Algorithmique (VBA) - Sommaire

1. INTRODUCTION 2
2. DEMARCHE DE CONCEPTION D'UN ALGORITHME ET NOTATIONS 6
3. IDENTIFICATION DES OBJETS : DECLARATIONS DES DONNEES 13
4. L'AFFECTATION 17
5. EXPRESSIONS DE CALCULS NUMERIQUES ET LOGIQUES 19
6. ECHANGES AVEC L'UTILISATEUR : ENTREES/SORTIES 22
7. STRUCTURE DE CONTROLE CONDITIONNELLE : TESTS, DECISION 24
8. STRUCTURE DE CHOIX MULTIPLES 28
9. STRUCTURE DE CONTROLE ITERATIVE : BOUCLES 29
10. MODULARITE ET SOUS-PROGRAMMES : FONCTIONS ET PROCEDURES 32

Algorithmique – Concepts de base

1. Introduction

Objectifs du module :

- Concevoir une solution algorithmique à partir de l'expression d'un besoin
- Comprendre un algorithme et le faire évoluer
- Evaluer, contrôler et valider l'algorithme

Algorithme, procédé systématique

Un **algorithme** décrit la succession des actions élémentaires qui, mécaniquement appliquée, vont permettre l'obtention d'un résultat souhaité à partir d'éléments de base. Il est généralement conçu pour être exécuté plusieurs fois. Le résultat doit être identique à chaque exécution, pour les mêmes éléments de base, dans des conditions de fonctionnement identiques et en un temps d'exécution déterminé.

(Synonymes : procédure, mode opératoire, recette, programme, etc.)

Exemple : La recette de la tarte au chocolat de Frédéric Anton est un bon exemple d'algorithme ! (Source http:' www.femmeactuelle.fr/cuisine/recettes/tarte-au-chocolat-de-frederic-anton)

Ingrédients

- 180 g de farine
- 70 g de sucre glace
- 75 g de beurre
- 3 jaunes d'oeuf

Pour la ganache

- 50 g de lait
- 120 g de crème fleurette
- 120 g de chocolat à 60%
- 12 g de beurre
- 2 petits oeufs

Éléments de base

Préparation

Pour la pâte à tarte (vous pouvez également acheter une pâte sablée ou brisée toute faite) :

Tamiser la farine et le sucre glace. Mélanger le beurre pommade (le faire sabler). Quand il ne reste plus de gros morceaux de beurre, incorporer les jaunes d'oeufs. Réserver au froid pendant 30 minutes. Abaisser à 3 mm d'épaisseur et foncer un cercle de 18 cm de diamètre. Cuire la pâte à blanc à 160° pendant environ 10 minutes. La sortir du four et la laisser refroidir.

Pour la ganache :

Préchauffer le four à 170°C.

Concasser le chocolat dans un cul de poule. Dans une casserole, faire bouillir le lait, le beurre et le crème. Verser ce mélange sur le chocolat. Mélanger délicatement en veillant à ne pas faire de bulles. Battre légèrement les oeufs, puis les ajouter à la préparation. Verser ce mélange dans le fond de la tarte préalablement cuite à blanc, éteindre le four et enfourner pendant 16 minutes pour laisser cuire doucement à four éteint.
Fermer le four, laisser cuire 15 minutes.
Vérifier la cuisson en tapotant légèrement, la tarte doit être légèrement tremblotante. Et sinon vous pouvez remettre la tarte en cuisson cinq à quinze minutes de plus.

La recette de la tarte au chocolat en vidéo

Actions

A partir de la liste des ingrédients, la succession d'actions va progresser ver
l'obtention du résultat attendu ➔

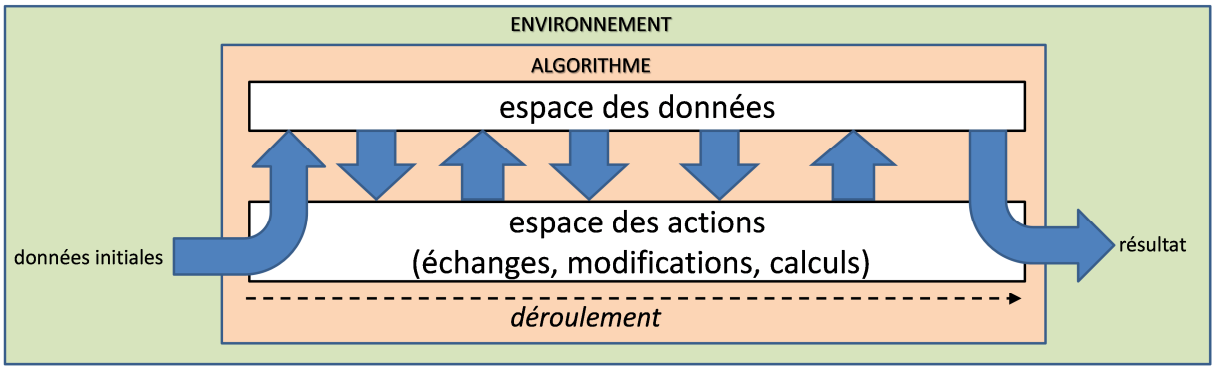
Mais nous traiterons, pour notre part, d'algorithmes informatiques…

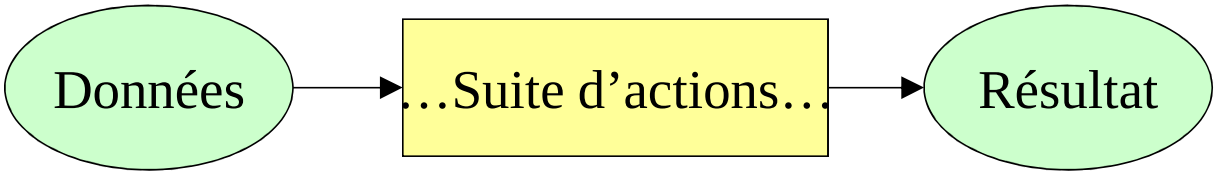
Algorithmique – Concepts de base

Un **algorithme informatique** est un enchaînement d'actions élémentaires nécessaire à la résolution de problèmes généralement calculatoires[1] à partir de données numériques[2] et dont l'implémentation sera réalisée dans un langage de programmation. Il représente la conception/maquette/le plan d'un programme informatique à venir.

Au même titre que notre recette, un algorithme informatique définit 2 espaces :

- l'espace des éléments de base, ici les données mises en œuvre (*représentation numérique des objets mis en œuvre dans la résolution*)
- l'espace des actions permettant, à partir des données, d'opérer des transformations (*modifications et calculs*) pour obtenir le résultat attendu





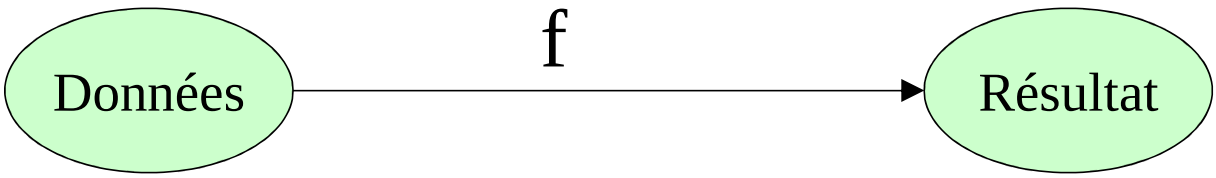
À un problème donné, peuvent correspondre

- plusieurs algorithmes qui fournissent le bon résultat, en utilisant des données différentes et/ou des calculs différents : il sera nécessaire de choisir le meilleur d'entre eux en étudiant leurs caractéristiques de complexité :
  - spatiale : la quantité d'espace utilisé pour les données et l'évolution de cette quantité en fonction des données fournies
  - temporelle : la durée d'exécution et l'évolution de cette durée en fonction des données fournies
- ou aucun algorithme : ces problèmes sont dits « indécidables »

L'algorithme fournit un service (*il a une utilité, une fonction*) grâce au fonctionnement de sa structure interne (*ensemble de données et d'actions*).

Fonctions mathématiques et fonctions informatiques

Un algorithme est assimilable à une fonction qui, à des données initiales, associe le résultat de leur transformation, avec cependant quelques différences



[1] = mettant en œuvre des calculs
[2] Représentations codées du monde réel

Une définition formelle en mathématiques

Une fonction $f : X \rightarrow Y$ est un ensemble E inclus dans ou égal à $X \times Y$ de couples (x, y) avec x appartenant à X et y appartenant à Y tel que x n'apparait qu'une fois comme composant d'un couple.

La méthode du calcul est peu, ou pas du tout, exprimée.

Une définition procédurale en informatique

Une fonction $f : X \rightarrow Y$ est une méthode de calcul qui décrit comment[3] obtenir une valeur y appartenant à Y à partir d'une valeur x appartenant à X. Plusieurs méthodes de calcul (*algorithmes*) peuvent être proposées pour décrire le mode de calcul d'une fonction mathématique.

L'algorithmique

L'algorithmique est la partie de l'informatique qui s'intéresse à « *l'art, l'ensemble de théories, les règles et techniques permettant de concevoir des algorithmes* ».

Il s'agit, face à un problème à résoudre, d'en percevoir les éléments clés en termes d'informations et de succession d'opérations logiques les plus efficaces afin d'aboutir à sa résolution de manière la plus efficace.

Pour la petite histoire…

La notion d'algorithme remonte à l'antiquité (3 millénaires avant JC, chez les babyloniens, en Mésopotamie, actuellement l'Irak) et décrivait des méthodes de résolution d'équations. Le mathématicien grec Euclide (plus récemment, 3 siècles av. JC) a décrit la détermination du plus grand diviseur commun de 2 nombres sous forme d'une suite de calculs.

Mais c'est bien plus tard qu'on a pu nommer ces démarches « algorithme ». C'est en effet du nom d'un mathématicien perse (vers 820 ap. JC, actuel Ouzbékistan), Al-Khawarizmi, qu'on a introduit, 600 ans plus tard, vers 1550, le terme « algorithme ».

Algorithme et programme informatique

Une fois l'algorithme informatique validé, il est implémenté dans un langage de programmation afin d'être ensuite traduit pour être exécutable par un automate programmable.

Programme source

Un programme source, ou code source, est le résultat de la traduction d'un algorithme dans un langage de programmation.

Il est stocké dans un fichier au format texte[4], dont l'extension varie selon le langage (*.C et .h pour le C, .CPP et .h pour le C++, .java pour JAVA, etc.*)

Programme exécutable

Le programme source est généralement compilé[5] pour former un programme exécutable[6] par une machine, généralement l'ordinateur.

