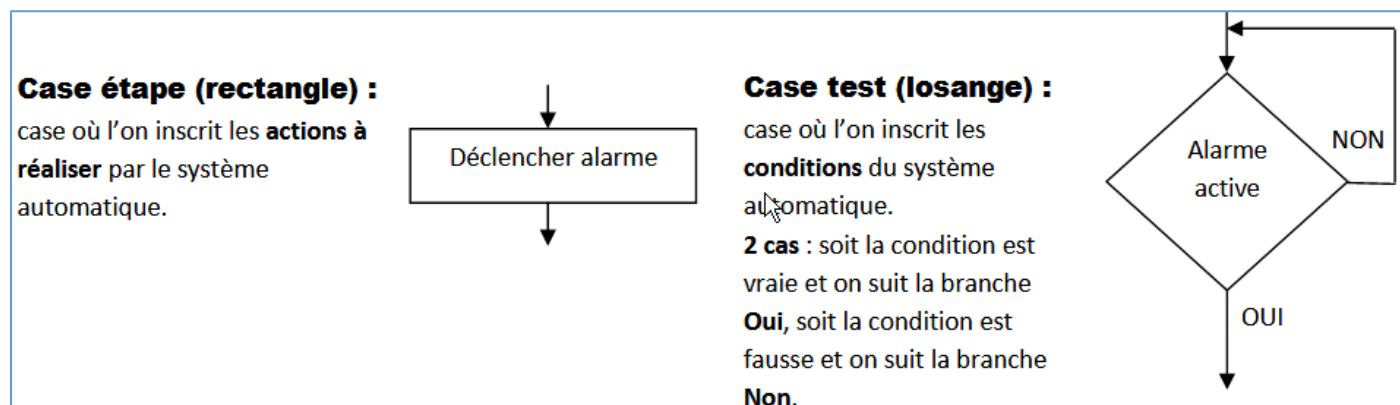
En programmation, l'organigramme est une représentation graphique normalisée de l'enchaînement des opérations et des décisions effectuées par un programme informatique.

Un organigramme peut également être appelé « algorithme » ou « logigramme » et est composé des éléments suivants :



Convention d'écriture d'un organigramme

Chaque « case » d'un organigramme possède une fonction précise.



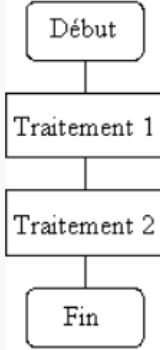
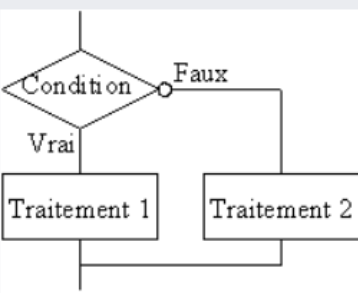
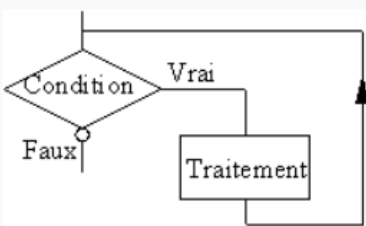
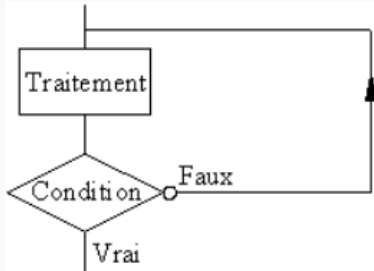
Le sens par défaut des liens du flux d'exécution est :

Du haut vers le bas pour les liens verticaux.

De la gauche vers la droite pour les liens horizontaux.

