



Initiation à l'algorithmique



Présentation

L'algorithmique vous permet de comprendre la logique de l'ordinateur, de développer ensuite un programme en informatique quelque soit le langage de programmation. Dans ce cours, nous abordons toutes les notions de base qui sont fondamentales en algorithmique. Nous voyons la lecture, l'affichage, nous travaillons les variables, les constantes, la structure itérative, la structure de choix, la structure alternative pour pouvoir aligner des instructions les unes après les autres. Nous mettons en pratique toutes ces notions de base dans des exemples dans des exercices.



Objectif

A la fin de ce cours, vous aurez acquis toutes les connaissances de base qui vont vous permettre d'aborder des notions plus complexes.



Plan du cours

Chapitre 1 : Instructions de base

Chapitre 2 : Les variables

Chapitre 3 : Les affectations

Chapitre 4 : Les structures alternatives

Chapitre 5 : Exercices d'algorithmique

Chapitre 6 : Structure itérative

Chapitre 7 : La boucle (pour ... FinPour)

Chapitre 8 : Les structures de choix

Chapitre 9 : Procédures et fonctions



Plan du cours

Chapitre 10 : Structures de données : les tableaux



Chapitre I : Instructions de base



1.1 Présentation de la structure du cours

Bienvenue en algorithmique !

Un algo, qu'est-ce-que c'est ? C'est une suite d'instructions alignées les unes après les autres dans une certaine logique qui vont nous permettre d'obtenir un résultat que l'on souhaite atteindre. L'algorithme n'est pas écrit en langage de programmation. C'est écrit dans un langage qui va être compréhensible par tout le monde. Il en existe quelques conventions d'écriture mais ceci dit nous pouvons comprendre un algo même si nous ne sommes pas en informatique.



1.1 Présentation de la structure du cours

L'algo est fondamental. pourquoi ? Parce que lorsque nous développons des programmes informatiques, nous reprenons la logique de cet algo et nous la traduisons dans un langage de programmation quel qu'il soit. Quand on écrit un algo, on l'écrit avec un langage humain et ensuite traduit dans un langage de programmation (langage machine). Donc, quand nous allons dérouler tous nos algo dans ce cours, nous allons penser d'abord développeur (on voit le déroulement de l'algo), rénovateur (savoir comment ça se passe dans la machine) et enfin utilisateur (voir comment se concrétise à l'écran sur un ordinateur)



1.1 Présentation de la structure du cours

Dans un algo on retrouve différentes parties. Un algo commence toujours par un début et se termine par une fin, toutes instructions sont alignées entre le début et la fin. Cela signifie qu'on part du principe qu'à partir du début la machine va exécuter les instructions les unes après les autres déroulées jusqu'à la fin. Un algo va nous permettre d'automatiser certaines tâches qui reviennent sans cesse. Le gros avantage de l'informatique, c'est de pouvoir répéter plusieurs fois des traitements récurrents. Vu que la machine ne peut pas se passer de nous, alors c'est nous qui lui dirons que faire afin d'aboutir à un résultat escompté.

1.2 Qu'est-ce qu'un algorithme ?

Prenons l'exemple du réveil matinal :

Je me lève

Je regarde par la fenêtre

Si le soleil brille alors
 je me lève

Sinon

je retourne au lit

FinSi

Là en quelque sorte, on vient d'écrire un algo. Ce sont des instructions alignées les unes après les autres. Une ligne correspond à une instruction.