Il est stocké dans un fichier binaire[7] dont l'extension varie selon le Système d'Exploitation ou le langage de programmation. *(.class pour Java, .exe sous Windows, pas d'extension particulière sous Linux, une propriété du fichier indique alors que le fichier est exécutabl*e).

Ainsi, à partir d'un besoin exprimé par un utilisateur (*le client*), l'informaticien[8]

[3] la liste des actions

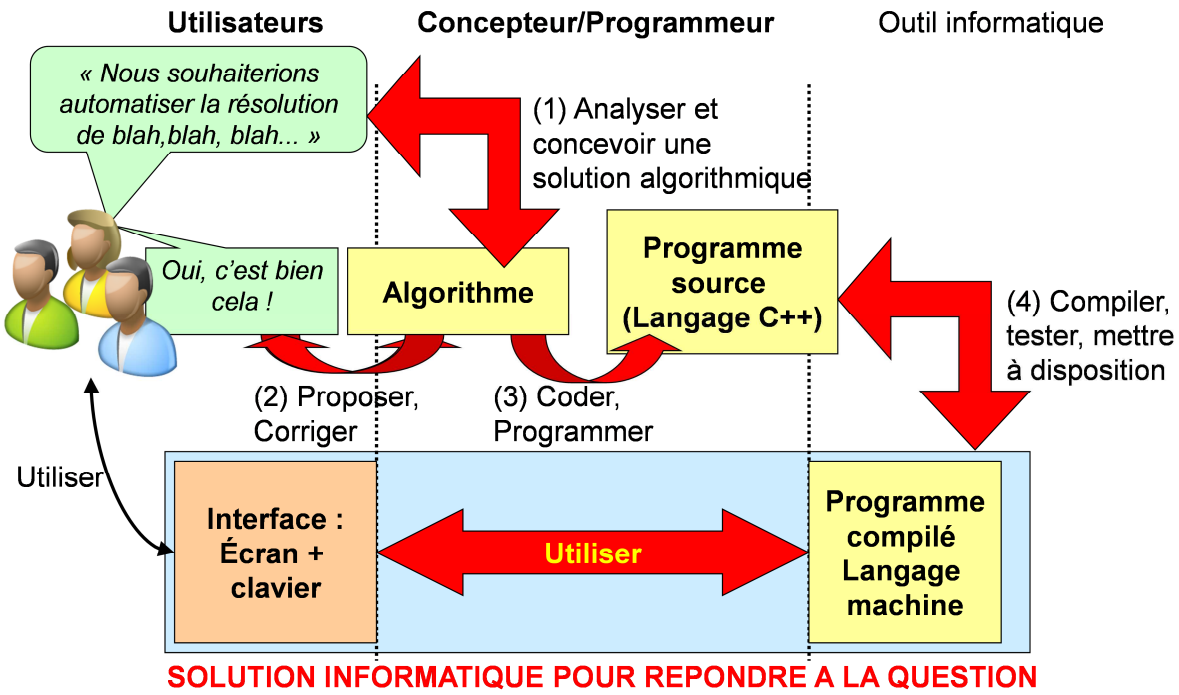
[4] Un fichier texte comporte une suite de caractères affichables ; on peut ouvrir ce fichier en utilisant un simple éditeur de texte comme BlocNote

[5] Compilation : traduction d'un programme source, exprimé dans un langage de programmation, en un programme exécutable, exprimé en langage machine

[6] La compilation correspond généralement à une traduction des instructions du code source en instructions machine

[7] Le fichier binaire comporte une succession de codes (données et instructions) correspondant à des instructions et des adresses mémoire ; il n'est pas lisible en utilisant un simple éditeur de texte.

1. analyse cette demande
2. il conçoit une solution algorithmique
3. il la propose et éventuellement l'ajuste ou la corrige en fonction des retours du client
4. puis il la code dans un langage de programmation, compile puis teste le programme ainsi réalisé
5. il en assure enfin la livraison au demandeur qui pourra ainsi l'utiliser



[8] Version très simplifiée de la « fabrication des produits logiciels » (ingénierie logicielle), travail d'équipe qui part d'un cahier des charges rigoureux, comporte de nombreuses phases de validation avant d'aboutir à la livraison

2. Démarche de conception d'un algorithme et notations

La démarche classique de production logicielle comporte un certain nombre de phases qui nécessitent la mise en œuvre de moyens plus ou moins importants, en fonction de la taille du problème à traiter.

Approches descendante et ascendante

Deux approches opposées, mais souvent utilisées de manière complémentaires, sont utilisées pour analyser un problème afin d'en concevoir la résolution sous forme d'un algorithme :

- l'approche descendante (*anglais : top-down*):
  - **décomposition** d'un problème en sous-problèmes de moindre complexité par affinements successifs, pour obtenir des éléments de résolution élémentaires ;
- l'approche ascendante (*anglais bottom-up*) :
  - **composition** de la résolution d'un problème par assemblage d'éléments de résolution connus, ceci dans un ordre optimal

Les étapes de la conception de la résolution

Analyse du problème

Il s'agit d'y définir de manière précise, complète, cohérente et non ambiguë les besoins :

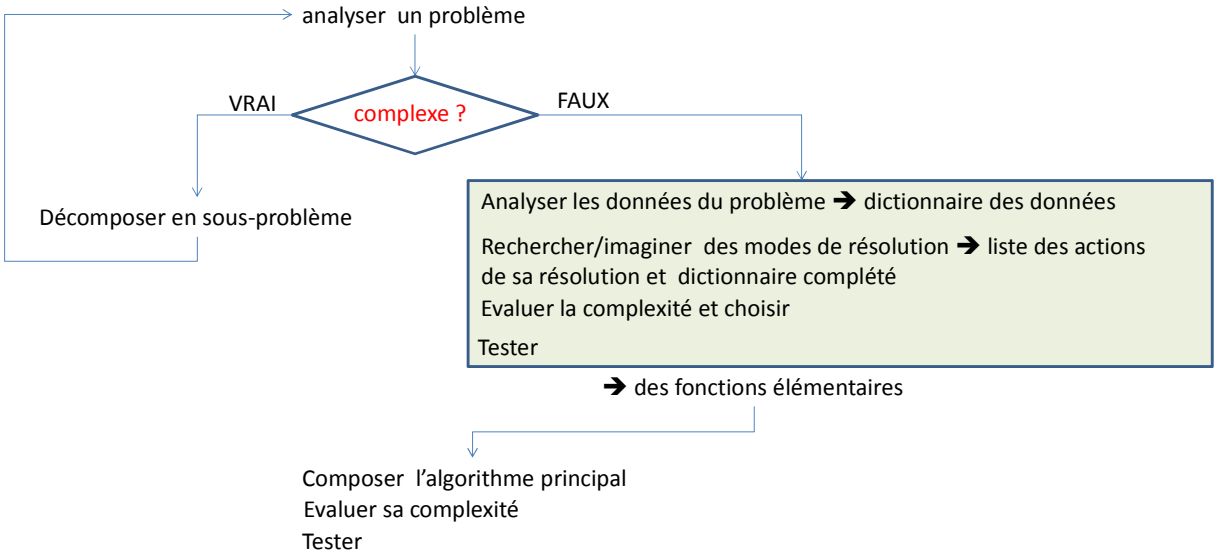
- reformuler le problème
- formaliser le problème et sa solution

1. analyser l'énoncé du problème : lire, questionner, comprendre afin de lever toute ambiguïté

➔ Reformuler la question ou le problème de manière plus formelle

Conception générale de la solution

Il s'agit de traduire l'énoncé dans un langage compréhensible par tous en utilisant une méthode de décomposition descendante et un langage graphique pour mettre en évidence les sous-problèmes.



1. décomposer le problème en sous-problèmes et proposer une organisation de sa résolution, spécifier les modes de résolution connus et/ou imaginer un mode de résolution *(décomposition)*

➔Schéma général de décomposition des éléments composants la solution

Conception détaillée

Il s'agit de détailler les éléments mis en évidence dans l'étape précédente et de proposer des algorithmes permettant de fournir une solution aux sous-problèmes posés.

C'est ici qu'interviendra une notation pour définir les algorithmes sans ambiguïté et de manière à ce qu'ils soient lisibles par tous ; elle sera concise et indépendantes des langages informatiques et de leurs spécificités.

Pour chaque algorithme composant la solution :

1. Spécifier les données

   ➔Liste des données : données initiales, données intermédiaires, résultat, constantes

2. Concevoir l'enchainement optimal d'actions visant à résoudre le problème et fournir le résultat escompté *(composition de la solution).* On sera parfois amené à modifier la liste des données

   ➔Liste des actions qui vont s'enchainer pour obtenir le résultat souhaité

3. Représenter l'algorithme en utilisation la notation

   ➔déclarer les données

   ➔définir les actions en 3 grandes parties : initialiser les données, effectuer le traitement des données et fournir le résultat

4. évaluer la complexité de l'algorithme
5. tester l'algorithme (jeux d'essai : données vraisemblables et données aux limites)

Dans les cas plus complexes où la résolution exige la mise en œuvre de plusieurs algorithmes, conception de la solution globale :

1. concevoir l'enchainement optimal des algorithmes précédents (**composition**)
2. évaluer la complexité de l'ensemble
3. tester la solution complète (jeux d'essai : données vraisemblables et aux limites)

Codage du programme informatique

Une fois l'algorithme construit et validé, il va s'agir ensuite de traduire cet algorithme dans un langage de programmation sélectionné, puis de compiler, tester les programmes avant de les mettre à disposition des utilisateurs.

1. Coder les algorithmes dans le langage cible

   ➔ code source du programme

2. Compiler le programme

   ➔ code exécutable

3. Tester le programme exécutable

Livraison, formation, maintenance

1. Mise à disposition de l'utilisateur

   ➔ Documentation

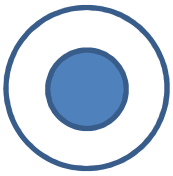
   ➔Formation

2. Maintenance (corrective et évolutive)

Un exemple simple :

Problème

Trouver l'aire d'une piste circulaire ? (*La piste correspond à la partie blanche ci-dessous, les 2 cercles sont concentriques*).



Analyse du problème

Reformuler le problème de manière plus formelle et non ambiguë

Soient un disque d1 de rayon r1, nombre réel, et un disque d2 de rayon r2,nombre réel, r2 étant inférieur à r1, aire1 et aire2 étant les aires respectivement de d1 et d2, calculer (aire1 – aire2). Les 2 disques sont concentriques. On ne tiendra pas compte de l'épaisseur du trait.