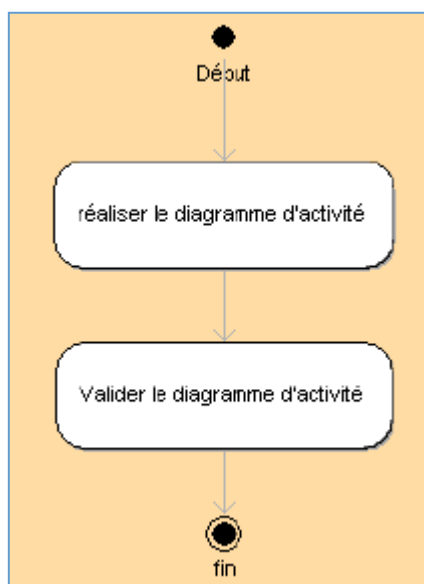
Lorsque le sens par défaut n'est pas respecté, il est nécessaire de le préciser par une flèche à l'extrémité du lien.

Les différentes structures d'organigramme

Séquence linéaire	Séquence alternative « si...alors...sinon »	Séquence répétitive « tant que...faire... »	Séquence répétitive « répéter...jusqu'à... »
			
Début <ul style="list-style-type: none"> • « Traitement 1 » • « Traitement 2 » Fin	Si « condition » <ul style="list-style-type: none"> • alors « Traitement 1 » • sinon « Traitement 2 » Fin si	Tant que « condition » <ul style="list-style-type: none"> • faire « traitement » Fin tant que	Répéter « traitement » jusqu'à « condition »

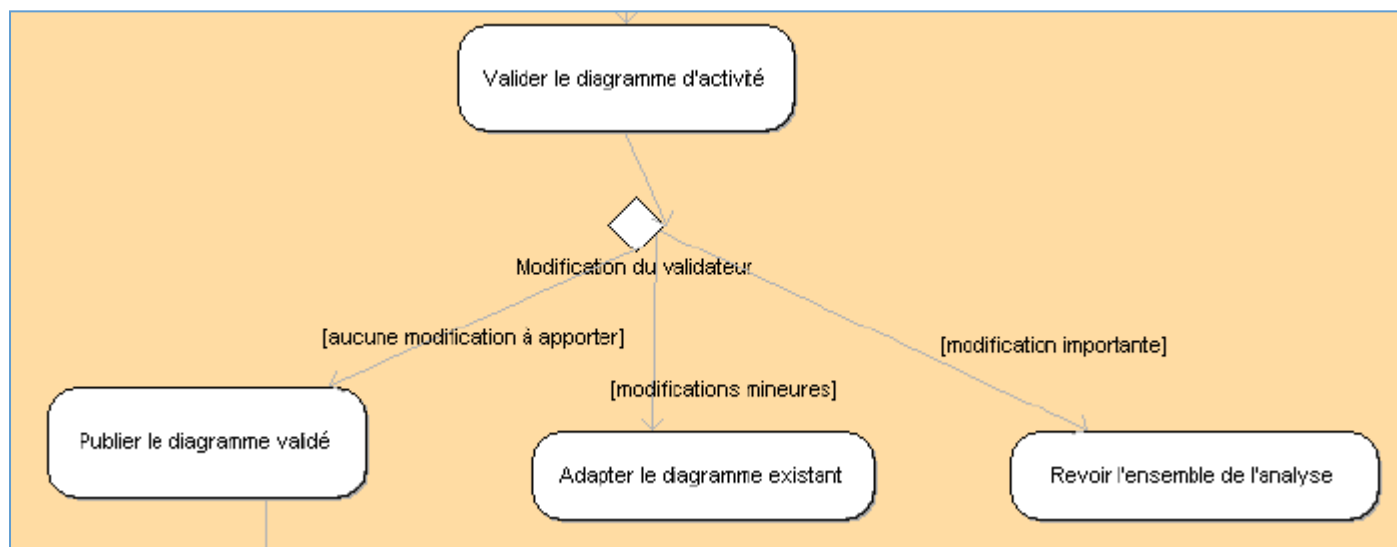
Transition linéaire

Dans un organigramme, la transition entre les éléments se fait séquentiellement. Plus concrètement, une action démarre lorsque l'action précédente se termine.