1.3 Exemple d'algorithme de multiplication

Prenons l'exemple de : $4 \times 5 = 20$

Ici notre cerveau connaît déjà ce résultat, mais en algo nous procédons autrement afin que chaque nombre se voit attribué un nom ainsi que le résultat.

$$\begin{array}{ccccc} 4 & \times & 5 & = & 20 \\ \downarrow & & \downarrow & & \downarrow \\ \text{Nbre1} & & \text{Nbre2} & & \text{Rslt} \end{array}$$

Algo Multiplication

Début

saisir Nbre1 :

saisir Nbre2 :

Rslt := Nbre1 x Nbre2

Fin

1.4 Structure d'un algorithme

ALGO1

Nom/titre de l'algo

Const N : entier N := 3

Var SOMME, A : entier

Déclaration des variables,
constantes

Début

Ouverture de l'algo

saisir A

SOMME := A + N

afficher SOMME

Les instructions

Fin

Fermeture de l'algo



Chapitre II : Les variables

2.1 Introduction aux variables

Une variable est un emplacement mémoire représenté comme une boîte initialement vide et nommée « Var » en algo. Elle a un nom ex. NomVar. Il en est de même pour une constante « Const », ex. NomConst. En tant que développeur, lorsque nous déclarons une variable ou une constante, nous réservons un emplacement mémoire dans la machine c'est-à-dire dans une boîte qui porte le nom de la variable ou la constante. Cette boîte attend qu'on lui affecte une valeur.

Ex. Var A, B {Nous déclarons deux variables A et B}

Const C {Nous déclarons une constante C}

2.2 Les types de variables

ENTIER

RÉEL

CHAÎNE DE CARACTÈRES (suite de caractères)

BOOLEEN (deux états : soit oui/non soit 0/1 soit vrai/faux)

CARACTÈRE (une lettre, un caractère)

La première chose que l'on fait lorsqu'on apprend un langage de programmation, on regarde quels sont les types possible dans ce langage et selon le type l'emplacement mémoire sera plus ou moins important. **L'algo ne gère pas l'emplacement mémoire.**

2.3 Différence entre constance et variable

Variable

- Nom servant à repérer un emplacement mémoire;
- Elle est utilisée tout le long du programme;
- Elle **PEUT CHANGER** de valeur dans le programme.

Constante

- Nom servant à repérer un emplacement mémoire;
- Elle est utilisée tout le long du programme;
- Elle **NE PEUT CHANGER** de valeur dans le programme.



Chapitre III : Les affectations

3.1 Présentation des affectations

$A \longleftarrow 5$ OU $A := 5$

Se lit « affecter à A la valeur de 5 »

Un développeur peut laisser librement l'utilisateur lui-même saisir sa valeur.

Var A : entier

Début

saisir A

Prendre la valeur saisie par l'utilisateur et l' affecter à la variable A »

3.1 Présentation des affectations

Après avoir affecté une valeur à une variable, il est important aussi de savoir comment afficher cette valeur à l'écran.

Var A : entier

Début

saisir A

afficher A

Se lit « prendre la valeur de la variable A et l'afficher »

3.2 Exemple d'affectation

ALGO : NOTRE_PREMIER_ALGO

Var A, B, RESULTAT : entier

Début

 afficher "Veuillez saisir une première valeur "

 saisir A

 afficher "Veuillez saisir une deuxième valeur "

 saisir B

 RESULTAT := A + B

 afficher "La somme de ces deux valeurs est : ", RESULTAT

Fin

3.2 Exemple d'affectation

- ❑ La **VARIABLE** doit être déclarée (nom et type) avant le début de l'algorithme;
- ❑ La **CONSTANTE** est déclarée et initialisée avant le début de l'algorithme;

Elle ne peut pas changer de valeur dans le corps de l'algorithme

- ❑ **L'AFFECTATION** d'une variable écrase l'ancienne valeur de cette variable.

3.3 Permutation d'affectation

Petit exercice : A vaut 5 et B vaut 22, alors nous souhaitons permuter les valeur de A et B.

Si nous affectons la valeur A dans B : $A := B$ on écrase la valeur A.

Solution : Alors pour éviter de perdre nos valeurs, on va devoir faire intervenir une troisième variable.

