Algorithmique

Instructions et types élémentaires

Présentation

- Enseignant :

  M. Abdessamad IMINE

  Bureau : 124

  email : **Abdessamad.Imine@univ-lorraine.fr**

- Progression du module :

  Cours et TD

  Lié au module **Programmation 1**

  Un soutien est prévu pour les étudiants

- Examens :

  Deux partiels (fin septembre et octobre)

  Note finale (moyenne des partiels)

Introduction

- L'informatique met en jeu des ordinateurs qui fonctionnent selon des schémas préétablis
- Pour résoudre un problème donné à l'aide d'un ordinateur, il faut lui indiquer la *suite d'actions (ou instructions)* à exécuter dans son schéma de fonctionnement
- Cette suite d'actions est un *programme* qui est exprimé dans un langage de programmation plus ou moins évolué (code machine, assembleur, C, Java, Caml, Prolog…)
- Pour écrire un programme (la suite d'actions), il faut donc d'abord savoir *comment faire* pour résoudre le problème

- Modèle Von Neuman

John Von Neumann est à l'origine (1946) d'un modèle de machine universelle (non spécialisée) qui caractérise les machines possédant les éléments suivants :

- une mémoire contenant programme (actions ou instructions) et données,
- une unité arithmétique et logique (UAL),
- une unité permettant l'échange d'information avec les périphériques : l'unité d'entrée/sortie (E/S),
- une unité de commande (UC).

- Modèle Von Neuman

Le fonctionnement schématique en est le suivant : l'UC

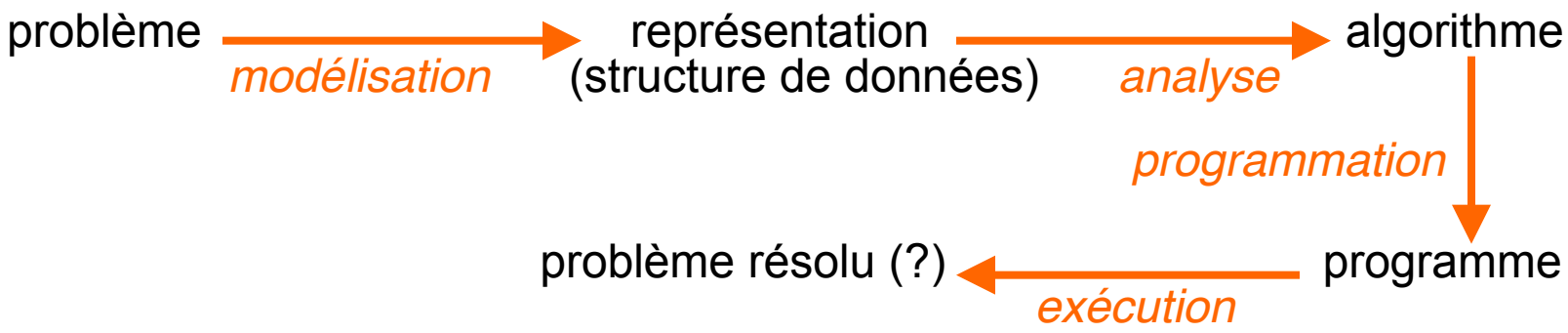
1. extrait une instruction de la mémoire,
2. analyse l'instruction,
3. recherche dans la mémoire les données concernées par l'instruction
4. déclenche l'opération adéquate sur l'UAL ou l'E/S,
5. range au besoin le résultat dans la mémoire.

Algorithme

- Un *algorithme* est l'expression de la résolution d'un problème de sorte que le résultat soit calculable par machine
- L'algorithme est exprimé dans un modèle théorique de machine universelle (Von Neumann) qui ne dépend pas de la machine réelle sur laquelle on va l'utiliser
- Il peut être écrit en langage naturel, mais pour être lisible par tous, on utilise un *langage algorithmique* plus restreint qui comporte tous les concepts de base de fonctionnement d'une machine

Schéma général

- On a l'enchaînement suivant :

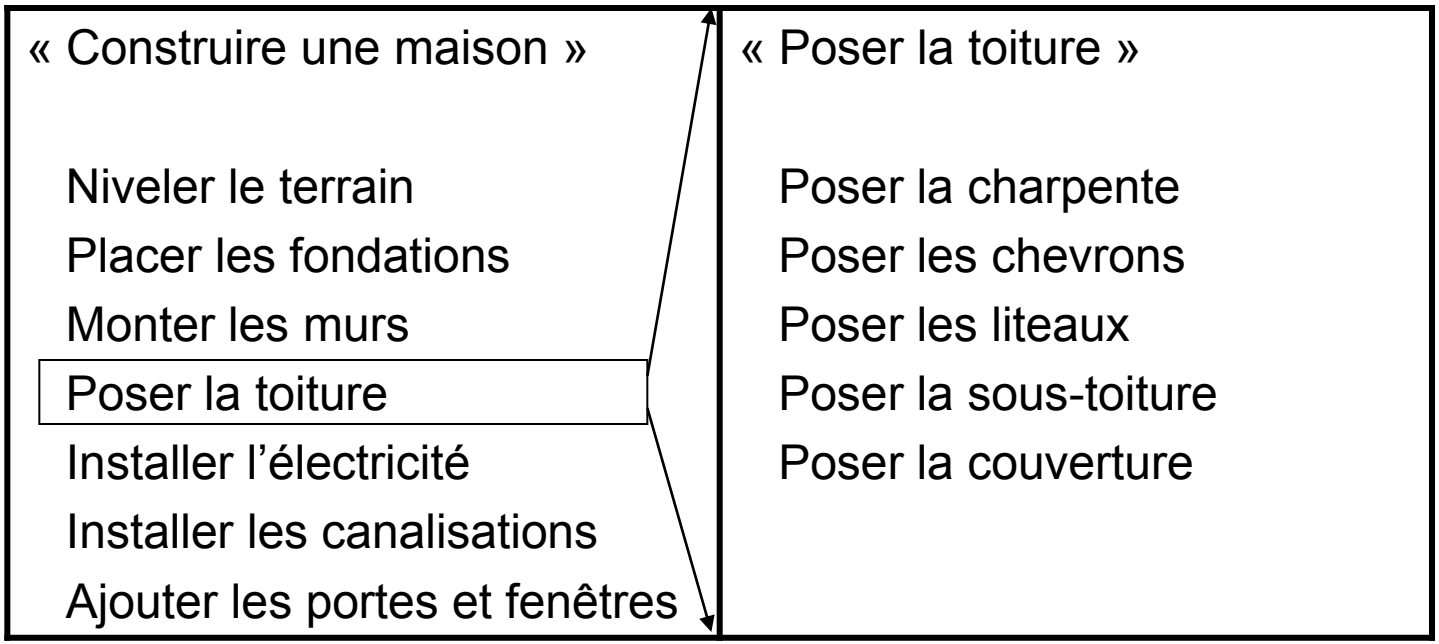


Problème et énoncé

- Un problème a un *énoncé* qui donne des informations sur le *résultat* attendu
- Il peut être en langage naturel, imprécis, incomplet et ne donne généralement pas la façon d'obtenir le résultat
- Exemples :
  - Calculer le PGDC de 2 nombres
  - Calculer la surface d'une pièce
  - Ranger des mots par ordre alphabétique
  - Calculer x tel que x divise a et b et si y divise a et b alors $x \geq y$
- Le dernier exemple décrit très précisément le résultat mais ne nous dit pas *comment le calculer*

Résolution de problèmes

- Pour résoudre un problème, il faut donner *la suite d'actions élémentaires* à réaliser pour obtenir le résultat
- Les actions élémentaires dont on dispose sont définies par le langage algorithmique utilisé
- Exemple :



Parfois, il faut *décomposer* les actions trop complexes

Un exemple

- L'ordre des opérations a son importance, mais dans certains cas plusieurs ordres sont possibles

- Algorithme d'Euclide : Calcule le PGDC de 2 entiers a et b

1) Ordonner les deux nombres tel que **a** $\geq$ **b**
2) Calculer leur différence **c** = **a**-**b**
   Recommencer en remplaçant **a** par **c** (**a**=**c**)
   jusqu'à ce que **a** devienne égal à **b**

Exécution avec 174 et 72

Étape 1 : a=174 et b=72

Étape 2 : c=174-72=102

Étape 3 : a=c=102 ≠ b=72

Étape 4 : a=102 et b=72

Étape 5 : c=102-72=30

Étape 6 : a=c=30 ≠ b=72

Étape 7 : a=72 et b=30

Étape 8 : c=72-30=42

Étape 9 : a=42 ≠ b=30

Étape 10 : a=42 et b=30

Étape 11 : c=42-30=12

Étape 12 : a=12 ≠ b=30

…

Suite et fin

…

Étape 13 : a=30 et b=12

Étape 14 : c=30-12=18

Étape 15 : a=18 ≠ b=12

Étape 16 : a=18 et b=12

Étape 17 : c=18-12=6

Étape 18 : a=6 ≠ b=12

Étape 19 : a=12 et b=6

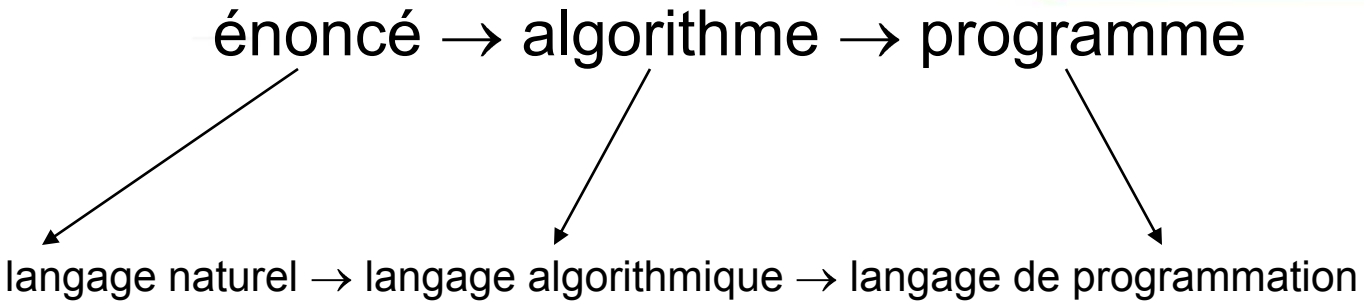
Étape 20 : c=12-6=6

Étape 21 : a=6 = b=6

Arrêt car a=b

le résultat est donc 6

Langages



- Un langage est composé de deux éléments :
  - La *grammaire* qui fixe la syntaxe
  - La *sémantique* qui donne le sens
- Le langage répond donc à deux questions :
  - Comment écrire les choses ?
  - Que signifient les choses écrites ?

Variable

- Une variable désigne
  - un **emplacement mémoire**,
  - muni d'un **identificateur** (nom de la variable),
  - auquel une **valeur** a été affectée (contenu codé de l'emplacement mémoire).