Remarque : dans cet exemple simplifié, les préconditions de bon fonctionnement suivantes ne seront pas vérifiées :

- r1 ne doit pas être inférieur à r2
- r2 ne doit pas être inférieur à 0

Découvrir les éléments de résolution connus

- Le mode de calcul de l'aire d'un disque est connu : PI * rayon * rayon
- La valeur de PI est connue : 3.14159…

Organisation générale

a) demander les valeurs de r1 et r2
b) effectuer les traitements : calculer l'aire du disque 1, puis celle du disque 2 et calculer la différence
c) et fournir le résultat, l'aire calculée

Données utiles

désignation	type / domaine de définition	identifiant	valeur
PI	réel	PI	3.14159
rayon du disque 1	réel	r1	(demandé)
rayon du disque 2	réel	r2	(demandé)
aire du disque 1	réel	aire1	(calculé)
aire du disque 2	réel	aire2	(calculé)
aire résultat	réel	aire	(calculé)

Liste des actions à réaliser : algorithme initial

Voici une proposition d'une liste d'actions qui vont permettre d'aboutir au résultat ; c'est une première version d'un algorithme :

1) demander le rayon du disque extérieur, r1
2) demander le rayon du disque intérieur, r2
3) calculer aire1 = PI * r1 * r1
4) calculer aire2 = PI * r2 * r2

5) calculer aire = aire1 – aire2
6) donner la valeur de l'aire : aire
7) terminer

Cependant la formulation en prose laisse libre cours à l'interprétation quant à la signification de certains termes : demander, indiquer, donner…

Notations algorithmiques

Une notation algorithmique est une représentation simplifiée *(une forme de langage*) permettant de définir sans ambiguïté un algorithme informatique (ou autre) de manière à ce qu'il soit lisible par tous.

Les primitives d'un langage algorithmique sont les actions élémentaires à partir desquelles on pourra construire toutes sortes d'algorithmes.

Notation textuelle, ou « pseudocode »

Cette notation définit un langage à mi-chemin entre le langage naturel et les langages de programmation classiques, sans cependant tomber dans les particularités d'un langage de programmation spécifique.[9] Elle utilise une syntaxe et une grammaire simples mais strictes afin de ne laisser aucune ambiguïté sur la sémantique d'un algorithme. Elle n'est pas normalisée, mais on trouve toujours représentées :

- une partie « Déclarations » dans laquelle tous les objets utilisés sont déclarées
- et une partie « Actions », entre les mots-clefs « DEBUT » et « FIN », qui décrit la suite d'actions réalisées par l'algorithme.

Exemple 1 : algorithme de calcul de l'aire d'une piste en pseudocode :

```
ALGO CalculerAirePiste
ROLE calculer l'aire d'une piste
DONNEES
   CONST PI  REEL ← 3.14159
   VAR  rExt, rInt REEL  'rayons, saisis
        aireExt, aireInt, airePiste REEL ' aires, calculées
DEBUT TRAITEMENT
   ' INITIALISATION - demander la saisie des rayons
   AFFICHER " rayon du disque extérieur ?"
   SAISIR rExt
   AFFICHER " rayon du disque intérieur ?"
   SAISIR rInt
   ' TRAITEMENT – calculer les aires
   aireExt ← (PI * rExt * rExt)
   aireInt ← (PI * rInt * rInt)
   airePiste ← aireExt – aireInt
  '  RESULTAT – afficher l'aire calculée
   AFFICHER "l'aire vaut ", airePiste
FIN TRAITEMENT
```

Données

Actions

En pseudocode, « ' » introduit un commentaire.

[9] La notation en pseudocode n'est pas normalisée ; bien qu'elle se dise « indépendante » de tout langage informatique, elle emprunte souvent sa syntaxe à une famille de langages de programmation (orientée Pascal, ou orientée C, VBA, etc.)

Notation graphique : organigramme de programmation

Les symboles graphiques de cette notation sont définis par la norme ISO 5807:1985. Elle peut être utile pour décrire des algorithmes simples.

De nombreuses notations graphiques sont utilisées dans le cadre de l'ingénierie logicielle (UML, Idefx, SADT, etc.).

Tester un algorithme

« *Tester un programme peut démontrer la présence de bogues, jamais leur absence !* »,

Edsger W. Dijkstra

Une fois l'algorithme élaboré, il est indispensable de tester manuellement son fonctionnement et de répondre à quelques questions fondamentales :

- l'algorithme se termine-t'il ?
- fournit-il le bon résultat ?
- fournit-il le même résultat lors de simulation successives à partir du même jeu d'essai ?
- fournit-il ce résultat dans un temps raisonnable ?

Le jeu d'essai

Le jeu d'essai correspond à l'ensemble des données d'entrées susceptibles d'être fournies par l'environnement (*généralement l'utilisateur au travers de saisies clavier*) et qui influencent l'exécution des actions (*l'ensemble des scénarios, des échantillons de données*). Le jeu d'essai sera appliqué à l'algorithme afin de tester son bon fonctionnement.

Il doit s'attacher à être (*le plus*) complet (*possible*) et dépasser les limites des valeurs attendues.

Remarque : Dans le temps imparti, nous n'aurons pas le temps nécessaire pour approfondir ce point. Des outils sont disponibles pour la construction de jeux d'essais lors de la phase de codage.

La trace d'exécution

La trace d'exécution d'un algorithme permet la vérification d'un scénario. Elle reprend l'ensemble des données du jeu d'essai et parcourt la séquence des actions de l'algorithme pour faire évoluer la valeur des variables du début jusqu'à la fourniture du résultat.

Dans un premier temps, il est nécessaire de numéroter les actions de l'algorithme dans l'ordre d'exécution. Puis il s'agira d'exécuter tour à tout chacune des actions.

Cas du pseudocode :

```
ALGO CalculerAirePiste
ROLE calculer l'aire d'une piste
DECLARATIONS
    CONST PI réel ← 3.14159
    VAR rExt, rInt réel '  rayons, saisis
        aireExt, aireInt, airePiste réel'  aires, calculées
DEBUT
    '  INITIALISATION - demander la saisie des rayons
(1)    AFFICHER " rayon du disque extérieur ?"
(2)    SAISIR rExt
(3)    AFFICHER " rayon du disque intérieur ?"
(4)    SAISIR rInt

    '  TRAITEMENT – calculer les aires
(5)    aireExt ← (PI * rExt * rExt)
```

```
(6)    aireInt ← (PI * rInt * rInt)
(7)    airePiste ← aireExt – aireExt
   ‘  RESULTAT – afficher l’aire calculée
(8)    AFFICHER ″l’aire vaut ″, airePiste
FIN
```

- Pas : le numéro de l’action et la trace d’exécution des différentes actions
- Entrée : les valeurs lues, c’est-à-dire dont la saisie est généralement demandée à l’utilisateur
- Condition : (*la valeur d’une condition – le fait qu’elle soit vraie ou fausse –, non applicable ici*)
- Données utilisées, constantes et variables
- Sortie : les valeurs écrites, c’est-à-dire généralement affichées à l’écran

La première ligne (DEB) marque le début de l’exécution et met en évidence les valeurs déjà connues (constantes et variables déjà initialisées).

Les lignes suivantes seront alimentées au fur et à mesure du parcours des différentes actions (= les différents pas) de l’algorithme afin de faire évoluer le contenu des données vers la solution attendue.

Dans le cas de lecture de données, les valeurs seront choisies de manière arbitraire (cf. « jeu d’essai », plus haut), ici 10 pour le rayon r1 et 5 pour le rayon r2. Ces valeurs correspondent à notre petit « jeu d’essai ».

Pas	**Entrée**	**condition**	**CONST**	**VAR**					**Sortie**
			Pi	**rExt**	**rInt**	**aireExt**	**aireInt**	**airePiste**	
DEB			3.14159						
1			‘’						rayon du disque extérieur ?
2	10		‘’	10					
3			‘’	‘’					rayon du disque intérieur ?
4	5		‘’	‘’	5				
5			‘’	‘’	‘’	314.159			
6			‘’	‘’	‘’	‘’	78.539		
7			‘’	‘’	‘’	‘’	‘’	235.62	
8			‘’	‘’	‘’	‘’	‘’	‘’	l’aire vaut 235.62
FIN									

Une vérification manuelle nous donne un résultat identique…c’est un élément minimal de vérification.

Codage

Il s’agit ensuite d’implémenter l’algorithme dans un langage de programmation. Bien que la transformation de l’algorithme en code soit relativement aisée dans les cas simples, certains langages de programmation offrent des fonctionnalités étendues qui vont simplifier l’écriture, ou au contraire réduites qui vont nécessiter un effort supplémentaire. Une bonne maitrise du langage de programmation est alors nécessaire.

1. Programmer la solution dans le langage VBA :

```
Sub calculerAirePiste()

    Const PI As Double = 3.14159
    Dim rExt, rInt As Double ' rayons : saisis
    Dim aireExt, aireInt, airePiste As Double ' aires : calculées
```

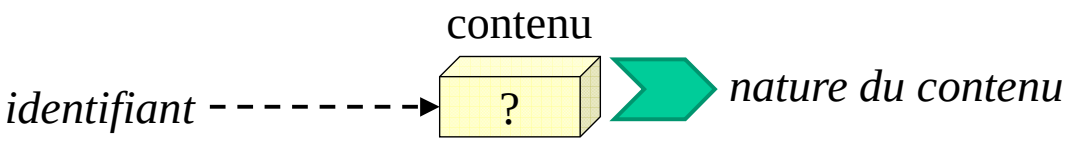
```
    ' initialisation - demander la saisie des rayons
    rExt = InputBox("rayon du disque extérieur")
    rInt = InputBox("rayon du disque intérieur")

    ' traitement - calculer les aires
    aireExt = PI * rExt * rExt
    aireInt = PI * rInt * rInt
    airePiste = aireExt - aireInt

    ' résultat - afficher l'aire calculée
    MsgBox ("l'aire vaut " + CStr(airePiste))

End Sub
```

3. Identification des objets : déclarations des données

L'identification des objets représente l'étape initiale permettant, à partir d'une situation réelle, d'en extraire les informations utiles à la résolution d'un problème. Il s'agira de créer un modèle numérique des informations sous forme de données élémentaires.