$C := A$

$A := B$

$B := C$



Chapitre IV : Les structures alternatives

4.1 Présentation des structures alternatives

ALGO1

! Affichage du plus grand nombre

Var X, Y : entier

Début

 afficher "Veuillez saisir un premier nombre : "

 saisir X

 afficher "Veuillez saisir un deuxième nombre : "

 saisir Y

 si $X > Y$ alors

 afficher "Le plus grand nombre est : ", X

 sinon

 afficher "Le plus grand nombre est : ", Y

 FinSi

Fin

4.2 Opérateurs de comparaison

Il s'agit de tester une variable à l'aide de symboles de comparaison qui sont les suivants :

Symbole	Signification
=	Égale à
<	Inférieur à
>	Supérieur à
<=	Inférieur ou égal à
>=	Supérieur ou égal à
<>	Différent de

4.2 Opérateurs de comparaison

Résultat de notre algo de départ (cf. slide 23) :

Veillez saisir un premier nombre

7

Veillez saisir un deuxième nombre

6

Le plus grand nombre est : 7

L'instruction SAISIR permet de stocker les valeurs saisies par l'utilisateur : 7 dans la variable X et 6 dans la variable Y.

L'instruction AFFICHER permet d'écrire à l'écran ce qu'il y a entre " ".

Le test effectué $X > Y$? Soit $7 > 6$? Est vrai donc l'instruction exécutée est celle qui se trouve après le ALORS

4.3 Condition fausse

Résultat de notre algo de départ (cf. slide 23) :

Veillez saisir un premier nombre

8

Veillez saisir un deuxième nombre

9

Le plus grand nombre est : 9

Le test effectué $X > Y$? Soit $8 > 9$? Est faux donc l'instruction exécutée est celle qui se trouve après le SINON

4.4 Exemple de structure appauvrie

ALGO2

! Affichage du nombre saisi si celui-ci est supérieur à 10

Var X, Y : entier

Const : C : entier

C := 10

Début

afficher "Veuillez saisir un nombre : "

saisir X

si $X > C$ alors

afficher "Le nombre saisi est : ", X

FinSi

afficher "Ceci est la fin de notre programme. "

Fin

4.5 Les conditions complexes- opérateur (ET)

ALGO3

! Affichage si le nombre saisi est compris ou non entre 20 et 30

Const : C1, C2 : entier

 C1 := 20

 C2 := 30

Var X : entier

Début

 afficher "Veuillez saisir un nombre : "

 saisir X

 si $X > C1$ ET $X < C2$ alors

 afficher "Le nombre saisi est compris entre : ", C1, " et ", C2

 sinon

 afficher "Le nombre saisi n'est pas compris entre : ", C1, " et ", C2

Fin

4.6 Les conditions complexes- opérateur (OU)

ALGO4

! Affichage si le nombre saisi est égal à 15 ou 20

Const : C1, C2 : entier

 C1 := 15

 C2 := 20

Var VAR1 : entier

Début

 afficher "Veuillez saisir un nombre : "

 saisir VAR1

 si VAR1 = C1 OU VAR1 = C2 alors

 afficher "Le nombre saisi est : ", VAR1

 sinon

 afficher "Le nombre saisi n'est pas égal à ", C1, " ni à ", C2

Fin

4.7 Synthèse des conditions

- Le « **SI** » doit être suivi obligatoirement du « **ALORS** »
- L'instruction qui suit le « **ALORS** » est exécutée quand le test est vrai;
- Avec l'opération logique **ET** : l'instruction qui suit le « **ALORS** » est exécutée seulement si tous les tests sont vrais;
- Avec l'opération logique **OU** : l'instruction qui suit le « **ALORS** » est exécutée dès que l'un des tests est vrai.
- Il est possible d'écrire plusieurs instructions après les « **ALORS** » et « **SINON** »



Chapitre V : Exercices



5.1 Exercice 1

Écrire un algorithme qui lit deux nombres et une lettre. Si la lettre est S (comme somme), il calcule et écrit la somme des deux nombres. Dans le cas contraire, il calcule et écrit le produit.