- Ne pas confondre variable et valeur. La valeur change pendant l'exécution du programme.

Type

- Ensemble de valeurs.
- Détermine la nature de ces valeurs (leur 'sémantique').

Types utilisés

Entier

- Pour manipuler des nombres entiers

Réel

- Pour manipuler des nombres réels

Booléen

- Pour manipuler des valeurs booléennes vrai ou faux

Caractère

- Pour manipuler des caractères alphabétiques et numériques

Chaîne

- Pour manipuler des chaînes de caractères permettant de représenter des mots ou des phrases

Type Entier

Les opérations utilisables sur les entiers sont :

- les opérateurs arithmétiques classiques : + (addition), - (soustraction), * (produit)
- la division entière, notée , telle que *n* *p* donne la partie entière du quotient de la division de n par p
- le modulo, *notée* mod, telle que n mod p donne le reste de la division entière de n par p
- les opérateurs de comparaison classiques : <, >,=,≠, ≥, ≤

Les opérations utilisables sur les réels sont :

- les opérateurs arithmétiques classiques :
  + (addition), - (soustraction), * (produit), / (division)

- les opérateurs de comparaison classiques : <, >,=,≠, ≥, ≤

Type Booléen

Il s'agit du domaine dont les seules valeurs sont vrai ou faux. Les opérations utilisables sur les booléens sont réalisées à l'aide des connecteurs logiques : et (pour le et logique), ou (pour le ou logique), non (pour le non logique)

Rappel :

non	
vrai	faux
faux	vrai

et	*vrai*	*faux*
vrai	vrai	faux
faux	faux	faux

ou	*vrai*	*faux*
vrai	vrai	vrai
faux	vrai	faux

Type Caractère

- Il s'agit du domaine constitué des caractères alphabétiques et numériques.
- Les opérations élémentaires réalisables sont les comparaisons : <, >,=,≠, ≥, ≤.

Type Chaîne

- Une chaîne est une séquence de plusieurs caractères.
- Les opérations élémentaires réalisables sont les comparaisons : <, >,=,≠, ≥, ≤ selon l'ordre lexicographique.

Affectation

- Modifie **le contenu** d'une variable.

Exemple : *var1* ← *37*

affecte la valeur 37 à la variable var1.
var1 étant de type entier on ne doit pas affecter une valeur réelle (3.5) ou booléenne (*true*)

- Syntaxe : ***identificateur ← valeur littérale du type***

Affectation du contenu d'une variable à une autre variable

- **Recopie le contenu** d'une variable dans une autre.

  Syntaxe : ***identificateur2*** ← ***identificateur1*;**

Exemple 1 :

x ← *87.62*
y ← *x*

Exemple 2 :

num1 ← *8762*
val_entiere ← *num2*
num2 ← *num1*

- La variable dont le contenu est recopié doit avoir été initialisée (contenir une valeur).

Expressions

- Permettent de « fabriquer » des valeurs à partir d'autres valeurs : des **opérateurs** sont appliqués à des valeurs ou à des (sous-)expressions placées entre parenthèses.
- **Evaluées** pendant l'exécution. Peuvent alors être assimilées à leur valeur.
- **Expressions arithmétiques** (addition, multiplication de valeurs numériques) et **expressions logiques** (comparaisons dont la valeur est de type *booléen)*.
- Expressions arithmétiques ≈ formules mathématiques.

  Ex : *2*3*

  2 (12+1) division entière!!

  34 mod 5 reste de la division entière (modulo)

  4*(-6*(98+97))

  (0 > 4)

Exemple d'algorithme

algorithme

début

x ← 12

y ←x + 4

x ←3

fin

lexique

- x : entier
- y : entier

Exemple d'algorithme

algorithme
début
x ← 12
y ←x + 4
x ←3
fin
lexique
- x : entier
- y : entier

L'ordre est très important : les deux premières instructions ne sont pas permutables. Pourquoi ???

Exemple d'algorithme (suite)

- Schéma de l'évolution de l'état des variables instruction par instruction

instructions / variables	1	2	3
x	12		3
y		16	

Instructions de lecture et d'écriture

- Les mécanismes **d'entrées-sorties** permettent au programme de communiquer avec l'extérieur.

- **Instruction de lecture**

variable ← lire()

  - C'est une instruction de prise de données sur le périphérique d'entrée (en général le clavier). L'exécution de cette instruction consiste à affecter une valeur à la variable en prenant cette valeur sur le périphérique d'entrée.

- **Instruction d'écriture**

écrire (liste d'expressions)

  - Cette instruction réalise simplement l'affichage des valeurs des expressions décrites dans la liste. Ces expressions peuvent être simplement des variables ayant des valeurs ou même des nombres ou des commentaires écrits sous forme de chaîne de caractères.
  - Exemple : écrire (x, y+2, « bonjour »)

Exercice (page 4)

- On désire écrire un algorithme qui lit sur le périphérique d'entrée une valeur représentant une somme d'argent et qui calcule et affiche le nombre de billets de 50 euros et 10 euros, et de pièces de 2 euros et 1 euro qu'elle représente.
- **Principe :** L'algorithme commence par lire sur l'entrée standard l'entier qui représente la somme d'argent et affecte la valeur à une variable somme. Pour obtenir la décomposition en nombre de billets et de pièces de la somme d'argent , on procède par des divisions successives en conservant chaque fois le reste.

Solution de l'exercice

```
algorithme
        début
        somme ← lire ( )                    // 1
        b50 ← somme ÷ 50                    // 2
        r50 ← somme mod 50                  // 3
        b10 ← r50 ÷ 10                      // 4
        r10 ← r50 mod 10                    // 5
        p2 ← r10 ÷ 2                        // 6
        r2 ← r10 mod 2                      // 7
        p1 ← r2                             // 8
        écrire (b50, b10, p2, p1)           // 9
        fin
```

lexique

- somme : entier, la somme d'argent à décomposer
- b50: entier, le nombre de billets de 50 euros
- b10: entier, le nombre de billets de 10 euros
- p2: entier, le nombre de pièces de 2 euros
- p1: entier, le nombre de pièces de 1 euro
- r50: entier, reste de la division entière de somme par 50
- r10: entier, reste de la division entière de r50 par 10
- r2: entier, reste de la division entière de r10 par 2

Solution de l'exercice (suite)

Schéma de l'évolution de l'état des variables instruction par instruction

instructions / variables	1	2	3	4	5	6	7	8	9
somme	188								
b50		3							écrire
b10				3					écrire
p2						4			écrire
p1								0	écrire
r50			38						
r10					8				
r2							0		

Exemple

- Calculer le périmètre de cinq rectangles

Algorithme

début

… lecture de toutes les variables …

périmètre1 ← longueur1 + longueur1+ largeur1 + largeur1
périmètre2 ← longueur2 + longueur2+ largeur2 + largeur2
périmètre3 ← longueur3 + longueur3+ largeur3+ largeur3
périmètre4 ← longueur4 + longueur4+ largeur4+ largeur4
périmètre5 ← longueur5 + longueur5 + largeur5 + largeur5
écrire (périmètre1,…, périmètre5)
fin

Lexique

périmètre1,…, périmètre5 : entier
longueur1, …, longueur5 : entier
largeur1, …, largeur5 : entier

Notion de fonctions

- Une fonction est un algorithme autonome, réalisant une tâche précise, auquel on transmet des valeurs lors de son appel et qui retourne une valeur à la fin de son exécution. La notion de fonction est très intéressante car elle permet, pour résoudre un problème, d'employer une méthode de décomposition en sous-problèmes distincts. Elle facilite aussi la réutilisation d'algorithmes déjà développés par ailleurs.
- Une fonction est introduite par un *en-tête, appelé aussi signature ou prototype, qui spécifie :*
  - – le nom de la fonction,
  - – les paramètres donnés et leur type,
  - – le type du résultat.
- La syntaxe retenue pour l'en-tête est la suivante :

  fonction nomFonction (liste des paramètres) : type du résultat
- La liste des paramètres précise, pour chaque paramètre, son nom et son type. La dernière instruction de la fonction indique la valeur retournée, elle n'apparaît qu'une seule fois et est toujours placée à la fin du corps de la fonction, nous la noterons:

  retourne expression

Exercice sur les fonctions (page 5)

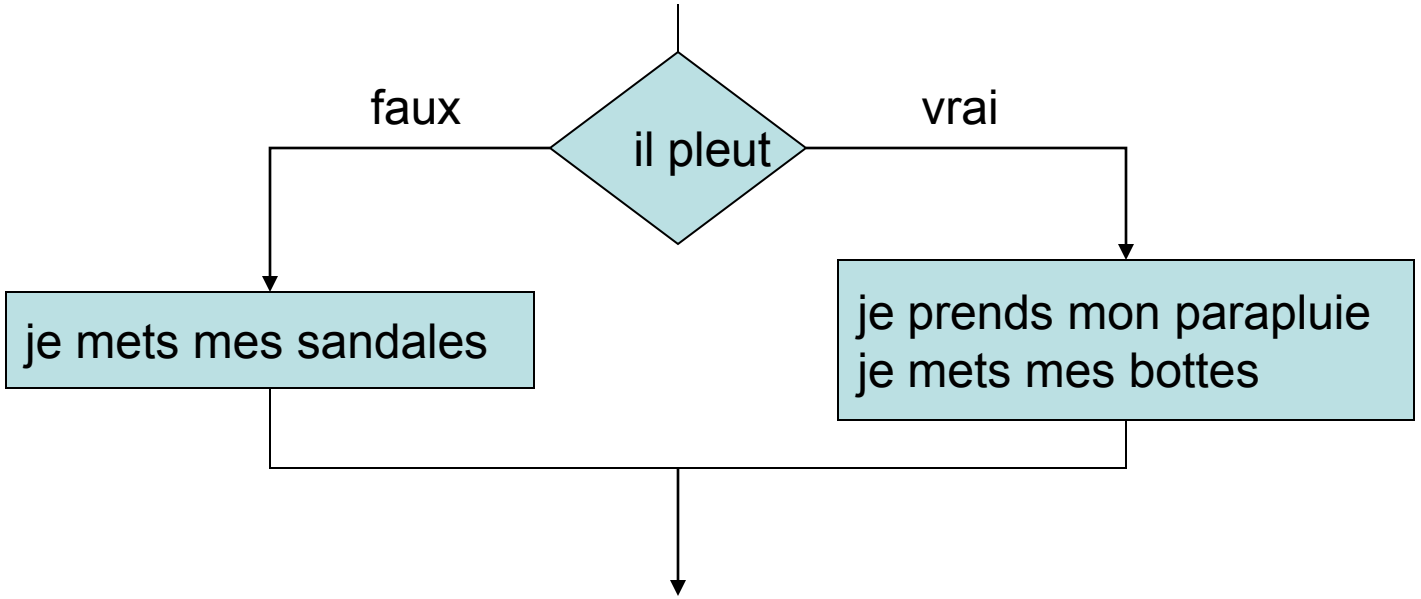
Ecrire une fonction calculant le périmètre d'un rectangle dont on donne la longueur et la largeur.