Transition alternative

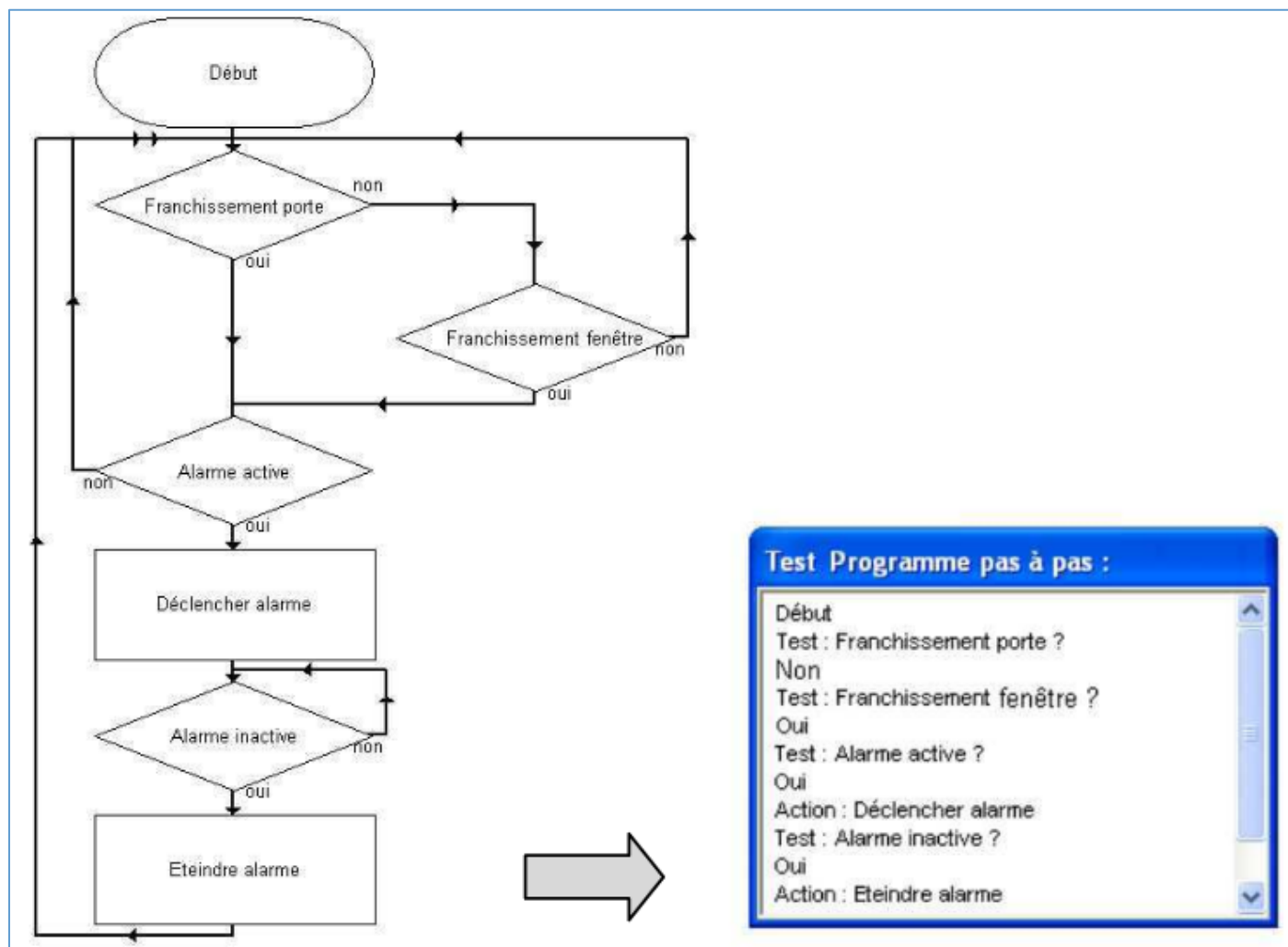
Les transitions alternatives (dépendant d'une condition) mènent vers des actions différentes.



Exemple : Fonctionnement d'une alarme de maison

Soit le problème suivant :

- Si quelqu'un franchit la porte ou une fenêtre de la maison, et si l'alarme est active à ce moment-là :
L'alarme sonore se déclenche.
- L'alarme s'arrête lorsque l'on désactive le système d'alarme.



--- FIN DU DOCUMENT ---

<http://www.arfp.asso.fr>