Les données représentent donc, dans l'algorithme, les caractéristiques utiles des objets utilisés dans la résolution (ou l'automatisation de la résolution) d'un problème. A chaque donnée devra correspondre :

- un identifiant unique qui donne accès une valeur
- une valeur : un contenu d'une certaine nature (nombre, texte, etc.) accessible à partir de l'identifiant



Toutes les données manipulées dans un algorithme doivent être identifiées et être déclarées. Cependant certaines d'entre elles n'offrant pas de caractéristiques utiles figureront à l'état de valeurs littérales[10] au sein de la partie « Action » des algorithmes.

La déclaration d'une donnée identifiée (*au contraire d'une valeur littérale*) comporte les éléments suivants :

1. le caractère constant ou variable du contenu de la donnée (*le contenu risque-t-'il de changer ou pas au cours du déroulement de l'algorithme ?*)
2. le type de donnée (*quelle est la nature du contenu de la donnée ?)*
3. l'identifiant, ou identificateur, de la donnée (*comment va-t-on nommer cette valeur pour y faire référence dans la suite de l'algorithme ?*)
4. une valeur initiale (*quelle est la valeur de la donnée au début du déroulement de l'algorithme ?)*
5. une information complémentaire sous forme de commentaire (*quel objet représente cette donnée et comment va-t-elle être utilisée – saisi, calculé, résultat – ?*)

Données identifiées : déclarations

Constante ou variable ?

Il s'agit de déterminer si le contenu d'une donnée est connu et ne sera pas modifié au cours de l'exécution de l'algorithme : si c'est le cas, la donnée sera déclarée comme constante (CONST), dans le cas contraire comme variable (VAR).

Types de donnée

Le type de donnée détermine la nature du contenu d'une donnée et l'étendue de sa valeur (*son domaine de définition*). Les types suivants sont généralement utilisés :

	nombres		textes		valeurs logiques	date
Type de donnée	**entier**	**réel**	**caractère**	**chaine** de caractères	**booléen**	
Exemple	-58 0 2013	-60.25 0.258 1.25E10 1.568E-25	'a' '5' '+' ' '	"nom" "vitesse du train" "" (*chaine vide)*	VRAI FAUX	

[10] Une valeur littérale est une valeur sans identifiant, donnée sous forme d'un nombre, d'une lettre, d'une chaine de caractère, etc. Par exemple : 3 est une valeur littérale numérique, « hello » est une valeur littérale textuelle

VBA	**byte integer long**	**single double currency decimal**	**string**	**boolean**	**date**
	variant				

Il faudra donc, dans la conception de la solution algorithmique, représenter les objets utiles à la résolution en utilisant ces types de données de base.

Identifiant et convention de nommage

Chaque donnée doit être identifiée afin de pouvoir faire référence à son contenu dans la partie « action » de l'algorithme. Une convention de nommage précise les règles relatives à la manière de nommer des objets, convention appliquée par l'ensemble des équipes travaillant dans une entreprise ou autour d'un projet.

Par exemple, la notation dite « hongroise », définit que les noms des données comportent un préfixe indiquant leur utilisation ou leur type de donnée, la liste des préfixes étant définie :

- les préfixes indiquant l'utilisation de la donnée (*les exemples sont ici francisés*) :
  - 'mt' pour « montant », nombre réel : mtDemande, mtFacture, etc.
  - 'qte' pour « quantité », nombre réel : qteCommandee, etc.
  - 'nb' pour « nombre de », nombre entier : nbArticles, nbMachines, etc.
  - 'ct' pour « compteur » (*pour compter …*), nombre entier : ct1, ct2, etc.
- les préfixes indiquant le type de la donnée[11] :
  - 'i' définit pour « nombre entier » : iIndex, iNbArticles, iNbMachines, iCt1, etc.
  - 'f' pour « nombre réel » : fMtDemande, fMtFacture
  - 'c' pour « caractère »: cChoix, cReponse, etc.
  - 's' pour « chaine de caractères » : sNom, sPrenom, etc.

Dans le cadre de ce module d'algorithmique, nous retiendrons une convention plus « légère ». Un identifiant :

- comportera les lettres de « a » à « z », de « A » à « Z », les chiffres de « 0 » à « 9 »
- devra commencer par une lettre
- devra être **clair** mais **concis**
- et ne devrait pas faire partie des mots réservés (*cf. Annexe 1*)

Par convention toujours : les identifiants des variables débuteront par une lettre minuscule, ceux des constantes comporteront des lettres majuscules.

On s'attachera à définir et utiliser des préfixes lorsque cela sera nécessaire à la lisibilité et à la compréhension d'un algorithme.

Déclaration des données constantes

Une **constante** est une donnée dont *la valeur est connue et n'est jamais modifiée* au cours de l'exécution d'un algorithme. C'est une donnée de référence, son contenu est fixé et invariable.

CONST identifiant type ← valeur ' description

(1) (2) (3) (4) (5) (6)

où :

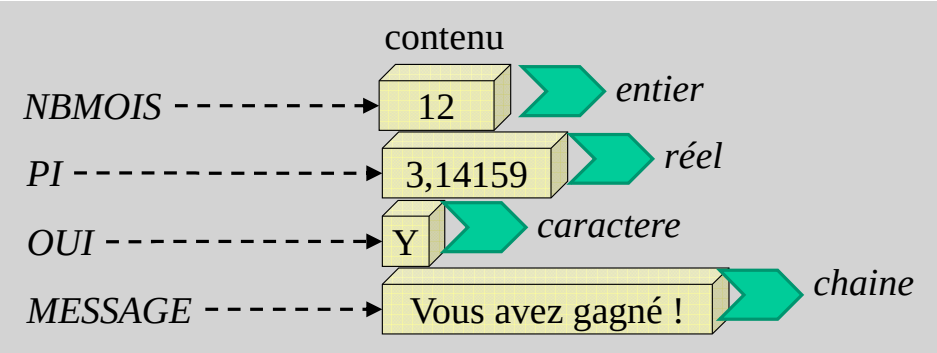
1) CONST : obligatoire, mot clef introduisant une constante

[11] Les préfixes indiquant un type de donnée sont associés aux noms anglais : i pour integer (entier), f pour float(réel), c pour character (caractère), s pour string (chaine de caractères) et b pour boolean (booléen)

2) identifiant : obligatoire, l'identifiant de la constante
3) type : obligatoire, type de la constante
4) = : obligatoire, attribution d'une valeur fixe
5) valeur : obligatoire, valeur fixe de la constante
6) ‘ description : facultatif, un commentaire précisant l'utilisation de la variable

Exemples

```
CONST NBMOIS  entier = 12 ‘  nombre de mois de l'année
CONST PI  réel = 3.14159
CONST OUI  caractère = ‘y’ ‘  valeur du choix pour « oui »/yes
CONST MESSAGE  chaine = "vous avez gagné !" ‘
```



Déclaration de variables

Une **variable** est une donnée dont *le contenu peut être modifié* au cours du déroulement d'un algorithme : son type de donnée est fixe, mais son contenu variable

VAR identifiant type ← valeur ‘ description

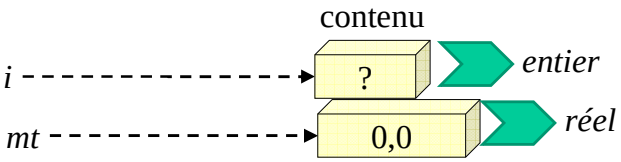
(1) (2) (3) (4) (5) (6)

où :

1) VAR : obligatoire, mot clef introduisant une variable
2) identifiant : obligatoire, l'identifiant de la variable
3) type : obligatoire, type de la variable
4) ← : facultatif, opérateur « reçoit » (cf. affectation ch. 4)
5) valeur : facultatif, valeur initiale de la variable
6) ‘ description : facultatif, un commentaire précisant l'utilisation de la variable

Exemples

```
VAR i entier ‘  indice, calculé
VAR mt réel ← 0 ‘  montant, saisi
```



Visibilité des déclarations

Les données déclarées ne sont visibles que dans l'algorithme où elles ont été déclarées, et après leur déclaration.

Si la résolution d'un problème nécessite plusieurs algorithmes, ceux-ci devront utiliser un mécanisme de passage d'informations pour s'échanger des données (cf. *arguments et paramètres des sous-programmes*).

Valeurs littérales, ou littéraux

Une valeur littérale, ou littéral, désigne une valeur (*nombre, texte, etc.*), qui ne comporte pas d'identifiant : sa signification est soit évidente soit sans importance pour la compréhension du fonctionnement de l'algorithme. Elle est directement utilisée dans le déroulement d'un algorithme.

Les valeurs littérales sont à éviter : il est souhaitable de les déclarer comme constantes (*surtout si elles sont susceptibles de subir des évolutions dans le temps, dans le cadre d'une maintenance évolutive*).

Exemple pour une valeur littérale « 0 » :

s'il s'agit de savoir si un nombre est négatif ou pas, l'utilisation du littéral « 0 » est approprié (*la déclaration d'une constante* `Zero` *initialisée à* `0` *n'aurait aucun sens*) ; cependant une constante `LimiteInferieure` valant 0 pourrait avoir un sens (= *une utilité dans la compréhension de l'algorithme*) si celle-ci pourrait être amenée à changer au cours de la vie de l'algorithme ou du programme.

Notation algorithmique

Pseudocode	VBA

`ALGORITHME identifiant de l'algorithme`
`DECLARATIONS`
`    CONST liste des constantes`
`    VAR liste des variables`
`DEBUT`
`    . . . actions . . .`
`FIN`
	`sub identifiant ()`
	`' déclarations`
	`    const liste des constantes`
	`    dim liste des variables`
	`' traitement`
	
	`end sub`

Exemple :

Pseudocode	VBA

`ALGORITHME AireDisque`
`DECLARATIONS`
`    CONST PI réel ← 3.14`
`    VAR entier r '  rayon, saisi`
`        aire '  aire réel calculé`
`DEBUT`
`    . . . actions . . .`
`FIN`
	`sub aireDisque ()`
	`' déclarations`
	`    const PI as Double = 3.14`
	`    dim r as Integer ' rayon, saisi`
	`    dim aire as double ' aire, calculé`
	`' traitement`
	
	`end sub`

4. L'affectation

L'affectation (*ou parfois assignation*) est l'action qui consiste à remplacer la valeur d'une variable par une nouvelle valeur. L'initialisation d'une variable correspond à l'affectation de sa première valeur.

La valeur affectée doit être du même type de donnée que le variable (*ou bien d'un type compatible*).