Solution :

```
fonction calculerPérimètreRectangle (longueur: réel, largeur: réel) : réel
        début
            périmètre ← 2*(longueur + largeur)
            retourne périmètre
        fin
lexique
-   longueur : réel, longueur du rectangle
-   largeur : réel, largeur du rectangle
-   périmètre : réel, périmètre du rectangle
```

Les conditionnelles

- Possibilité de choisir une séquence d'instructions selon une condition donnée
- Exemple :
  "s'il pleut, je prends mon parapluie et je mets mes bottes sinon je mets mes sandales"



Les conditionnelles (suite)

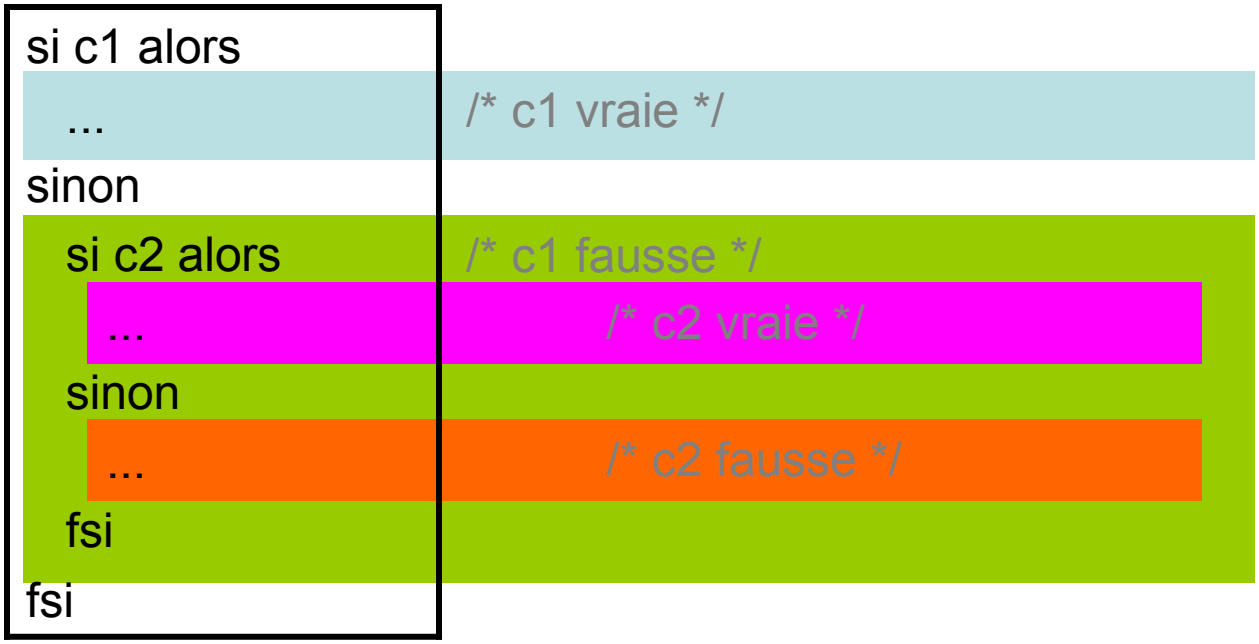
- Synopsis :

si <*condition*> alors
  <*instruction*>
  <*instruction*>
  ...
sinon
  <*instruction*>
  <*instruction*>
  ...
fsi
	si <*condition*> alors
	  <*instruction*>
	  <*instruction*>
	  ...
	fsi

- La condition est une expression booléenne {vrai,faux}
- Si la condition est vraie, on exécute la branche *alors*
- Si la condition est fausse, on exécute la branche *sinon*

Imbrication de conditionnelles

- Toute instruction algorithmique peut être placée dans une conditionnelle, donc également une conditionnelle !
- Cela permet de multiplier les choix possibles d'exécution
- Exemple :



- L'ordre des imbrications est généralement important

Exercice : conditionnelle simple (page 6)

Ecrire un algorithme qui permet d'imprimer le résultat d'un étudiant à un module sachant que ce module est sanctionné par une note d'oral de coefficient 1 et une note d'écrit de coefficient 2. La moyenne obtenue doit être supérieure ou égale à 10 pour valider le module.

données : la note d'oral et la note d'écrit

résultat : impression du résultat pour le module (*reçu ou refusé)*

principe : on calcule la moyenne et on la compare à 10.

Solution de l'exercice : conditionnelle simple (page 6)

algorithme :

début

ne ← lire ( )

no ← lire ( )

moy← (ne * 2 + no) / 3

si moy ≥ 10

alors écrire (« reçu »)

sinon écrire (« refusé »)

fsi

fin

lexique :

- moy : réel, moyenne
- ne : réel, note d'écrit
- no : réel, note d'oral

Exercice : conditionnelles imbriquées (page 6)

On veut écrire une fonction permettant de calculer le salaire d'un employé payé à l'heure à partir de son salaire horaire et du nombre d'heures de travail.

Les règles de calcul sont les suivantes :
le taux horaire est majoré pour les heures supplémentaires 25% au-delà de 160 h et 50% au-delà de 200 h.

Solution de l'exercice : conditionnelles imbriquées (page 6)

```
fonction calculerSalaire ( sh : réel, nbh : entier) : réel
        début
        si   nbh ≤ 160
            alors   salaire ← nbh * sh
            sinon  si  nbh ≤ 200
                                alors salaire ← 160 * sh + (nbh - 160) * sh * 1.25
                                sinon salaire ←160 * sh + 40 * sh * 1,25 + (nbh - 200) * sh * 1.5
                        fsi
        fsi
        retourne salaire
        fin
lexique
-    sh : réel, salaire horaire de l'employé
-    nbh : entier, nombre d'heures de travail de l'employé
-    salaire : réel, salaire de l'employé
```

Algorithmique

Itérations

Motivations

- Exemple : Ecrire l'algorithme permettant d'afficher la table de multiplication par 9. Un algorithme possible est le suivant :

```
algorithme
        début
        écrire(1*9)
        écrire(2*9)
        écrire(3*9)
        écrire(4*9)
        écrire(5*9)
        écrire(6*9)
        écrire(7*9)
        écrire(8*9)
        écrire(9*9)
        écrire(10*9)
        fin
```

Motivations (suite)

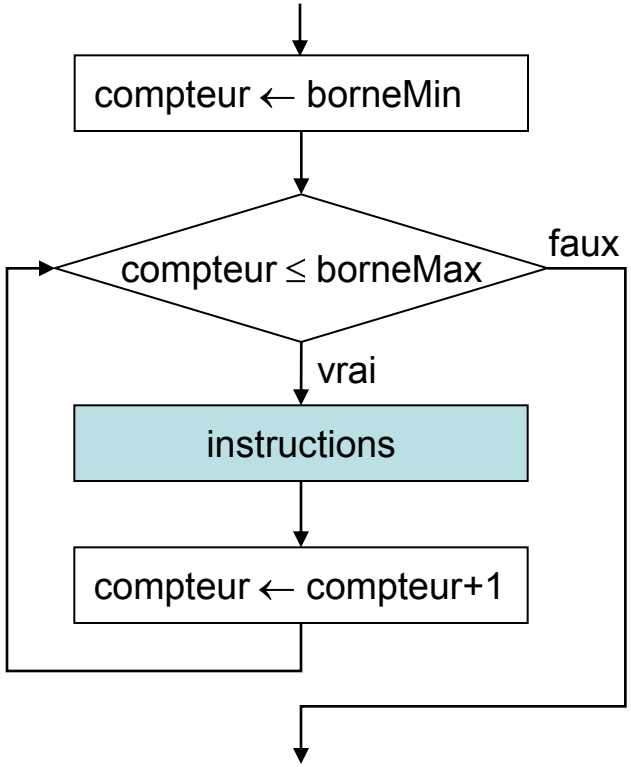
- Il arrive souvent dans un algorithme que la même action soit répétée plusieurs fois, avec éventuellement quelques variations dans les paramètres qui précisent le déroulement de l'action.
- Il est alors fastidieux d'écrire un algorithme qui contient de nombreuses fois la même instruction. De plus, ce nombre peut dépendre du déroulement de l'algorithme. Il est alors impossible de savoir à l'avance combien de fois la même instruction doit être décrite.

Répétitions inconditionnelles

- Lorsque la répétition ne porte que sur le nombre d'itérations et qu'il est connu avant de commencer la boucle, on utilise une écriture plus condensée, c'est la structure de contrôle **pour**

- Synopsis :

```
pour <compteur> de <borneMin> à <borneMax>
   <instruction>
   <instruction>
   ...
fpour
```



Répétitions inconditionnelles (suite)

Exemple de table de multiplication par 9

Il est plus simple d'utiliser une boucle avec un compteur prenant d'abord la valeur 1, puis augmentant peu à peu jusqu'à atteindre 10

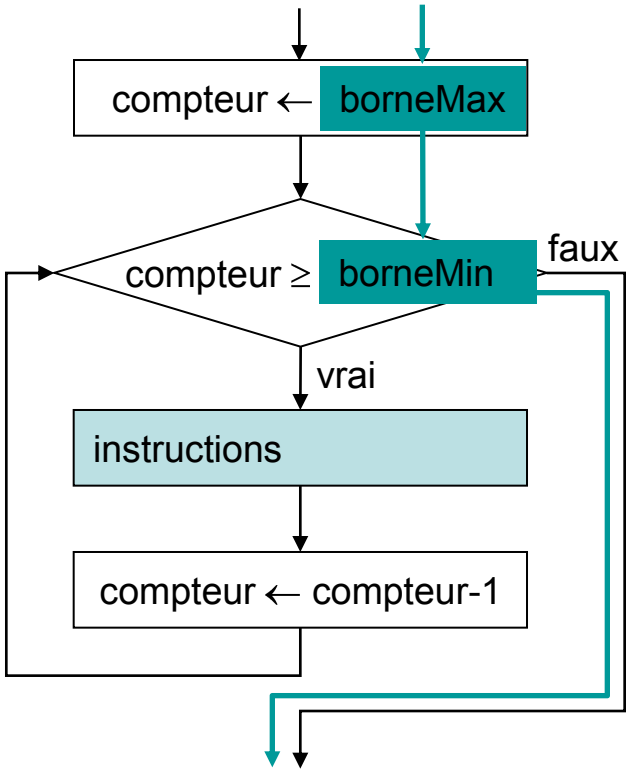
```
algorithme
        début
        pour i de 1 à 10 faire
            écrire (i*9)
        fpour
        fin
lexique
-     i : entier, indice d'itération
```

Répétitions inconditionnelles (suite)

- Par défaut, une boucle **pour** va dans le sens croissant
- On peut inverser le sens : décroissant
- Synopsis :

```
pour <compteur> décroissant de <borneMax> à <borneMin>
   <instruction>
   <instruction>
   ...
fpour
```

On n'entre pas dans la boucle
si ses bornes sont dans l'ordre
inverse de son sens



Répétitions inconditionnelles (suite)

Compte à rebours : écrire l'algorithme de la fonction qui, à partir d'un nombre entier positif *n,* affiche tous les nombres par ordre décroissant jusqu'à 0.