5.2 Exercice 2

Écrire un algorithme qui lit un caractère écrit s'il est placé avant ou après la lettre « m » dans l'ordre alphabétique.



5.3 Exercice 3

Écrire un algorithme qui lit un caractère écrit s'il est placé avant ou après la lettre « m » dans l'ordre alphabétique.



5.4 Exercice 4

Écrire un algorithme qui lit deux caractères et écrit s'ils sont ou non rangés dans l'ordre alphabétique.

5.5 Exercice 5

Écrire un algorithme qui lit un nombre et dit s'il est ou non compris entre 10 (inclus) et 20 (inclus).

- Écrire que le nombre est dans la fourchette indiquée ou
- Écrire que le nombre n'est pas dans la fourchette indiquée



5.6 Exercice 6

Écrire un algorithme qui lit trois nombres et écrit s'ils sont ou non rangés par ordre croissant.

5.7 Exercice 7

Écrire un algorithme qui affiche soit :

- Le plus grand des deux nombres X et Y ;
- Les deux nombres sont égaux.

Pour cela, utiliser une variable booléenne et séparer la partie traitement de la partie édition.



5.9 Exercice 9

Écrire un algorithme qui affiche trois nombres par ordre croissant en permutant leurs valeurs de façon à obtenir le plus petit nombre dans V1 et le plus grand dans V3

5.10 Exercice 10

Écrire un algorithme pour le **CALCUL D'UNE REMISE**

A partir d'un montant lu en données, déterminer un montant net en sachant que l'on accorde :

- Une remise de 5 % si le montant est compris entre 2000 FCFA et 5000 FCFA (ces valeurs comprises),
- Une remise de 10% si le montant est supérieur à 5000 FCFA.



Chapitre VI : Structures itératives

6.1 Présentation des structures itératives

Une structure itérative aussi appelée boucle consiste à répéter n fois un traitement en informatique.

TANT QUE *<expr. Logique>* FAIRE
 BLOC

REFAIRE OU FinTantQue

Tester d'abord *<expr. Logique>*. Si elle est vraie, Exécuter les instructions du bloc et boucler de nouveau sur le bloc. Si *<expr. Logique>* est fausse, la boucle se termine (Bloc d'instructions n'est pas exécuté)

6.1 Présentation des structures itératives

Exemple :

Var A, B : ENTIER

DEBUT

 TANT QUE $A < 0$ ou $B < 0$ FAIRE

 afficher " entrer 2 nombres entiers positifs "

 saisir A, B

 REFAIRE

FIN

6.1 Présentation des structures itératives

REPETER

BLOC

JUSQU'A <expr. Logique>

La condition d'arrêt est évaluée après la boucle donc le traitement est effectué au moins une fois.

Exemple :

Var A, B : ENTIER

DEBUT

REPETER

afficher " saisir deux nombres entiers positifs "

saisir A, B

JUSQU'A $A < 0$ ou $B < 0$

FIN



6.2 Exercice 11

Problème :

Effectuer une multiplication de 2 entiers positifs en n'utilisant que l'addition.

6.3 Solution Exercice 11

Algo : MultiAddition

Var nbre1, nbre2, P : entier

Début

 P := 0

 saisir nbre1, nbre2

 Tant que nbre2 \neq 0

 P := P + nbre1

 nbre2 := nbre2 - 1

 Refaire

 afficher " Le produit de ces deux nombres est égal à ", P

Fin



6.4 Exercice 12

Problème :

Si, malgré le message prévu (exercice 1), l'utilisateur saisit un ou deux nombres négatifs. Que se passe t-il ?

6.5 Solution Exercice 12

ALGO : OPE

Var X, Y, RES : ENTIER

Var OPERATION : caractère

DEBUT

saisir X, Y, OPERATION

 Tant que $X < 0$ ou $Y < 0$ FAIRE

 Afficher " Veuillez saisir valeurs possibles "

 saisir X, Y

 refaire

Si OPERATION = « s » alors

 RES := $A + B$

 Afficher " somme ", RES

Sinon RES := $A * B$

 Afficher « produit », RES

FinSi

FIN



6.6 Exercice 13

Problème :

Écrire un algorithme permettant de simuler une facturation.