L'action d'affectation est représentée par le symbole « ← » :

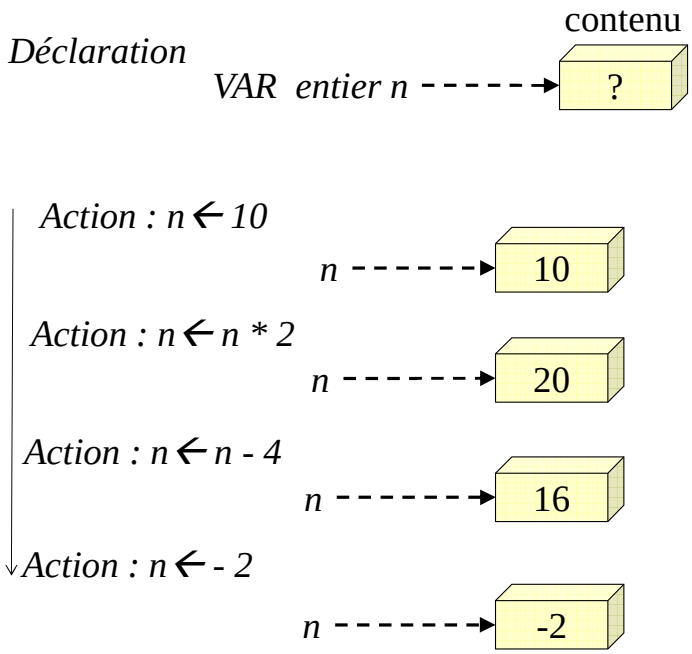
variable ← valeur

*Soit « variable **reçoit** valeur ».*

- À droite (*rvalue ou rigth value*), la valeur de l'expression évaluée (*elle doit être du même type de donnée que la variable affectée*) ; « valeur » peut être un littéral, une variable ou une expression
- À gauche (*lvalue ou left value*) : l'identifiant de la variable qui reçoit la valeur

Après affectation, la variable contient la nouvelle valeur (*l'ancienne a été écrasée, elle est perdue à jamais!).*

Exemple d'évolution de la valeur de la variable n au fur et à mesure des affectations :



La valeur peut prendre la forme :

- d'une valeur littérale
- ou d'un identifiant de constante ou de variables,
- ou d'une expression de calcul
- ou d'un appel de fonction (*assimilable à la valeur retournée par cette fonction*),

Notation algorithmique

Pseudocode	VBA
ALGORITHME identifiant de l'algorithme DECLARATIONS . . . DEBUT	**sub** identifiant () ' déclarations **const** liste des constantes **dim** liste des variables ' traitement

. . . actions . . . variable1 ← expression1 . . . actions . . . FIN	variable1 = expression1 **end sub**

Exemple :

Pseudocode	Arbre programmatique
ALGORITHME AireDisque DECLARATIONS CONST Pi reel ← 3.14 VAR r entier ‘ rayon, saisi aire réel‘ aire, calculé DEBUT aire ← 0 r ← 10 aire ← (Pi * r * r) FIN	**sub** aireDisque () ‘ déclarations **const** PI as Double = 3.14 **dim** r as Integer ‘ rayon, saisi **dim** aire as double ‘ aire, calculé ‘ traitement aire = 0 r = 10 aire = PI * r * r **end sub**

5. Expressions de calculs numériques et logiques

Une expression désigne le calcul d'une valeur à partir d'opérandes (*littéraux, constantes et variables et autre expressions)* et d'opérateurs.

Quand des variables participent au calcul d'une expression, leur valeur n'est pas modifiée[12].

L'évaluation d'une expression fournit un résultat d'un certain type de donnée déterminé par les opérateurs utilisés. Si ce résultat n'est pas affecté à une variable, sa valeur est perdue.

Expressions numériques

Opérateurs binaires

Une expression numérique est une expression utilisant des nombres et des opérateurs arithmétiques et dont l'évaluation fournit un nombre *(le résultat du calcul est un nombre)*. L'usage des parenthèses n'est utile que si l'expression de calcul est ambiguë *(cf. priorité des opérateurs)*.

valeur1 opérateurArithmétique valeur2

où valeur1 et valeur2 sont des nombres.

Opérateur arithmétique	Signification	Exemple avec des valeurs littérales	Résultat de l'évaluation de l'expression
+	Addition	5 + 2	7
-	Soustraction	2 – 5	-3
*****	Multiplication	5 * 2	10
/	Division	5 / 2	2.5

La priorité des opérateurs est la suivante :

Priorité	**Opérateurs binaires**
forte	*, /, %
faible	+, -

L'évaluation est réalisée de gauche à droite en cas de même priorité. L'utilisation des parenthèses lève tout risque d'ambiguïté et clarifie généralement les expressions numériques.

- Exemples : pour a et b, entiers, valant respectivement 2 et 5
  - `a + b` : vaut 7
  - `2 * a + b` : vaut (2*a)+b soit 9
  - `2 * (a + b)` : vaut 14

Opérateur unaire

Les opérateurs unaires n'est applicable qu'aux valeurs littérales et permet de préciser le signe d'un nombre :

[12] Attention : lors de la phase de programmation, on peut rencontrer certains langages comportant des opérateurs susceptibles de modifier le contenu des variables présentes dans une expression (exemple : ++, --). Cette pratique n'est cependant pas recommandée

signe litteral1

où valeur est une valeur littérale et signe parmi « + », « - »

- Exemples : -2, +2000, -0.56, +18250.25 sont des valeurs littérales signées

Pour changer le signe d'une variable ou d'une expression, il faut utiliser la multiplication :

- Exemples : pour a et b, entiers, valant respectivement 2 et 5
  - `-1 *(a + b)` : vaut -7

Expressions logiques ou booléennes

Une expression logique ou booléenne est une expression utilisant des opérateurs de comparaison (*dits « opérateurs relationnels*[13] ») et des objets de mêmes types pour les comparer, et des connecteurs logiques permettant d'associer plusieurs expressions logiques élémentaires.

L'évaluation d'une expression logique fournit un résultat booléen (*valeurs littérales VRAI ou FAUX*).

Opérateurs relationnels ou opérateurs de comparaison

valeur1 opérateurRelationnel valeur2

où valeur1 et valeur2 sont des données de même type (*valeurs littérales, variables ou expressions*)

	Opérateurs relationnels	Signification	Exemple	Résultat de l'évaluation
égalité	**=**	égal	2 = 3	**F**AUX
inégalité	**<>**	différent	2 <> 3	**V**RAI
	<	Inférieur	2 < 3	**V**RAI
	<=	Inférieur ou égal	2 <= 3	**V**RAI
	>	Supérieur	2 > 3	**F**AUX
	>=	Supérieur ou égal	2 >= 3	**F**AUX

- Exemples :
  - `age >= 18` : VRAI si `age` est supérieur ou égal à 18, FAUX sinon
  - `1 = 1` : toujours VRAI
  - `1 = 2` : toujours FAUX
  - `( age + 2) < (3 * 6)` : VRAI si `age` est inférieur à 16

Connecteurs logiques

Les connecteurs logiques permettent de combiner des expressions logiques élémentaires pour former de nouvelles expressions logiques plus complexes.

expressionLogique1 opérateurLogique expressionLogique2

pour les connecteurs binaires ET et OU

[13] ou opérateurs de comparaison, opérateurs qui mettent en relation entre 2 valeurs pour les comparer (égalité ou inégalité)

NON expressionLogique

pour l'opérateur unaire NON

Table de vérité :

Expression logique A	Expression logique B	**A ET B** *conjonction* *(A . B)* $A \wedge B$	**A OU B** *disjonction* *(A + B)* $A \vee B$	**NON A** *négation* *(* $\bar{a}$ **)** $\neg A$
FAUX	FAUX	FAUX	FAUX	VRAI
FAUX	VRAI	FAUX	VRAI	VRAI
VRAI	FAUX	FAUX	VRAI	FAUX
VRAI	VRAI	VRAI	VRAI	FAUX

Figure 1 : Table de vérité

- Exemples :
  - `(age >= 18) ET (note > 10)` : VRAI si les 2 expressions sont vraies - FAUX sinon
  - `(age >= 18) OU (note > 10)` : VRAI si au moins des deux expressions logique est VRAI - FAUX sinon

Expressions alphanumériques

Opérateurs binaires

Une expression alphanumérique utilise des chaines de caractères et l'opérateur de concaténation et fournit un résultat de type chaine de caractères. La concaténation de 2 chaines combine les valeurs des 2 chaines en mettant la seconde au bout de la première.

valeur1 + valeur2

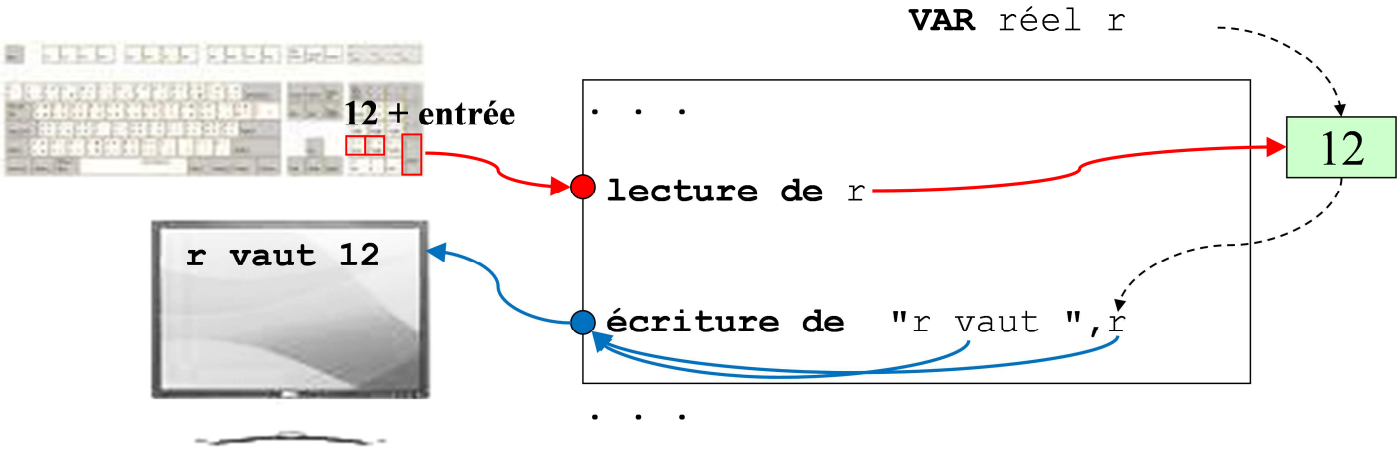
où valeur1 et valeur2 sont des valeurs alphanumériques.