Exemple : pour n = 5, le résultat sera 5 4 3 2 1 0.

```
fonction compteARebours ( n : entier)
        début
        pour i décroissant de n à 0 faire
                écrire (i)
        fpour
        fin
lexique
-   n : entier,
-   i : entier, indice d'itération
```

Exercice : Calcul par récurrence de la somme (page 9)

On veut imprimer, pour *n donné, la somme des carrés des n premiers entiers.*

Cette somme, notée s, est obtenue en calculant le n-ième terme d'une suite définie par récurrence :

$$s_n = s_{n-1} + n^2$$

Algorithmiquement, le calcul d'une telle suite se fait en deux étapes :

- initialisation (ici, $s_0 = 0$),

- répétition de : calcul du $i^{ème}$ terme en fonction du terme d'indice i-1

Exercice : Calcul par récurrence de la somme

Solution :

```
algorithme
        début
        n ← lire ( )                    // 1
        s ← 0                           // 2
        pour i de 1 à n faire           // 3
                s ← s + i²              // 4
        fpour
        écrire (s)                      // 5
        fin
lexique
-   s : entier, somme des carrés des n premiers entiers
-   n : entier,
-   i : entier, indice d'itération
```

Exercice : Calcul par récurrence de la somme

Solution :

Schéma de l'évolution de l'état des variables instruction par instruction. On suppose que la valeur introduite par l'utilisateur est 4.

variables / instructions	n	s	i
1	4		
2		0	
3			1
4		1	
3			2
4		5	
3			3
4		14	
3			4
4		30	
3			(fin)
5		écrire	

Exercice : Calcul par récurrence d'un maximum

Cas 1 : Initialisation à un terme artificiel

Ecrire l'algorithme qui permet d'imprimer le maximum de *n* entiers positifs donnés au fur et à mesure.

Comment trouver ce maximum ? C'est le plus grand des 2 nombres : maximum des n-1 premiers entiers positifs donnés, n-ème entier donné.

Ceci est une définition par récurrence :

si $nombre_n > maximum_{n-1}$
alors $maximum_n \leftarrow nombre_n$
sinon $maximum_n \leftarrow maximum_{n-1}$
fsi

Comment initialiser la suite maximum *?* Il faut que $maximum_1$ prenne la valeur $nombre_1$. Il suffit alors de choisir une valeur de $maximum_0$ plus petite que tout entier positif, par exemple $maximum_0 = -1$, de manière à assurer que $maximum_1$ aura bien $nombre_1$ pour valeur. Il s'agit d'une initialisation à un terme artificiel.

Exercice : Calcul par récurrence d'un maximum

Cas 1 : Initialisation à un terme artificiel (solution)

```
algorithme
        début
        n ← lire ( )                                    // 1
        maximum ← -1                                    // 2
        pour i de 1 à n faire                           // 3
            nombre ← lire ( )                           // 4
            si nombre > maximum                         // 5
                    alors maximum ← nombre              // 6
            fsi
        fpour
        écrire (maximum)                                // 7
        fin
lexique
-   n : entier,
-   maximum : entier, maximum des i premiers nombres entiers
-   nombre : entier, ième entier positif donné
-   i : entier, indice d'itération
```

Exercice : Calcul par récurrence d'un maximum

Cas 1 : Initialisation à un terme artificiel (solution)

Schéma de l'évolution de l'état des variables instruction par instruction. On suppose les valeurs introduites par l'utilisateur sont : 4 2 0 8 7

variables / instructions	n	maximum	i	nombre	nombre>maximum
1	4				
2		-1			
3			1		
4				2	
5					vrai
6		2			

Exercice : Calcul par récurrence d'un maximum

Cas 1 : Initialisation à un terme artificiel (solution)

Schéma de l'évolution de l'état des variables instruction par instruction. On suppose les valeurs introduites par l'utilisateur sont : 4 2 0 8 7

3			2		
4				0	
5					faux
3			3		
4				8	
5					vrai
6		8			
3			4		
4				7	
5					faux
3			(fin)		
7		écrire			

Cas 2 : Initialisation à un terme artificiel utile

Ecrire l'algorithme qui permet d'imprimer le maximum de *n* entiers donnés.

L'initialisation a un terme artificiel n'est plus possible ici car les valeurs ne sont plus bornées inférieurement. Une solution consiste alors à initialiser au premier terme et à commencer la formule générale de récurrence à 2. Il s'agit d'une initialisation à un terme utile.

Cas 2 : Initialisation à un terme artificiel utile (solution)

```
algorithme
        début
        n ← lire ( )                                    // 1
        maximum ← lire ( )                              // 2
        pour i de 2 à n faire                           // 3
            nombre ← lire ( )                           // 4
            si nombre > maximum                         // 5
                    alors maximum ← nombre              // 6
            fsi
        fpour
        écrire (maximum)                                // 7
        fin
lexique
-   n : entier,
-   maximum : entier, maximum des i premiers nombres entiers
-   nombre : entier, ième entier positif donné
-   i : entier, indice d'itération
```

Exercice : Calcul par récurrence d'un maximum

Cas 2 : Initialisation à un terme artificiel utile (solution)

Schéma de l'évolution de l'état des variables instruction par instruction. On suppose les valeurs introduites par l'utilisateur sont : 4 2 0 8 7

variables / instructions	n	maximum	i	nombre	nombre>maximum
1	4				
2		2			
3			2		
4				0	
5					faux
3			3		
4				8	
5					vrai
6		8			
3			4		
4				7	
5					faux
3			(fin)		
7		écrire			

Répétitions conditionnelles

L'utilisation d'une boucle ***pour*** *nécessite de connaître à l'avance* le nombre d'itérations désiré, c'est-à-dire la valeur finale du compteur.

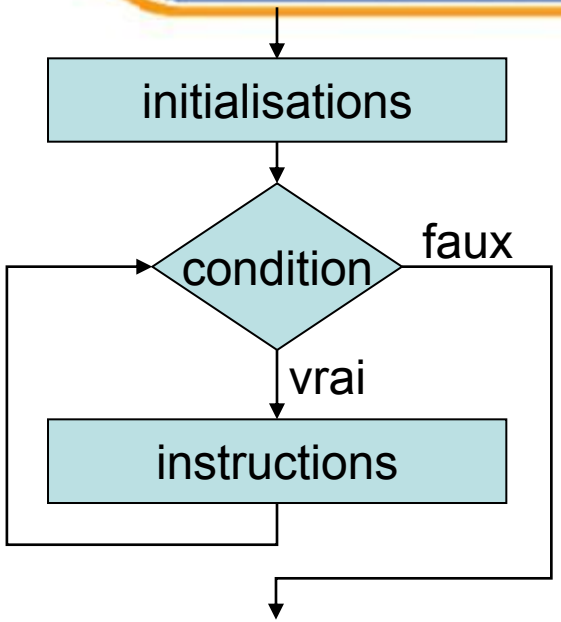
Dans beaucoup de cas, on souhaite répéter une instruction tant qu'une certaine condition est remplie, alors qu'il est a priori impossible de savoir à l'avance au bout de combien d'itérations cette condition cessera d'être satisfaite.

Le mécanisme permettant cela est la boucle **Tant que.**

Répétitions conditionnelles (suite)

- Synopsis :

```
<initialisations>
tant que <condition>
   <instruction>
   <instruction>
   ...
ftant
```



- Les *initialisations* spécifient les valeurs initiales des variables intervenant dans la *condition*
- Il faut *au moins une* instruction dans la boucle susceptible de modifier la valeur de la condition
- Les *instructions* de la boucle peuvent *ne pas* être exécutées

Répétitions conditionnelles (suite)

Exemple : On veut laisser un utilisateur construire des rectangles de taille quelconque, à condition que les largeurs qu'il indique soient supérieures à 1 pixel. On peut utiliser une répétition conditionnelle qui permet de redemander à l'utilisateur de saisir une nouvelle valeur tant que celle-ci n'est pas valide.

```
fonction saisirLargeurRectangle ( ) : entier
        début
        écrire (« indiquez la largeur du rectangle :  »)
        largeur ← lire ( )
        tant que largeur < 1 faire
                écrire (« erreur : indiquez une valeur strictement positive »)
                écrire (« indiquez la largeur du rectangle : »)
                largeur ← lire ( )
        ftant
        retourne largeur
        fin
lexique
-    largeur : entier, largeur courante saisie
```

Exercice sur les répétitions conditionnelles

Enoncé :
Un poissonnier sert un client qui a demandé 1kg de poisson. Il pèse successivement différents poissons et s'arrête dès que le poids total égale ou dépasse 1kg. Donner le nombre de poissons servis.

Remarque sur la terminaison :
Ce problème est typique des cas où le dernier terme (celui qui fait basculer le test) doit être retenu. Nous verrons en exercice des problèmes dans lesquels le dernier terme (celui qui fait basculer le test) doit être rejeté (exemple : le passager d'un ascenseur qui fait dépasser la charge maximale).

Exercice sur les répétitions conditionnelles (solution)

données
poids des poissons successifs en grammes

résultats
nombre de poissons vendus

algorithme
```
        début
        poidstotal ← 0                                  // 1
        nbpoisson ← 0                                   // 2
        tant que poidstotal < 1000 faire // itération i  // 3
            poidspoisson ← lire ( )                     // 4
            nbpoisson ← nbpoisson + 1                   // 5
            poidstotal ← poidstotal + poidspoisson      // 6
        ftant
        écrire (nbpoisson)                              // 7
        fin
```
lexique
- poidspoisson : réel, poids du i ème poisson, en grammes
- nbpoisson : entier, nombre de poissons vendus après la i ème itération (c'est i)
- poidstotal : réel, poids total après la i ème itération (poids des i premiers poissons)

Exercice sur les répétitions conditionnelles (solution)

Schéma de l'évolution de l'état des variables instruction par instruction. On suppose les valeurs introduites par l'utilisateur sont : 350 280 375

variables / instructions	poidstotal	nbpoisson	poidspoisson	poidstotal < 1000
1	0			
2		0		
3				vrai
4			350	
5		1		
6	350			
3				vrai
4			280	
5		2		
6	630			
3				vrai
4			375	
5		3		
6	1005			
3				faux
7		écrire		

Boucles imbriquées

Le corps d'une boucle est une liste d'instructions. Mais cette boucle est elle-même une instruction. Donc le corps d'une boucle peut contenir une boucle dite imbriquée, qui utilise un compteur différent. Il faut alors faire attention à l'ordre d'exécution des instructions : à chaque passage dans le corps de la boucle principale, la boucle imbriquée va être exécutée totalement.