6.7 Variable et valeurs saisies

Lire un nombre saisi au clavier par l'utilisateur et poser la question suivante jusqu'à ce que la réponse convienne.

Algorithme :

ALGO REP_CORRECTE

Var nbre : numérique

Répéter

 Afficher "donnez un nombre inférieur à 100 "

 saisir nbre

Jusqu'à $\text{nbre} < 100$

Afficher "OK"

Fin

6.7 Variable et valeurs saisies

RÉSULTAT :

Donnez un nombre inférieur à 100 :

248

Donnez un nombre inférieur à 100 :

103

Donnez un nombre inférieur à 100 :

83

OK

6.7 Variable et valeurs saisies

Améliorez le programme pour que l'exécution se présente ainsi :

Donnez un nombre inférieur à 100 :

248

SVP inférieur à 100

103

SVP inférieur à 100

83

OK

La solution est possible avec une boucle tant que.

Allez on réfléchit un peu. !!!

6.8 Explication du compteur

Exemple :

Var a, b, c, n : réel

Début

n := 0

Répéter

saisir a

c := a * a

afficher c

n := n + 1

jusqu'à a = 0

Afficher "Vous avez donné " , n, " valeurs "

Fin

Exécution du programme :

2

4

5

25

0

0

Vous avez donné 3 valeurs

 n : Compteur



6.9 Exercice 14

Écrire un algorithme qui lit des notes saisies au clavier et donne le pourcentage de notes supérieures à la moyenne 10.



Chapitre VII : La boucle (pour ... finpour)

7.1 Présentation de la boucle (pour ...FinPour)

Jusque là nous avons vu « RÉPÉTER ... JUSQU'A », « TANTQUE ... REFAIRE /FINTANTQUE ». A présent nous abordons une autre structure répétitive qui est la boucle « POUR ... FINPOUR », cette boucle la plus utilisée dans tous les langages de programmation. C'est la « *for i i++* »

Cette boucle permet de recommencer un nombre de fois décidé à l'avance les instructions écrites entre les lignes Pour...allant de...à ... et FinPour

7.2 Mise en pratique de la boucle (pour ... FinPour)

Exemple 1 : *Répéter 5 fois une instruction identique.*

L'algorithme suivant affiche 5 fois de suite le message

"Bonjour !"

ALGO BONJOUR

Var n : entier

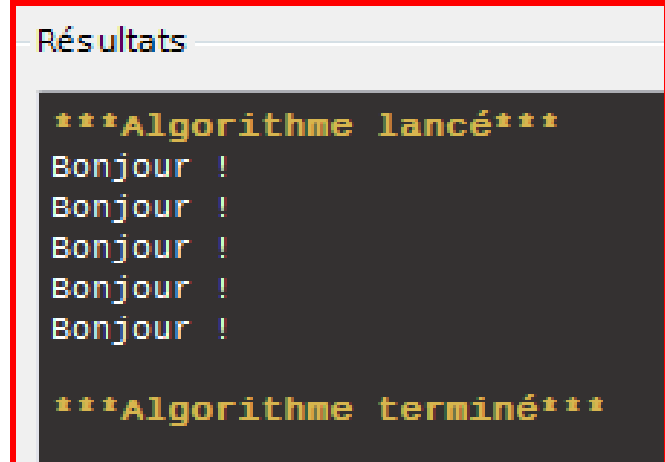
Début

Pour n := 1 A 5

Afficher "Bonjour !"