Opérateur alphanumérique	Signification	Exemple avec des valeurs littérales	Résultat de l'évaluation de l'expression
+	concaténation	"colo"+"rado"	"colorado"

6. Echanges avec l'utilisateur : entrées/sorties

Les échanges des données avec l'environnement immédiat, généralement l'utilisateur, permettent la construction d'algorithmes répondant à une grande variété de questions, sans à avoir à élaborer un nouvel algorithme. Ils sont généralement associés aux périphériques écran et clavier :

- envoi/écriture de données vers l'écran,
- ou bien récupération/lecture de données du clavier pour les affecter à des variables.



Sortie, écriture, affichage, restitution

L'action d' « écriture » commande l'envoi de données vers l'extérieur, généralement l'écran : les données peuvent être des valeurs littérales, des contenus de variables ou des valeurs d'expressions.

Le contenu des variables sources n'est pas modifié.

Dans l'exemple ci-dessus, le contenu de r est affiché, mais n'est pas modifiée.

Entrées, lecture, saisie, acquisition

L'action de « lecture » commande la récupération de valeurs provenant de l'extérieur, généralement le clavier, pour les affecter à des variables.

Le contenu des variables cibles est affecté.

Dans l'exemple ci-dessus, le contenu de r est modifié par la valeur saisie.

Notation algorithmique

Pseudocode	VBA
AFFICHER valeur1 [, valeur2, …, valeurN] ou **AFFICHER** valeur1 [, valeur2, …, valeurN]	**msgBox**(message)
SAISIR variable1 [, variable2, …, variableN] ou **SAISIR** variable1 [, variable2, …, variableN]	variable1 = **inputBox**(message)

Exemple :

Pseudocode	VBA

ALGORITHME aireDisque
DECLARATIONS
  CONST PI reel = 3.14
  VAR r entier ‘ rayon, saisi
  aire réel ‘ aire, calculé
DEBUT
  ‘ Initialisation
  AFFICHER "rayon ?"
  SAISIR r
  ‘ Traitement
  aire ← (PI * r * r)
  ‘ Résultat
  AFFICHER "l'aire vaut ", aire
FIN
	sub aireDisque ()
	‘ déclarations
	  const PI as Double = 3.14
	  dim r as Integer ‘ rayon, saisi
	  dim aire as double ‘ aire, calculé
	‘ traitement
	  ‘ initialisation
	  r = **inputBox**("rayon ?")
	  ‘ traitement
	  aire = PI * r * r
	  ‘ résultat
	  msgbox("l’aire vaut "+cstr(aire))
	end sub
	
	La fonction VBA `cstr` permet de convertir une valeur en chaine de caractères

7. Structure de contrôle conditionnelle : tests, décision

Les structures de contrôle permettent la création d'une rupture dans l'exécution en séquence des actions en proposant

- un choix dans l'exécution d'une séquence d'actions : *structures de contrôles conditionnelles*,
- ou la répétition de l'exécution d'une séquence d'actions : *structures de contrôles répétitives*.

Ces structures sont basées sur l'évaluation de conditions définies sous forme d'expression logiques.

Une fois la structure de contrôle traitée, l'exécution reprend à l'instruction qui suit.

Structure de contrôle conditionnelle

La structure de contrôle conditionnelle permet de choisir d'exécuter ou pas une séquence d'actions selon qu'une condition est vraie ou pas.

Par exemple :

- **si** tu es à l'heure ce soir **alors** nous irons au cinéma.
  - la condition, exprimée sous forme d'une expression logique, est : « tu es à l'heure ce soir »
  - si cette condition est vraie, alors « nous irons au cinéma »
  - dans le cas contraire, on ne fait rien
- **si** ta moyenne est supérieure à 20, **alors** tu auras ta photo dans le journal local
  - la condition, exprimée sous forme d'une expression logique, est : « ta moyenne est supérieure à 20 »
  - si cette condition est vraie, alors « tu auras ta photo dans le journal local »
  - dans le cas contraire, rien ne se passe

Structure de contrôle conditionnelle avec alternative

Cette structure permet également l'exécution d'une séquence d'action alternative au cas où la condition est fausse :

Par exemple :

- **si** tu es à l'heure ce soir **alors** nous irons au cinéma **sinon** nous regarderons un film sur le petit écran
  - la condition, exprimée sous forme d'une expression logique, est : « tu es à l'heure ce soir »
  - si cette condition est vraie, alors « nous irons au cinéma »
  - dans le cas contraire, « nous regarderons un film sur le petit écran »
- si mes économies me le permettent **alors** nous partirons sur la côte d'Azur **sinon** nous profiterons de la belle plage de Calais
  - la condition, exprimée sous forme d'une expression logique, est : « mes économies me le permettent »
  - si cette condition est vraie, alors « nous partirons à Nice »
  - dans le cas contraire, « nous profiterons de la belle plage de Calais »

Notation algorithmique

Pseudocode	Arbre programmatique

ALGORITHME Algorithme
DECLARATIONS
. . .
DEBUT
SI condition
ALORS
actions si vrai
SINON
actions si faux
fin SI
FIN
	sub programme()
	‘ déclarations
	
	‘ traitement
	if (condition) **then**
	actions si vrai
	else
	actions si faux
	end if
	end sub

Exemple : (il faut au moins 1000 euros d'économies pour partir sur la côte d'azur)

Pseudocode	VBA

ALGORITHME ChoisirVacances
DECLARATIONS
CONST MINIECO réel ← 1000
VILLEA chaine ← "Nice"
VILLEB chaine ← "Calais"
VAR mesEco réel
destination chaine
DEBUT
' INITIALISATION
AFFICHER "montant éco. ?"
SAISIR mesEco
desination ← " ?"
' TRAITEMENT
SI mesEco >= MINIECO
ALORS
destination ← VILLEA
SINON
destination ← VILLEB
fSI
' RESULTAT
AFFICHER "on part à ", destination," !"
FIN
	Sub ChoisirVacances()
	' déclarations
	Const MINIECO As Currency = 1000
	Const VILLEA As String = "Nice"
	Const VILLEB As String = "Calais"
	Dim mesEco As Currency
	Dim destination As String
	' traitement
	' INITIALISATION
	mesEco = InputBox("montant éco. ?")
	destination = "?"
	' TRAITEMENT
	If (mesEco >= MINIECO) Then
	destination = VILLEA
	Else
	destination = VILLEB
	End If
	' RESULTAT
	MsgBox ("on part à " + destination + " !")
	End Sub

ALGORITHME ChoisirVacancesVersion2
DECLARATIONS
CONST MINIECO réel ← 1000
VILLEA chaine ← "Nice"
VILLEB chaine ← "Calais"
VAR mesEco réel
destination chaine
DEBUT
' INITIALISATION
AFFICHER "montant éco. ?"
SAISIR mesEco
destination ← VILLEB
' TRAITEMENT
SI mesEco >= MINIECO²
ALORS
destination ← VILLEA
fSI
' RESULTAT
AFFICHER "on part à", destination," !"
FIN
	Sub ChoisirVacancesVersion2()
	' déclarations
	Const miniEco As Currency = 1000
	Const villeA As String = "Nice"
	Const villeB As String = "Calais"
	Dim mesEco As Currency
	Dim destination As String
	' traitement
	' INITIALISATION
	mesEco = InputBox("montant éco. ?")
	destination = villeB
	' TRAITEMENT
	If (mesEco >= miniEco) Then
	destination = villeA
	End If
	' RESULTAT
	MsgBox ("on part à " + destination + " !")
	End Sub

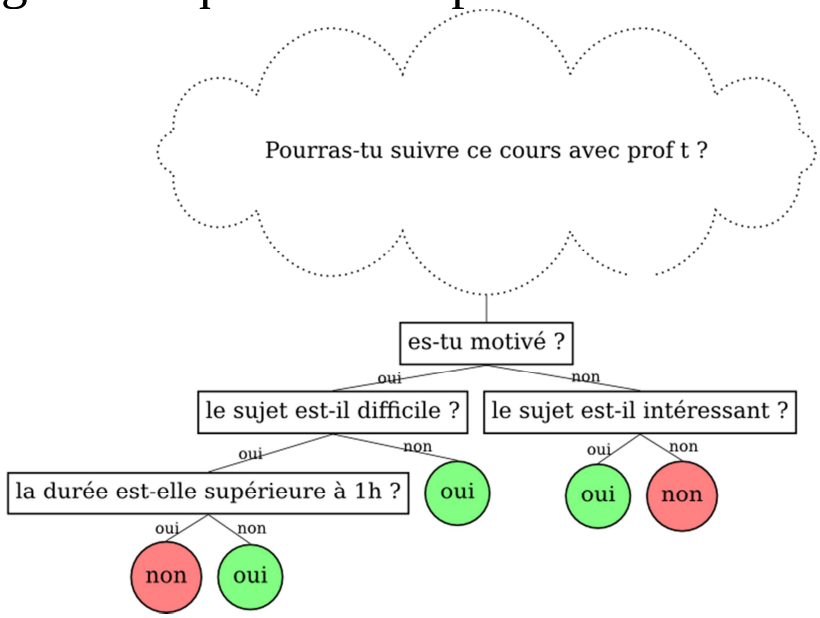
Cette 2ème version offre une simplification d'écriture.

Arbres de décisions et imbrication

Dans le cas de traitement de décisions[14] plus complexes, des structures conditionnelles peuvent être imbriquées.

Par exemple : comment prévoir si oui ou non tu pourras suivre ce cours (où n'importe quel autre…) avec profit ?