Exemple

Ecrire les tables de multiplication de 1 à 9 :

```
Pour i de 1 à 9 faire
        Pour j de 1 à 9 faire
                écrire (j*i)
        Fpour
Fpour
```

Boucles imbriquées (suite)

Autre exemple

Ecrire l'algorithme permettant d'imprimer le triangle suivant, le nombre de lignes étant donné par l'utilisateur :

1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
...

Boucles imbriquées

```
algorithme
        début
        nblignes ← lire ( )                 // 1
        pour i de 1 à nblignes faire        // 2
            pour j de 1 à i faire           // 3
                    écrire (j )             // 4
            fpour
            retour à la ligne               //5
        fpour
        fin
```

lexique

- nblignes : entier, nombre de lignes à imprimer
- i : entier, indice d'itération sur les lignes
- j : entier, indice d'itération sur les éléments de la i ème ligne

Boucles imbriquées

variables / instructions	nblignes	i	j	
1	3			
2		1		
3			1	
4			écrire	
3			(fin)	
5				retour à la ligne
2		2		
3			1	
4			écrire	
3			2	
4			écrire	
3			(fin)	
5				retour à la ligne
2		3		
3			1	
4			écrire	
3			2	
4			écrire	
3			3	
4			écrire	
3			(fin)	
5				retour à la ligne
2		(fin)		

Schéma de l'évolution de l'état des variables instruction par instruction.

On suppose que la valeur introduite par l'utilisateur est : 3

Algorithmique

Compléments sur les fonctions

Introduction

- Morceaux d'algorithmes réutilisables à volonté
- Les fonctions permettent la décomposition des algorithmes en sous-problèmes (raffinements successifs)
- Prennent en entrée des paramètres
- Restituent à l'algorithme appelant un résultat
- Définies par :
  - Un en-tête :
    - *Nom* : identifiant de la fonction
    - *Liste des paramètres* : informations extérieures à la fonction
    - *Résultat* : valeur de retour de la fonction
    - *Description* en langage naturel du rôle de la fonction
  - Un corps :
    - Algorithme de la fonction

En-tête de la fonction

- Repéré par le mot clé fonction
- Exemple :

optionnel

```
fonction nomFonction( mode nomParam : typeParam, … ) : typeRet
/* indique ce que fait la fonction et le rôle de ses paramètres */
```

  - nomFonction : identifiant de la fonction
  - mode : à donner pour chaque paramètre
    - in : paramètre *donnée*, non modifiable par la fonction
    - out : paramètre *résultat*, modifié par la fonction, valeur initiale non utilisée
    - in-out : paramètre *donnée/résultat*, valeur initiale utilisée, modifié par la fonction
  - nomParam : identifiant du paramètre dans la fonction
  - typeParam : type du paramètre
  - retour :
    - typeRet donne le type de la valeur renvoyée par la fonction
    - Si pas de retour, on remplace typeRet par le mot clé vide

Paramètres formels / effectifs

Les paramètres de la fonction *ne doivent pas* être redéfinis dans le corps de la fonction

- Ce sont déjà des variables/constantes *locales* à la fonction dans lesquelles on recopie les éléments reçus de l'algorithme appelant
- Les paramètres dans l'en-tête de la fonction sont les *paramètres formels*
- Les paramètres utilisés lors de l'appel de la fonction par l'algorithme appelant sont les *paramètres effectifs*
- Les fonctions peuvent ne pas avoir de paramètres

Exemple

fonction volPrisme(côté : réel, hauteur : réel) : réel
/* calcule le volume d'un prisme de côté et hauteur donnés */

Lexique local des variables :

hauteurTri	(réel)	Hauteur du triangle formant la base	INTERMÉDIAIRE
surfTri	(réel)	Surface du triangle formant la base	INTERMÉDIAIRE

Algorithme de volPrisme :

hauteurTri ← côté * sqrt(3) / 2
surfTri ← côté * hauteurTri / 2
retourne (surfTri * hauteur)

Exemple (suite)

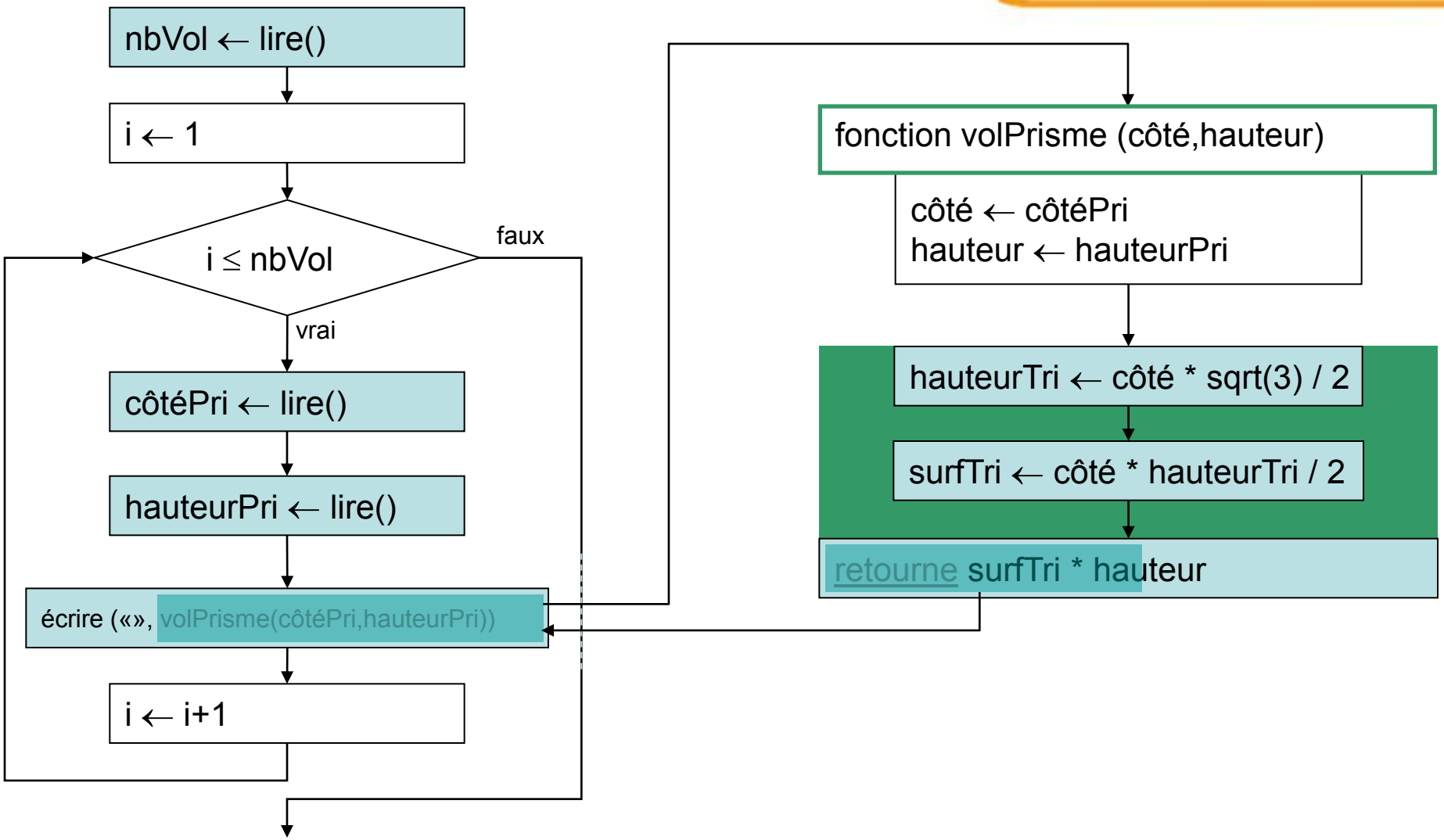
Lexique des variables :

nbVol	(entier)	Nombre de volumes à calculer	DONNÉE
i	(entier)	Compteur de la boucle	INTERMÉDIAIRE
côtéPri	(réel)	Côté d'un prisme	DONNÉE
hauteurPri	(réel)	Hauteur d'un prisme	DONNÉE

Algorithme :

```
nbVol ← lire()
pour i de 1 à nbVol
   côtéPri ← lire()
   hauteurPri ← lire()
   écrire (« le volume du prisme est », volPrisme(côtéPri,hauteurPri))
fpour
```

Explication



Paramètres modifiables

Les paramètres modifiables sont modifiés par des fonctions après chaque appel. La modification reste inchangée même après la fin de l'appel d'une fonction.

Exemple :

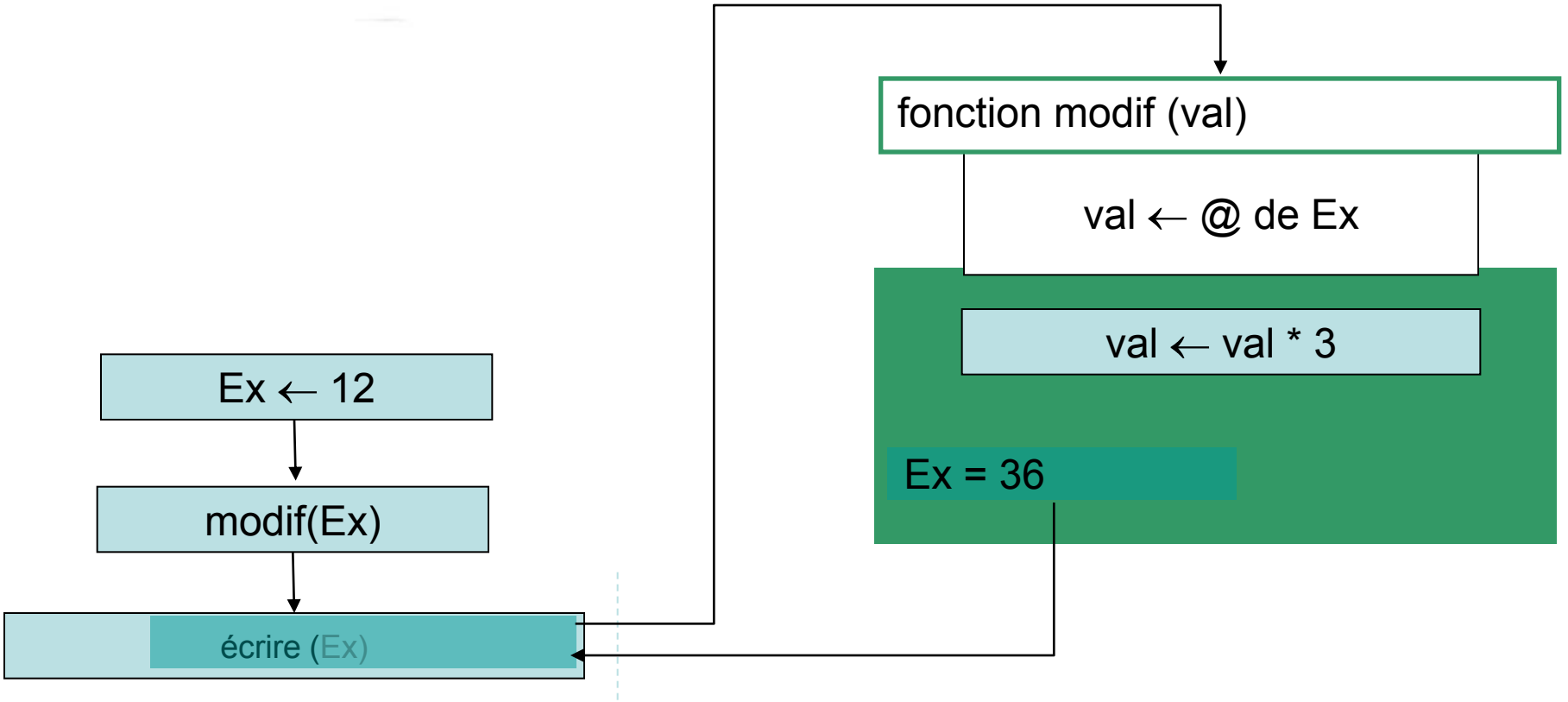
```
fonction modif( val InOut : réel)
/* Modifie la valeur d'une variable passée en paramètre */

début

  val ← val * 3

  fin
```