FinPour

Fin



```
Résultats

***Algorithme lancé***
Bonjour !
Bonjour !
Bonjour !
Bonjour !
Bonjour !

***Algorithme terminé***
```

7.2 Mise en pratique de la boucle (pour ... FinPour)

Exemple 2 : Répéter une instruction , avec des variantes.
L'instruction à répéter, peut dépendre de la valeur de n , comme dans l'exemple suivant :
La première fois que la ligne « POUR » est lu, n prend la valeur 1, et chaque fois que cette ligne est à nouveau lue, la valeur de n augmente automatiquement de 1.

ALGO BONJOUR

Var n : entier

Début

Pour $n := 1$ A 5

Afficher n

FinPour

Fin



Chapitre VIII : Structure de choix

8.1 Présentation de la structure de choix

La structure **SELON** permet d'effectuer tel ou tel traitement en fonction de la valeur des conditions 1 ou 2 ou ...n.

Syntaxe :

SELON <Variable>

<condition 1> : <action 1>

<condition 2> : <action 2>

...

<condition n> : <action n>

SINON : <action sinon>

FINSELON

8.1 Présentation de la structure de choix

Exemple :

Selon moyenne

Cas ≥ 0 et ≤ 5 : mention := "Médiorce"

Cas > 5 et ≤ 9 : mention := "Insuffisant"

Cas ≥ 10 et ≤ 12 : mention := "Assez-Bien"

Cas > 12 et ≤ 14 : mention := " Bien"

Cas > 14 et ≤ 20 : mention := " Très-Bien"

cas Sinon : mention := "Pas de mention"

FinSelon

Afficher mention

8.1 Présentation de la structure de choix

Remarque :

NB :

En programmation, cette structure peut exister mais avec une forme ou un fonctionnement éventuellement différent. Si elle n'existe pas, il faut se souvenir que, en fait, **SELONQUE** est un raccourci d'écriture pour des **SI** imbriqués.



Chapitre IX : Procédures et fonctions



9.1 Introduction

Lorsque l'on développe un programme et que le problème à résoudre est complexe, le nombre d'instruction devient vite important. Il est nécessaire de l'organiser (Modularité). Il suffit de regrouper sous un même nom les instructions agissant dans le même but.



9.1 Introduction

On distingue :

- ❑ Les **procédures** qui réalisent un traitement(lecture d'un complexe, tri du fichier étudiant)
- ❑ Les **fonctions** qui effectuent un calcul et **retournent un résultat**

Les fonctions et les procédures sont des portions de code réutilisable partout dans le programme. Ils sont parfois appelées des **sous-programmes**.

9.2 Syntaxe d'une fonction

FONCTION <nom_fonction> (<liste des paramètres>) : <type
de résultat>

<déclaration des objets locaux à la fonction>

DEBUT

{ corps de la fonction }

RETOURNER(résultat)

FIN

9.3 Exemple de fonction

FONCTION perimetre_rectangle (largeur, longueur :

ENTIER) : ENTIER

DEBUT

RETOURNER($2 * (largeur + longueur)$)

FIN

9.4 Syntaxe de procédure

PROCEDURE <nom_procedure>(<liste des paramètres>)

<déclaration des objets locaux de la procédure>

DEBUT

{corps de la procédure}

FIN

9.5 Syntaxe de procédure

PROCEDURE <nom_procedure>(<liste des paramètres>)

<déclaration des objets locaux de la procédure>

DEBUT

{corps de la procédure}

FIN

9.6 Exemple de procédure

PROCEDURE Echange(Var A, B : réel) Nom de la procédure
Var AUX : réel Variable locale
DEBUT
 AUX := A
 A := B
 B := AUX
 afficher " Les valeurs de ", A, " et de ", B, " ont été échangées"
FIN

9.7 Appel de fonction

FONCTION max3 (x, y, z : entier) : entier

Var max : entier

DEBUT

 SI (x < y) ALORS

 SI (z < y) ALORS

 max := y

 SINON max := z

 FINSI

 SINON

 SI (x < z) ALORS

 max := z

 SINON max := x

 FINSI

 FINSI

 RETOURNER (max)

FIN

9.7 Appel de fonction dans procédure

Maintenant, on peut créer la procédure qui détermine le maximum et le minimum de trois entiers, en faisant appel à la fonction max3.

```
PROCEDURE maxETmin(a,b,c : entier)
Var min, max : entier
DEBUT
    max := max3(a, b, c)
    min := -max3(-a, -b, -c)
    afficher " Le minimum de ", a, b, c, " est : ", min, " le maximum est ", max
FIN
```

9.8 Remarque

x, y, z sont les paramètres formels de la fonction $\text{max3}(x,y,z)$.