[14] Un arbre de décision représente une succession de tests à l'issue desquels une conclusion est obtenue



Une représentation de l'algorithme permet la mise en œuvre du questionnement correspondant et va nécessiter une imbrication de structures conditionnelles :

Pseudocode	VBA

ALGORITHME suivreUnCours
DECLARATIONS
  CONST OUI chaine = « oui »
  CONST NON chaine = « non »
  VAR chaine reponse
  VAR chaine suivre
DEBUT
  suire = « ? »
  AFFICHER "motivé ?"
  SAISIR reponse
  SI reponse = « oui »
  ALORS
  AFFICHER « sujet difficile ? »
  SAISIR REPONSE
  SI reponse = « oui »
  ALORS
  AFFICHER « duree > 1h ? »
  SAISIR reponse
  SI reponse = « oui »
  ALORS
  suivre ← « non »
  SINON
  suivre ← « oui »
  FinSI
  SINON
  suivre ← « oui »
  FinSI
  SINON
  AFFICHER « interessant ? »
  SAISIR reponse
  SI reponse = « oui »
  ALORS
  suivre ← « oui »
  SINON
  suivre ← « non »
	Sub SuivreUnCours()
	
	' déclarations
	  Const OUI As String = "oui"
	  Const NON As String = "non"
	  Dim reponse As String
	  Dim suivre As String
	' traitement
	  suivre = "?"
	  reponse = InputBox("motivé ?")
	  If reponse = OUI Then
	  reponse = InputBox("sujet difficile ?")
	  If reponse = OUI Then
	  reponse = InputBox("duree > 1h ?")
	  If reponse = OUI Then
	  suivre = NON
	  Else
	  suivre = OUI
	  End If
	  Else
	  suivre = OUI
	  End If
	  Else
	  reponse = InputBox("interessant ?")
	  If reponse = OUI Then
	  suivre = OUI
	  Else
	  suivre = NON
	  End If
	  End If
	  ' RESULTAT
	  MsgBox ("le résultat est " + suivre + " !")
	
	End Sub

```
        FinSI
    FinSI
    ‘  RESULTAT
    AFFICHER  "le  résultat  est  ",
suivre," !"
FIN
```

8. Structure de choix multiples

La structure de choix est une extension de la structure conditionnelle : elle permet de traiter différents cas de valeurs d'une variable, c'est-à-dire d'effectuer des traitements particuliers selon les différentes valeurs de cette variable.

Notation algorithmique

Pseudocode

```
ALGO testerCasMoyenne
DECLARATION
    VAR moyenne réel
    VAR resultat chaine
DEBUT
‘ INITIALISATION
    AFFICHER("entrer la moyenne:")
    SAISIR moyenne
    resultat ← "?"
‘ TRAITEMENT
    SELON moyenne
        CAS < 10:
            resultat ← "revoir les
notions"
        CAS 10:
            resultat ← "tout
juste..."
        CAS < 15:
            resultat ← "encore
quelques efforts"
        CAS <= 20 :
            resultat ← "c'est
bon...mais continuer"
        CAS SINON:
            resultat ← " à voir..."
    FIN SELON
' RESULTAT
    AFFICHER (resultat)

FIN
```

Arbre programmatique

```
Sub testerCasMoyenne()
' DECLARATION
    Dim moyenne As Double
    Dim resultat As String
' INITIALISATION
    moyenne = InputBox("entrer la moyenne:")
    resultat = "?"
' TRAITEMENT
    Select Case moyenne
        Case Is < 10:
            resultat = "revoir les notions"
        Case 10:
            resultat = "tout juste..."
        Case Is < 15:
            resultat = "encore quelques
efforts"
        Case Is <= 20:
            resultat = "c'est bon...mais
continuer"
        Case Else:
            resultat = " à voir..."
    End Select
' RESULTAT
    MsgBox (resultat)

End Sub
```

9. Structure de contrôle itérative : boucles

Les structures itératives ou répétitives (*ou boucles*) permettent l'exécution répétée d'une séquence d'actions (*appelé corps de la boucle*).

Le nombre de répétitions est toujours contrôlé soit par une condition de poursuite de la répétition ou en utilisant un compteur. la condition d'arrêt (ou de poursuite) d'une boucle doit être parfaitement identifiée[15].

Par exemple : on souhaite remplir une baignoire de 200 litres avec un seau d'une contenance de 15 litres (*attention, l'eau est une ressource très précieuse ; une douche consomme 5 fois moins d'eau qu'un bain !)*

Deux stratégies s'offrent à nous :

1. Soit calculer le nombre de seaux à verser, 200 / 15, soit 13 seaux, puis compter de 1 à 13 et verser un seau à chaque fois
2. Soit verser des seaux tant que la baignoire n'est pas pleine, autrement dit : tant que la baignoire n'est pas pleine, verser un seau (*ou bien encore, autre solution : tant qu'on n'a pas ajouté 13 seaux, on ajoute un seau*)

Structure de contrôle TANT QUE

La boucle TANT QUE permet d'exécuter une séquence d'actions si une condition est vraie et tant qu'elle est vraie. Le corps de la boucle s'exécutera 0 à N fois.

La boucle TANT QUE s'applique bien à la 2eme stratégie

- **tant que** la baignoire n'est pas pleine verser un seau.
  - la condition, exprimée sous forme d'une expression logique, est : « la baignoire n'est pas pleine »
  - si cette condition est vraie, et tant qu'elle est vraie, on va répéter « verser un seau »
  - dans le cas contraire, on ne fait rien

ATTENTION : Les données permettant d'évaluer l'expression logique doivent avoir été initialisés auparavant ET devront être à nouveau modifiées dans le corps de la boucle (sinon on obtient une boucle infinie !)

Notation algorithmique

Pseudocode	Arbre programmatique
ALGORITHME identifiant de l'algorithme DECLARATIONS . . . DEBUT . . . actions . . . **TANT QUE** condition actions à répéter **fin TANT QUE** . . . actions . . . FIN	Sub remplirBaignoire() ' DECLARATIONS ' DEBUT **While** condition instructions à répéter **Wend** End Sub

Exemple : algorithme de remplissage de la baignoire (on considère la baignoire vide au départ)

Remarque : on risque d'avoir un petit problème…saurez-vous le découvrir ?

Pseudocode	VBA
ALGORITHME RemplirBaignoire	Sub remplirBaignoire()

[15] en anglais : loop guard

```
DECLARATIONS
    CONST VOLMAX entier ← 200
          VOLSEAU entier ← 15
    VAR volumeBain entier
DEBUT
  ‘  Initialisation
  volumeBain ← 0
  ‘  Traitement
  TANT QUE volumeBain < VOLMAX
      volumeBain ← volumeBain + VOLSEAU
  fin TANT QUE
  ‘  Résultat
  AFFICHER       "prêt       pour       le
bain ! ",volumeBain
FIN
```

```
' DECLARATIONS
    Const VOLMAX As Integer = 200
    Const VOLSEAU As Integer = 15
    Dim volumeBain As Integer
' DEBUT
  ' Initialisation
  volumeBain = 0
  ' Traitement
  While volumeBain < VOLMAX
      volumeBain = volumeBain + VOLSEAU
  Wend
  ' Résultat
  MsgBox ("prêt pour le bain ! " +
CStr(volumeBain))

End Sub
```

…attention, la baignoire va déborder…l’algorithme n’est pas correct…

Pseudocode	**VBA**

`ALGORITHME RemplirBaignoire`
`DECLARATIONS`
`    CONST VOLMAX entier ← 200`
`          VOLSEAU entier ← 15`
`    VAR volumeBain entier`
`DEBUT`
`  ‘  Initialisation`
`  volumeBain ← 0`
`  ‘  Traitement`
`  TANT QUE (volumeBain + VOLSEAU) < VOLMAX`
`      volumeBain ← volumeBain + VOLSEAU`
`  fin TANT QUE`
`  ‘  Résultat`
`  AFFICHER       "prêt       pour       le bain ! ",volumeBain`
`FIN`
	`Sub remplirBaignoire()`
	`' DECLARATIONS`
	`    Const VOLMAX As Integer = 200`
	`    Const VOLSEAU As Integer = 15`
	`    Dim volumeBain As Integer`
	`' DEBUT`
	`  ' Initialisation`
	`  volumeBain = 0`
	`  ' Traitement`
	`  While (volumeBain + VOLSEAU) < VOLMAX`
	`      volumeBain = volumeBain + VOLSEAU`
	`  Wend`
	`  ' Résultat`
	`  MsgBox ("prêt pour le bain ! " + CStr(volumeBain))`
	`End Sub`

Structure de contrôle POUR

La boucle POUR permet de répéter l’exécution d’une séquence d’actions un nombre déterminé de fois.

Pour cela une variable est utilisé en tant que compteur (on l’appelle « variable de boucle »):

- elle est initialisée grâce à une valeur de début
- tant que le « compteur est inférieur ou égal au nombre de répétition », la séquence d’action du corps de la boucle est exécutée
    - dans le cas d’une valeur de pas négative, la condition de poursuite de la boucle est « compteur est supérieur ou égal à nombre de répétitions »)
- à la fin de chaque itération, la variable de boucle est incrémentée de la valeur du pas

Le nombre d’itérations est connu.

La boucle POUR s’applique bien à la 1ère stratégie de remplissage de la baignoire (il faudra compter de 1 à 13 et verser ainsi 13 seaux) :

- **pour** mon compteur commençant à 1, et tant qu’il est inférieur ou égal à 13 (=la baignoire n’est pas pleine) **répéter** verser un seau puis ajouter 1 au compteur
    - la condition, exprimée sous forme d’une expression logique, est : « le nombre de seaux versés est inférieur à13»

- si cette condition est vraie, et tant qu'elle est vraie, on va répéter « verser un seau », puis ajouter 1 au compteur pour tester à nouveau si on verse de nième seau.

Notation algorithmique

Pseudocode	Arbre programmatique
ALGORITHME identifiant de l'algorithme DECLARATIONS DEBUT . . . actions . . . POUR V DE D a F [PAS DE P] actions à répéter fin POUR FIN	Sub remplirBaignoire3() ' DECLARATIONS ' DEBUT **For** V = D To F Step P instructions à répéter **Next** compteur End Sub

Exemple :

Pseudocode	Arbre programmatique
ALGORITHME RemplirBaignoire DECLARATIONS CONST VOLMAX entier ← 200 VOLSEAU entier ← 15 VAR volumeBain entier nbSeaux entier compteur entier DEBUT ' Initialisation volumeBain ← 0 nbSeaux ← VOLMAX / VOLSEAU AFFICHER "il faudra ",nbSeaux," seaux" ' Traitement **POUR** compteur DE 1 a nbSeaux PAS DE 1 vol ← vol + volSeau **fin POUR** ' Résultat AFFICHER "prêt pour le bain !", volumeBain FIN	Sub remplirBaignoire3() ' DECLARATIONS Const VOLMAX As Integer = 200 Const VOLSEAU As Integer = 15 Dim volumeBain As Integer Dim nbSeaux As Integer Dim compteur As Integer ' DEBUT ' Initialisation volumeBain = 0 nbSeaux = VOLMAX / VOLSEAU MsgBox ("il faudra " + CStr(nbSeaux) + " seaux") ' Traitement **For** compteur = 1 To nbSeaux Step 1 volumeBain = volumeBain + VOLSEAU **Next** compteur ' Résultat MsgBox ("prêt pour le bain ! " + CStr(volumeBain)) End Sub

Comparaison des boucles POUR et TANT QUE

La boucle TANTQUE est utilisée quand on ne connait pas le nombre d'itérations à l'avance.