Paramètres modifiables (suite)



Exemple 1

La fonction suivante prend deux paramètres réels et les modifie de manière à ce qu'ils soient dans l'ordre croissant

```
fonction ordonner2Réels(x InOut : réel, y InOut : réel)
        début
            si x > y
                alors
                    tmp ← x
                    x ← y
                    y ← tmp
            fsi
        fin
lexique :
-   x : réel, valeur fournie, est modifiée (éventuellement) en le minimum de x et y
-   y : réel, valeur fournie, est modifiée (éventuellement) en le maximum de x et y
-   tmp : réel, variable de stockage temporaire
```

Exemple 2

La fonction suivante prend trois paramètres réels : les deux premiers sont les bornes d'un intervalle, et le troisième est (éventuellement) modifié de manière à être dans l'intervalle spécifié.

```
fonction seuillerRéel(inf : réel, sup : réel, x InOut : réel)
        début
            si x < inf
                alors  x ← inf
                sinon
                    si x > sup
                        alors x ← sup
                    fsi
            fsi
        fin
lexique :
-   inf : réel, borne inférieure de l'intervalle
-   sup : réel, borne supérieure de l'intervalle
-   x : réel, valeur fournie, est modifiée (éventuellement) par seuillage
```

Exemple 3

La fonction suivante est similaire à celle de l'exemple précédent. Elle renvoie en plus un booléen égal à vrai si et seulement si le paramètre x a été effectivement modifié.

```
fonction modifierRéelParSeuillage(inf : réel, sup : réel, x InOut : réel) : booléen
        début
            res ← faux
            si x < inf
                alors
                    x ← inf
                    res ← vrai
                sinon
                    si x > sup
                        alors
                            x ← sup
                            res ← vrai
                    fsi
            fsi
            retourne res
        fin
```

lexique :

- inf : réel, borne inférieure de l'intervalle
- sup : réel, borne supérieure de l'intervalle
- x : réel, valeur fournie, est modifiée (éventuellement) par seuillage
- res : booléen, résultat vrai si x est modifié

Exemple 4

Ecrire un algorithme qui calcule le maximum de 4 réels saisis au clavier en utilisant une fonction qui calcule le maximum de deux réels.

```
fonction calculerMax2Réels(x : réel, y : réel) : réel
        début
            si x > y                    // f1
                alors
                    res ← x             // f2
                sinon
                    res ← y             // f3
            fsi
            retourne res                // f4
        fin
```

lexique :

- x : réel,
- y : réel,
- res : réel, maximum trouvé

Exemple 4 (suite)

L'algorithme qui fait appel à la fonction :

```
algorithme
        début
        maximum ← lire ( )                                  // 1
        pour i de 2 à 4 faire                               // 2
            nombre ← lire ( )                               // 3
            maximum ← calculerMax2Réels(nombre,maximum)     // 4
        fpour
        écrire (maximum)                                    // 5
        fin
lexique
-   maximum : réel, maximum des i premiers nombres réels
-   nombre : réel, ième réel donné
-   i : entier, indice d'itération
```

Exemple 4 (suite)

Schéma de l'évolution de l'état des variables instruction par instruction :

On suppose que les valeurs introduites par l'utilisateur sont : 35 80 37 22.

variables / instructions	maximum (algo)	nombre (algo)	i (algo)	x (calculer…)	y (calculer…)	x>y (calculer…)	res (calculer…)
1	35						
2			2				
3		80					
4				80	35		
f1						vrai	
f2							80
f4, 4	80						Retourne
2			3				
3		37					
4				37	80		
f1						faux	
f3							80
f4, 4	80						Retourne
2			4				
3		22					
4				22	80		
f1						faux	
f3							80
f4, 4	80						Retourne
2			(fin)				
5	écrire						

Exemple 5

On souhaite résoudre le même problème que dans l'exemple 1 : calculer le maximum de 4 réels mais en utilisant la fonction ordonner2Réels.

Rappel :

```
fonction ordonner2Réels(x InOut : réel, y InOut : réel)
        début
            si x > y                                    //f1
                alors
                    tmp ← x                             //f2
                    x ← y                               //f3
                    y ← tmp                             //f4
            fsi
        fin
```

lexique :

- x : réel, valeur fournie, est modifiée (éventuellement) en le minimum de x et y
- y : réel, valeur fournie, est modifiée (éventuellement) en le maximum de x et y
- tmp : réel, variable de stockage temporaire

```
algorithme
        début
        maximum ← lire ( )                              // 1
        pour i de 2 à 4 faire                           // 2
            nombre ← lire ( )                           // 3
            ordonner2Réels (nombre, maximum)            // 4
        fpour
        écrire (maximum)                                // 5
        fin
```

lexique

- maximum : réel, maximum des i premiers nombres réels
- nombre : réel, ième réel donné
- i : entier, indice d'itération

Exemple 5 (suite)

Schéma de l'évolution de l'état des variables instruction par instruction :

On suppose que les valeurs introduites par l'utilisateur sont : 35 80 37 22.

variables / instructions	maximum (algo)	nombre (algo)	i (algo)	x (ordonner…)	y (ordonner…)	tmp (ordonner…)	x>y (ordonner…)
1	35						
2			2				
3		80					
4		←		←			
f1							vrai
f2						80	
f3		35					
f4,4	80						
2			3				
3		37					
4		←		←			
f1, 4							faux
2			4				
3		22					
4		←		←			
f1, 4							faux
2			(fin)				
5	écrire						

Exemple 6

Un étudiant doit, pour obtenir son diplôme, passer un écrit et un oral dans deux modules. Le coefficient du premier module est le double de celui du second module. La moyenne d'un module accorde un coefficient double à la meilleure des deux notes obtenues. On veut décrire un algorithme où, après saisie des quatre notes, la décision finale est affichée (diplôme obtenu si la moyenne est supérieure ou égale à 10, aucun module ne devant avoir une moyenne inférieure à 8).

```
fonction calculerMoyenne(n1 : réel, n2 : réel) : réel
        début
            moy ← (n1+2*n2)/3                    // cm1
            retourne moy                         // cm2
        fin
lexique :
-   n1 : réel, note de coefficient 1
-   n2 : réel, note de coefficient 2
-   moy : réel, moyenne calculée

fonction calculerNoteModule(n1 : réel, n2 : réel) : réel
        début
            si n1>n2                                    // cnm1
                alors
                    note ← calculerMoyenne(n2,n1)       // cnm2
                sinon
                    note ← calculerMoyenne(n1,n2)       // cnm3
            fsi
            retourne note                               // cnm4
        fin
lexique :
-   n1 : réel, première note
-   n2 : réel, deuxième note
-   note : réel, note du module
```

Exemple 6 (suite)

```
fonction accorderDiplôme(m1 : réel, m2 : réel) : booléen
        début
            moy ← calculerMoyenne(m2,m1)          // ad1
            si moy < 10                            // ad2
                alors
                    reçu ← faux                    // ad3
                sinon
                    si (m1<8) ou (m2<8)            // ad4
                        alors
                            reçu ← faux            // ad5
                        sinon
                            reçu ← vrai            // ad6
                    fsi
            fsi
            retourne reçu                          // ad7
        fin
lexique :
-   m1 : réel, moyenne premier module
-   m2 : réel, moyenne second module
-   moy : réel, moyenne générale
-   reçu : booléen, à vrai si l'étudiant a obtenu son diplôme
```

Exemple 6 (suite)

```
Algorithme
    début
        ne_m1 ← lire ( )                                                    // 1
        no_m1 ← lire ( )                                                    // 2
        ne_m2 ← lire ( )                                                    // 3
        no_m2 ← lire ( )                                                    // 4
        obtenu ← accorderDiplôme(calculerNoteModule(ne_m1,no_m1),
                                 calculerNoteModule(ne_m2,no_m2))           // 5
        si obtenu                                                           // 6
            alors écrire(« diplôme obtenu »)                                // 7
            sinon  écrire(« diplôme non accordé »)                          // 8
        fsi
    fin
lexique
-   ne_m1 : réel, note d'écrit du premier module
-   no_m1 : réel, note d'oral du premier module
-   ne_m2 : réel, note d'écrit du second module
-   no_m2 : réel, note d'oral du second module
-   obtenu : booléen, vrai si le diplôme est accordé
```

Exemple 6 (suite)

Schéma de l'évolution de l'état des variables instruction par instruction :

On suppose que les valeurs introduites par l'utilisateur sont : 12 7 10 8.

	algorithme principal					accorderDiplôme				calculerNoteModule			calculerMoyenne		
variables / instructions	ne_m1	no_m1	ne_m2	no_m2	obtenu	m1	m2	moy	reçu	n1	n2	note	n1	n2	moy
1,2,3,4	12	7	10	8											
5										12	7				
cnm1															
cnm2													7	12	
cm1															10.33
cm2,cnm2												10.33			ret.
cnm4,5						10.33						ret.			
5										10	8				
cnm1															
cnm2													8	10	
cm1															9.33
cm2,cnm2												9.33			ret.
cnm4,5							9.33					ret.			
ad1													9.33	10.33	
cm1															10
cm2,ad1								10							ret.
ad2															
ad4															
ad6									vrai						
ad7,5					vrai				ret.						
6															
7															

Algorithmique

Chaînes de caractères

Introduction

- Les caractères sont notés entre apostrophes
  - Exemples : ‘a’, ‘A’, ‘3’, ‘{‘ sont des caractères
- On utilise un seul jeu de caractères à la fois
  - Il contient tous les caractères utilisables (lettres, chiffres, ponctuation)
  - Chaque caractère est repéré par sa position dans le jeu de caractères
- Le type chaîne permet de décrire des objets formés par la juxtaposition de plusieurs caractères
  - Exemples : « bonjour », « en 2010, il fera beau »
- Dans la plupart des langages de programmation, il y a des outils pour manipuler les chaînes de caractères
- Nous allons présenter quelques fonctions pour manipuler les chaînes de caractères au niveau des algorithmes