La boucle POUR quand le nombre d'itérations est connu à l'avance.

Vérification des boucles : éléments de preuve de terminaison et d'exactitude

Les boucles sont une des causes d'erreur les plus fréquentes ; l'erreur la plus fréquente est une mauvaise gestion des variables de boucles.

Il faut donc vérifier que les valeurs des variables entrant dans les conditions de poursuite des boucles :

- avant d'entrer dans la boucle : valeur initiale
- et dans le corps de la boucle : modification de la condition de poursuite

10. Modularité et sous-programmes : fonctions et procédures

Un sous-programme est un moyen de nommer une action complexe, généralement composée de plusieurs actions ou calculs élémentaires,

- soit parce qu'on est amené à l'utiliser plusieurs fois dans un ou plusieurs autres algorithmes (*réutilisation, factorisation*),
- soit de manière à rendre la structure d'un algorithme plus claire et lisible (*issue de la décomposition d'un algorithme complexe en sous-algorithmes*)

Un sous-programme est donc une forme particulière d'algorithme : on ne l'exécutera jamais directement, il sera toujours « appelé » par un autre algorithme.

On parle de modularité pour décrire la qualité d'un algorithme à être divise en plusieurs sous-programmes (*ou modules*), de manière à permettre le test individuel de chacun des sous-programmes puis à réaliser un test global après intégration des sous-programmes dans l'algorithme principal.

Formes des sous-programmes

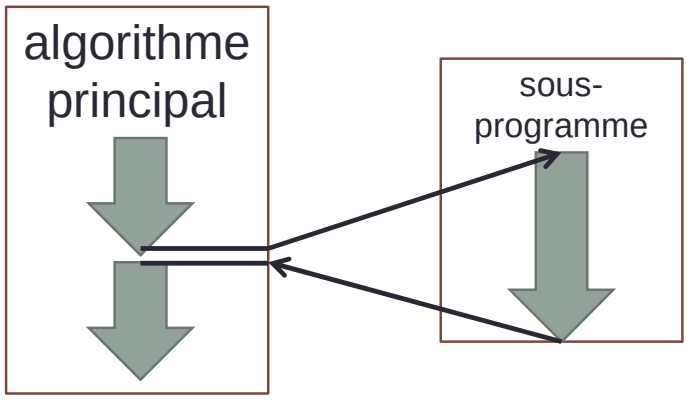
On distingue 2 formes de sous-programmes :

- la fonction, où sous-programme de type « expression », renvoie une valeur à la fin de son exécution
- la procédure, ou sous-programme de type « action », exécute un certain nombre d'actions

Exécution

Lorsque, dans l'exécution d'un algorithme, un sous-programme (*ici une fonction*) est appelé,

1. l'algorithme passe le contrôle de l'exécution au sous-programme,
2. celui-ci s'exécute jusqu'à la fin, ou à la rencontre d'une l'instruction RETOURNER,
3. puis le contrôle est à nouveau passé à l'algorithme initial, à l'instruction qui suit l'appel du sous-programme



Paramètres d'un sous-programme

Pour fonctionner, le sous-programme a généralement besoin d'échanger des données avec l'algorithme qui l'a appelé.

- Les paramètres (*paramètres formels*) d'une procédure (*ou fonction*) définissent les données nécessaires à son fonctionnement (*les valeurs attendues pour son fonctionnement*).
- Les arguments (*paramètres réels ou paramètres effectifs*) correspondent aux valeurs qui lui sont effectivement passées au moment de son appel.

Plusieurs manières de passer les paramètres vers un sous-programme (cf plus bas)

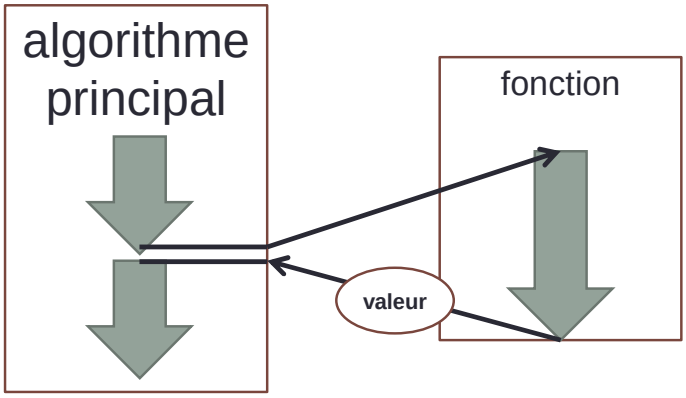
Appel d'un sous-programme

L'appel d'un sous-programme est réalisé en donnant son nom suivi de la liste des arguments entre parenthèses (*cette liste peut être vide : on ne gardera alors que le couple de parenthèses*).

Fonctions

Une fonction est un sous-programme qui renvoie une valeur de retour d'un certain type de donnée au terme de son exécution. On qualifie la fonction de sous-programme de type « expression » : son exécution fournit en effet un résultat d'un certain type utilisable comme toute autre élément dans une expression.

Dans une fonction, l'action '**RETOURNER**' commande le renvoi d'une donnée résultat à l'algorithme appelant et quitte immédiatement la fonction ; l'exécution de l'algorithme appelant reprend après le point d'appel de la fonction.



Par exemple, la fonction mathématique f(x) = 2x peut être décrite comme une fonction qui :

- attend un paramètre nommé x
- calcule la valeur (2 * x) (*la fonction calcule le double de la valeur passée en argument*)
- et retourne la valeur ainsi calculée

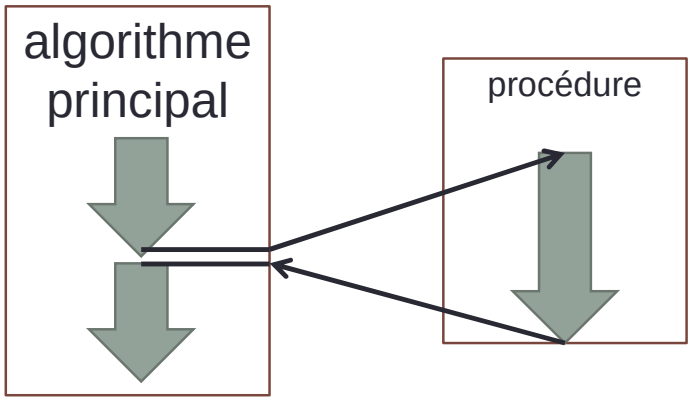
Dans l'idéal, une fonction exécute une séquence d'actions sans interruption : les opérations de lecture et d'écriture ne devraient pas être incluses dans une fonction.

Remarque : différence entre fonctions mathématiques et fonctions informatiques

- Les mathématiques décrivent les fonctions de manière formelle.
- En informatique, une fonction est définie de manière opérationnelle, procédurale : une fonction f:X→Y est une **méthode de calcul qui décrit comment obtenir une valeur y** ∈ Y à partir de x ∈ X

Procédures

Une procédure est un sous-programme qui exécute un certain nombre d'actions sans fournir de valeur de retour après son exécution.



Une procédure est un sous-programme de type « instruction » ou « action » (*non utilisable dans une expression*).

Par exemple, l'affichage d'une suite d'options proposées à l'utilisateur peut être incluse dans une procédure.

Notation algorithmique :

Fonction

```
FONCTION  identifiant (paramètres) type
  corps de la fonction
```

où :

- identifiant est le nom donné à la fonction
- paramètres : liste des valeurs attendues par la fonction
- type est le type de donnée retourné par la fonction
- corps de la fonction : déclarations et actions réalisées par la fonction, et retour d'une valeur

Pseudocode	VBA

FONCTION identifiant (paramètres) **type**
DECLARATIONS
  VAR type valeur ‘ resultat
  . . . autres . . .
DEBUT
 	
  ‘ Résultat
  RETOURNER valeur
FIN
	Function identifiant (parametres) AS **type**
	‘ DECLARATIONS
	  Dim valeur as type
	‘ TRAITEMENT
	
	‘ RESULTAT
	  identifiant = valeur
	end function

Où :

- **type**: type de données de la fonction, c’est-à-dire le type de la valeur retournée
- **identifiant** : correspond à l’identifiant de la fonction
- **paramètres** : liste des données attendues par la fonction, sous forme de déclaration :
  - **type de la donnée**
  - **identifiant de la donnée**
  - **mode de passage de la valeur** (par défaut « entree »)
- **déclarations** : éventuellement, déclaration de données locales à la fonction
- **valeur** : valeur qui sera effectivement renvoyée ; elle sera du même type que la fonction

Exemple :

Pseudocode	VBA

FONCTION doublerValeur (reel x) réel
DECLARATIONS
  VAR calcul réel ‘ calcul
DEBUT
  ‘ Traitement
  calcul ← 2 * x
  ‘ Résultat
  RETOURNER calcul
FIN
	Function doublerValeur(x As Double) As **Double**
	' DECLARATIONS
	  Dim calcul As Double
	' TRAITEMENT
	  calcul = 2 * x
	' RESULTAT
	  doublerValeur = calcul
	End Function

Exemple : appel de la fonction doublerValeur dans un autre algorithme

Pseudocode	Arbre programmatique

ALGO testerDoublerValeur
DECLARATIONS
  VAR nombre réel
  doubleNombre réel
DEBUT
  ‘ initialisation
  AFFICHER "nombre réel ?"
  SAISIR nombre
  ‘ Traitement
  doubleNombre ← **doublerValeur(calcul)**
  ‘ Résultat
  AFFICHER "le double de ", nombre, " est ", doubleNombre
FIN
	Sub testerDoublerValeur()
	' DECLARATIONS
	  Dim nombre As Double
	  Dim doubleNombre As Double
	' INITIALISATION
	  nombre = InputBox("entrer un nombre")
	' TRAITEMENT/RESULTAT
	  doubleNombre = **doublerValeur**(nombre)
	' RESULTAT
	  MsgBox ("le double de " + CStr(nombre) + " est " + CStr(doubleNombre))
	End Sub

Procédures

Une procédure est similaire à une fonction.

Elle ne retourne pas de valeur mais sert souvent à afficher des résultat selon une certain présentation

Pseudocode	Arbre programmatique
PROCEDURE identifiant (paramètres) déclarations DEBUT . . . actions . . . FIN	sub identifiant (paramètres) ‘ DECLARATIONS ‘ TRAITEMENT . . . instructions . . . end sub