Quelques fonctions

- **Concaténation de chaînes**
  - Syntaxe : fonction concat(ch1 : chaîne, ch2 : chaîne) : chaîne
  - But : elle retourne une chaîne formée par la fusion de ch1 et ch2. La chaîne résultat est formée par ch1 suivie de ch2.
  - Exemple : ch1 ← « Ali »
    ch2 ← « Baba »
    ch3 ← concat(ch1,ch2) ➔ « AliBaba »
- **Longueur d'une chaîne**
  - Syntaxe : fonction longueur(ch : chaîne) : entier
  - But : elle retourne la longueur de la chaîne ch (le nombre de caractères).
  - Exemple : ch ← « Ali »
    n ← longueur(ch) ➔ 3
    n ← longueur(«») ➔ 0

Quelques fonctions (suite)

- **Sous-chaîne**
  - Syntaxe : fonction sousChaîne(ch : chaîne, i : entier, l : entier) : chaîne
  - But : elle retourne une sous-chaîne de longueur l extraite de ch à partir de la position i .
  - Exemple : ch1 ← « informatique »
    ch2 ← sousChaîne(ch1,6,2) ➔ « ma »
- **Accès au $i^{ème}$ caractère**
  - Syntaxe : fonction ième(ch : chaîne, i : entier) : caractère
  - But : elle retourne le caractère qui se trouve à la position i. Il faut toujours satisfaire la condition suivante : i <= longueur(ch).
  - Exemple : ch ← « Bonjour »
    c1 ← ième(ch,2) ➔ 'o'

Quelques fonctions (suite)

- **Modification du $i^{ème}$ caractère**
  - Syntaxe : fonction remplace(ch InOut: chaîne, i : entier, c : caractère)
  - But : elle remplace le $i^{ème}$ caractère de la chaîne ch par le caractère c.
  - Exemple :

```
ch1 ← « informer »
c ← 'é'
remplace(ch1,2,c)
écrire(ch1)          ➔ « iéformer »
c ← 'd'
remplace(ch1,1,c)    ➔ « déformer »
```

Exemple

On donne un télégramme mot par mot. On souhaite compter le nombre d'unités de paiement du télégramme sachant qu'il se termine par le mot « stop », qu'un mot de longueur *l* coûte (l 10) + 1 unités et que le mot « stop » ne coûte rien.

Données :
la suite des mots composant le télégramme suivi de « stop »

Résultats :
nombre d'unités de paiement

Ecrire un algorithme qui permet de faire cette tâche

Exemple (suite)

algorithme
début
nup ← 0
mot ← lire ( )
tant que mot ≠ « stop » faire
prixmot ← (longueur (mot) ÷ 10) + 1

nup ← nup + prixmot
mot ← lire ( )
ftant
écrire (nup)
fin

lexique
- nup : entier, nombre d'unités de paiement
- mot : chaîne, ième mot du texte
- prixmot : entier, prix du ième mot du texte

Algorithmique

Tableaux

Notion de tableau

- Structure de données qui permet de rassembler un ensemble de valeurs de *même* type sous un *même* nom de variable en les différenciant par un *indice*
- Le type d'un tableau précise l'intervalle de définition et le type (commun) des éléments :

  Tab : tableau type_des_éléments [borne_inférieure .. borne_supérieure]
- Nous choisirons toujours la valeur 0 pour la borne inférieure dans le but de faciliter la traduction en C ou en Java. Par exemple, pour un tableau t de 10 entiers , on pourra écrire :

  t : tableau entier [0..9]

0	1	2	3	4	5	6	7	8	9
45	54	1	-56	22	134	49	12	90	-27

Notion de tableau (suite)

- Pour accéder à un élément du tableau, il suffit de préciser entre crochets l'indice de la case contenant cet élément :
  t[6] permet d'accéder à la valeur 49

0	1	2	3	4	5	6	7	8	9
45	54	1	-56	22	134	49	12	90	-27

- Affectation : x ← t[0]
- L'instruction t[6] ← 43 permet de modifier le tableau comme suit :

0	1	2	3	4	5	6	7	8	9
45	54	1	-56	22	134	43	12	90	-27

- Initialisation : t ← (11,2,19,25,9,0,54,33,6,11)

0	1	2	3	4	5	6	7	8	9
11	2	19	25	9	0	54	33	6	11

Parcours complet d'un tableau

- Les algorithmes utilisent des itérations permettant de faire un parcours complet ou partiel des différents éléments du tableau.
- Exemple : afficher un à un tous les éléments d'un tableau de n éléments

```
fonction écrireTableau(n : entier, tab : tableau entier [0..n-1])
        début
            pour i de 0 à n-1 faire                    // f1
                écrire(tab[i])                         // f2
            fpour
        fin
lexique
-   i : entier,  indice d'itération
-   n : entier, taille du tableau
-   tab : tableau entier [0..n-1]

Algorithme
    début
        n ← lire ( )                                   // 1
        tab ← lire ( )                                 // 2
        écrireTableau(n, tab)                          // 3
    fin
lexique
-   n : entier, taille du tableau
-   tab : tableau entier [0..n-1]
```

Parcours complet d'un tableau (suite)

- Schéma de l'évolution de l'état des variables instruction par instruction pour le tableau saisi, de taille 3, contient : 7 10 8.

	algorithme principal				fonction écrireTableau				
variables / instructions	n	tab			n	i	tab		
		tab[0]	tab[1]	tab[2]			tab[0]	tab[1]	tab[2]
1	3								
2		7	10	8					
3					3		7	10	8
f1						0			
f2							écrire		
f1						1			
f2								écrire	
f1						2			
f2									écrire
f1, 3						(fin)			

Parcours complet d'un tableau (suite)

- Exemple : Multiplier par 2 les éléments d'un tableau

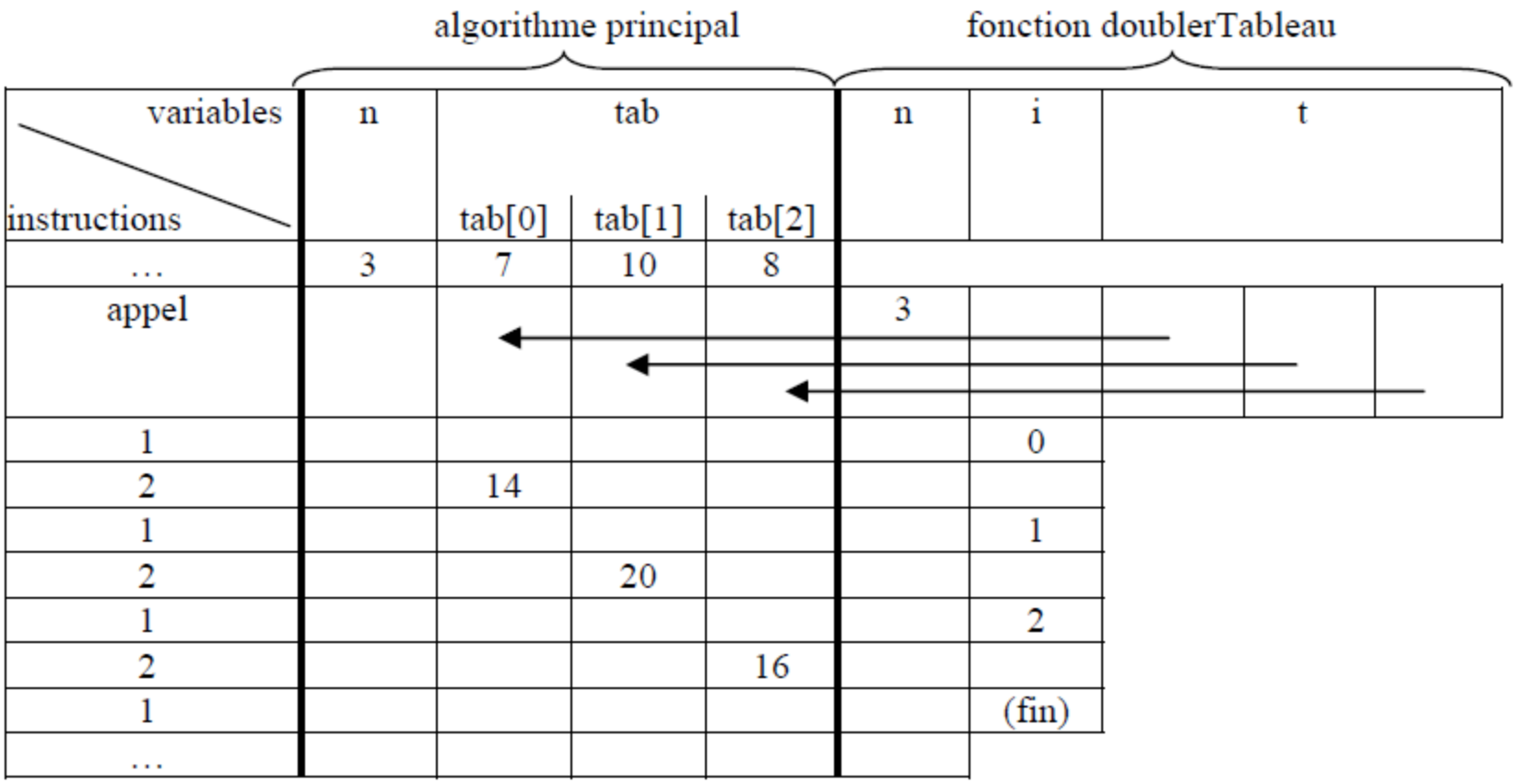
```
fonction doublerTableau(n : entier, t InOut : tableau entier [0..n-1])
        début
            pour i de 0 à n-1 faire                    // 1
                t[i] ← t[i]*2                          // 2
            fpour
        fin
```

lexique :

- n : entier, taille du tableau
- t : tableau entier [0..n-1], tableau modifiable

Parcours complet d'un tableau (suite)

- On suppose que l'appel de fonction est « doublerTableau(3, tab) », où tab est un tableau de taille 3 qui contient : 7 10 8.



Parcours partiel d'un tableau

- Certains algorithmes sur les tableaux se contentent de parcourir successivement les différents éléments du tableau jusqu'à rencontrer un élément satisfaisant une certaine condition. Un tel parcours partiel est le plus souvent basé sur une répétition conditionnelle

- Exemple :
  On cherche ici à savoir si un tableau saisi au clavier n'est constitué que d'entiers positifs

Parcours partiel d'un tableau

- Exemple : Savoir si un tableau lu contient que des entiers positifs

```
algorithme
        début
            n ← lire()
            tab ← lire()
            i ← 0
            positif ← vrai
            tant que positif et i < n faire
                si tab[i] < 0
                    alors positif ← faux
                fsi
                i ← i + 1
            ftant
            si positif
                alors écrire(« tableau d'entiers naturels »)
                sinon écrire(« tableau d'entiers relatifs »)
            fsi
        fin
lexique
-   i : entier,  indice d'itération
-   n : entier, taille du tableau
-   tab : tableau entier [0..n-1]
-   positif : booléen, vrai si aucun entier négatif n'a été détecté
```

Parcours imbriqués

- Certains algorithmes sur les tableaux font appel à des boucles imbriquées : la boucle principale sert généralement à parcourir les cases une par une, tandis que le traitement de chaque case dépend du parcours simple d'une partie du tableau (par exemple toutes les cases restantes), ce qui correspond à la boucle interne.

- Exemple : La fonction suivante calcule, pour chaque case d'un tableau, le nombre de cases suivantes qui contiennent un élément strictement supérieur. Les résultats sont placés dans un tableau.

Si t est le tableau suivant :

9	4	6	2	8	5
0	*1*	*2*	*3*	*4*	*5*

Alors le tableau résultat res est :

0	3	1	2	0	0
0	*1*	*2*	*3*	*4*	*5*

Parcours imbriqués (suite)

```
fonction calculerNbSuccesseursSup(n : entier, t : tableau entier [0..n-1]) : tableau entier [0..n-1]
        début
            pour i de 0 à n-1 faire
                tres[i] ← 0
            fpour
            pour i de 0 à n-2 faire
                pour j de i+1 à n-1 faire
                    si t[i]<t[j]
                        alors
                            tres[i] ← tres[i]+1
                    fsi
                fpour
            fpour
            retourne tres
        fin
```

lexique :

- n : entier, taille du tableau
- t : tableau entier [0..n-1],
- tres : tableau entier [0..n-1], tableau résultat (case i contient le nombre de cases de t d'indice supérieur à i qui contiennent un élément supérieur à t[i])
- i : entier, indice de la boucle principale (parcours pour remplir tres)
- j : entier, indice de la boucle interne (parcours des cases restantes de t)

Tableaux multidimensionnels

- Les cases d'un tableau à une dimension sont indicées de manière consécutive (cases « alignées »). Il est possible de disposer ces cases selon des grilles (tableaux à deux dimensions), des cubes (tableaux à trois dimensions), …
- Les algorithmes les plus simples sur ces tableaux utilisent en général des boucles imbriquées : chaque niveau de boucle correspond au parcours selon une dimension.
- Le type d'un tableau précise l'intervalle de définition selon chaque dimension :

tableau type_des_éléments [borne_inf_dim1 .. borne_sup_dim1, borne_inf_dim2 .. borne_sup_dim2, …]

Tableaux à deux dimensions

- Ces tableaux sont faciles à se représenter comme une grille ayant un certain nombre de lignes (première dimension) et un certain nombre de colonnes (seconde dimension) :

  tableau type_des_éléments [0 .. nb_lignes-1, 0 .. nb_colonnes-1]

- Exemple : Un tableau t avec 5 colonnes et 3 lignes

	0	1	2	3	4
0	45	54	1	-56	22
1	64	8	54	34	2
2	56	23	-47	0	12

- Pour accéder à un élément du tableau, il suffit de préciser entre crochets l'indice de la case contenant cet élément, et ce pour chacune des dimensions. Par exemple, pour accéder à l'élément 23 du tableau d'entiers ci-dessus, on écrit : t[2,1].

Tableaux à deux dimensions (suite)

- L'instruction suivante affecte à la variable x la valeur du premier élément du tableau, c'est à dire 45 :

$$x \leftarrow t[0,0]$$

- L'élément désigné du tableau peut alors être utilisé comme n'importe quelle variable :

	0	1	2	3	4
0	45	54	1	-56	22
1	64	8	54	34	2
2	56	23	-47	0	12

$$t[2,1] \leftarrow 43$$

45	54	1	-56	22
64	8	54	34	2
56	43	-47	0	12

Tableaux à deux dimensions (suite)

- Exemple : La fonction suivante calcule la somme des éléments d'un tableau à 2 dimensions.

```
fonction somme(li : entier, co : entier, t : tableau entier [0..li-1,0..co-1]) : entier
        début
            s ← 0
            pour i de 0 à li-1 faire
                pour j de 0 à co-1 faire
                    s ← s + t[i,j]
                fpour
            fpour
            retourne s
        fin
lexique :
-   li : entier, nombre de lignes du tableau
-   co : entier, nombre de colonnes du tableau
-   t : tableau entier [0..li-1,0..co-1], tableau dont on cherche la somme
-   s : entier, somme des éléments déjà parcourus
-   i : entier, indice d'itération sur les lignes
-   j : entier, indice d'itération sur les colonnes
```

Recherche dichotomique

- La fonction *rechercheDicho* recherche un élément dans un tableau trié et retourne l'indice d'une occurrence de cet élément (ou –1 en cas d'échec). Une telle recherche peut être réalisée de manière séquentielle ou dichotomique. La recherche dichotomique est la plus efficace en temps d'exécution.
- Principe : on compare l'élément cherché à celui qui se trouve au milieu du tableau. Si l'élément cherché est plus petit, on continue la recherche dans la première moitié du tableau sinon dans la seconde. On recommence ce processus sur la moitié. On s'arrête lorsqu'on a trouvé ou lorsque l'intervalle de recherche est nul.
- Exemple : Recherchons l'entier 46 puis 10 dans le tableau d'entiers suivant défini sur l'intervalle [3. .10] :

5	13	18	23	46	53	89	97
3	4	5	6	7	8	9	10

Recherche dichotomique (suite)

```
fonction rechercheDicho(e : entier, n : entier, t : tableau entier [0..n-1]) : entier
    début
        trouve ← faux
        debut ← 0
        fin ← n-1
        tant que debut≤fin et non trouve faire
            i ← (debut+fin)÷2
            si t[i] = e
                alors trouve ← vrai
                sinon
                    si t[i] > e
                        alors fin ← i-1
                        sinon debut ← i+1
                    fsi
            fsi
        ftant
        si trouve
            alors indice ← i
            sinon indice ← -1
        fsi
        retourne indice
    fin
```

lexique :

- n : entier, taille du tableau
- t : tableau entier [0..n-1], tableau trié par ordre croissant
- e : entier, élément recherché
- trouve : booléen, faux tant que l'élément cherché n'est pas trouvé
- i : entier, indice de la case observée (case du milieu)
- debut: entier, indice minimum de la zone de recherche
- fin: entier, indice maximum de la zone de recherche
- indice : entier, indice de l'élément recherché ou -1

Algorithmique

Variables Composites

Définition

- On appelle *type composite* un type qui sert à regrouper les caractéristiques (éventuellement de types différents) d'une valeur complexe.
- Chaque constituant est appelé *champ* et est désigné par un *identificateur de champ*. Le type composite est parfois aussi appelé *structure* ou *produit cartésien.*
- On utilise chaque identificateur de champ comme sélecteur du champ correspondant.
- Définition lexicale : c : <chp1 : type1, ... , chpn : type n>

  où c est une variable composite dont le type est décrit,
  chp1, chp2, ..., chpn sont les noms des champs de la variable composite c, type1 est le type du champ chp1,..., typen le type du champ chpn.

Présentation du type composite

- Exemples :
  - date: <jour: entier, mois: chaîne, année: entier >
  - personne: <nom: chaîne, prénom: chaîne, datenaiss: <jour: entier, mois: chaîne, année: entier>, lieunaiss : chaîne>
- Sélection du champ chpi de c : c.chpi
  c.chpi est une variable de type typei et peut être utilisé comme toute variable de ce type. On utilise chaque identificateur de champ comme sélecteur du champ correspondant.
- Exemples :
  - *date* désigne une variable de type composite <jour: entier, mois: chaîne, année: entier >.
  - *date.jour* désigne une variable de type entier.
  - *personne.datenaiss.année* est aussi une variable de type entier.

Présentation du type composite (suite)

- **Création d'une variable composite par la liste des valeurs des champs**
  c ← (c1, c2, ..., cn) où *c1 : type1 , c2 : type2, ..., cn : typen.*
  La liste des valeurs des champs est donnée dans l'ordre de la description du type composite.
- Exemples :
  Soit les deux types composites :
  Date = <jour : entier, mois: chaîne, année: entier>
  Personne = <nom: chaîne, prénom: chaîne, datenaiss: Date, lieunaiss: chaîne> et les variables :
  père, fils : Personne ,
  date : Date.
  On peut écrire :
  date ← (1, "janvier", 1995)
  fils ← (père.nom, "Paul", (14, "mars", 1975), "Lyon")

Présentation du type composite (suite)

- **Modification de la valeur d'un champ**

  c.chpi ← ci où *ci* est de type *typei*, la valeur *ci* est affectée au champ *chpi* de *c*.

  Exemple :

  date.jour ← 30

- **Autres « opérations »**

  c ← lire( )

  écrire(c)

  c1 ← c2 où c1 et c2 sont deux variables composites du même type.

Exemple

Dans le service qui affecte les nouveaux numéros INSEE, on veut écrire l'algorithme de la fonction qui, à partir des renseignements nécessaires sur une personne donnée et du dernier numéro personnel affecté par ce service, crée un nouveau numéro INSEE.
Les renseignements sur la personne sont fournis sous la forme d'une variable composite formée de:

- nom
- prénom
- date de naissance sous la forme d'un triplet d'entiers (jour, mois, année)
- code de la ville de naissance
- numéro du département de naissance
- sexe : 'M' pour masculin, 'F' pour féminin

Le numéro d'INSEE sera de type composite, formé de :

- code: 1 pour masculin , 2 pour féminin
- année de naissance (seulement les 2 derniers chiffres)
- mois de naissance sous la forme d'un entier
- numéro du département de naissance
- code de la ville de naissance
- numéro personnel

Le numéro personnel est obtenu en ajoutant 1 au dernier numéro personnel affecté par le service.

Algorithme

```
algorithme
fonction inseeCreer (personne : EtatCivil, dernier_numéro : entier) :Insee
        début
        si personne.sexe = 'M'
            alors  code ←1
            sinon  code ← 2
        fsi
        numéro_insee ←  (code, personne. datenaiss.année mod 100, personne.datenaiss.mois ,
                          personne.depnaiss, personne.villenaiss , dernier_numéro + 1)
        retourne numéro_insee
        fin
```

lexique

- Etatcivil = <nom: chaîne, prénom: chaîne, datenaiss: Date, depnaiss : entier, villenaiss : entier, sexe : caractère >
- Date = <jour : entier, mois: entier, année: entier>
- personne : Etatcivil, personne qui souhaite un numéro INSEE
- dernier_numéro : entier, dernier numéro personnel affecté
- code : entier, code pour représenter le sexe
- Insee = < codesexe : entier, annaiss : entier, moisnaiss : entier, depnaiss : entier, villenaiss : entier, numéro : entier>
- numéro_insee : Insee, numéro INSEE affecté.