Ce sont des paramètres d'entrées : lors de l'appel de la fonction max3 , les **valeurs des arguments** d'appel (ici : a, b, c) ou $(-a, -b, -c)$ **sont transmises** aux paramètres x, y, z en fonction de leur ordre.

Les arguments sont des expressions (par exemple : $\text{max} := \text{max3}(2*a+b, c-b, a*c)$) qui sont évaluées à l'appel. Leur valeur est transmise aux paramètres.

Naturellement, le type des expressions doit être compatible avec le type des paramètres.

9.9 Mode de transmission des paramètres

Définition

Il existe deux modes de transmission des paramètres :

- Par **valeur** : le *paramètre transmis n'est jamais affecté* par les modifications dans la procédure ou la fonction (on ne récupère pas les résultats)
- Par **adresse (référence)** : le *paramètre transmis dans ce cas peut être modifié* et on récupère les résultats.

9.9 Mode de transmission des paramètres

Transmission par **valeur**

PROCEDURE PERMUT1(Var x, y : entier)

Var tmp := entier

Début

 tmp := x

 x := y

 y := tmp

Fin

9.9 Mode de transmission des paramètres

Transmission par **adresse**

PROCEDURE PERMUT2(Var x, y : entier)

Var tmp := entier

Début

 tmp := x

 x := y

 y := tmp

Fin

9.9 Mode de transmission des paramètres

ALGO TEST

Var a, b := entier

Début

 a := 10

 b := 20

 PERMUT1(a, b)

 afficher a, b ! Le résultat est (a, b) = (10, 20) : pas de permutation

 PERMUT2(a, b) ! Le résultat est (a, b) = (20, 10) : il y a permutation

Fin



9.10 Récursivité

Définition

La **récursivité** consiste à remplacer une boucle par un **appel** à la **fonction elle-même**.

9.10 Récursivité

Exemple :

Considérons la suite factorielle, elle est définie par :

$$0! = 1$$

$$n! = n(n-1)!$$

La fonction peut s'écrire simplement :

FONCTION factorielle(n : entier) : entier

Début

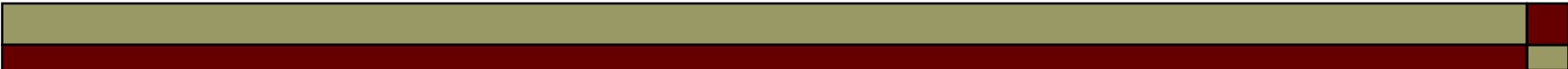
SI (n = 0) ALORS
retourner(1)

SINON
retourner(factorielle(n-1) * n)

FINSI

FIN

Appel de la fonction elle-même



Chapitre X : Structures de données : les tableaux



10.1 Définitions

Un **tableau** est un **ensemble de même type** indicé par un ensemble non vide d'indices, permettant un accès direct à chacun des objets.

La contrepartie de cette possibilité d'accès direct est que le tableau doit être **contigu en mémoire**: L'adresse d'un objet peut alors facilement être calculée à partir de l'adresse de départ du tableau, de l'indice de l'objet et de la taille de chaque objet.

10.1 Définitions

Syntaxe:

nom_tableau : **TABLEAU**[min_indice..max_indice]

DE <type_predefini>;

- ce qui signifie que les éléments ont pour type le type_predefini
- les **indices** des éléments vont de min_indice à max_indice,
- avec min_indice < max_indice,

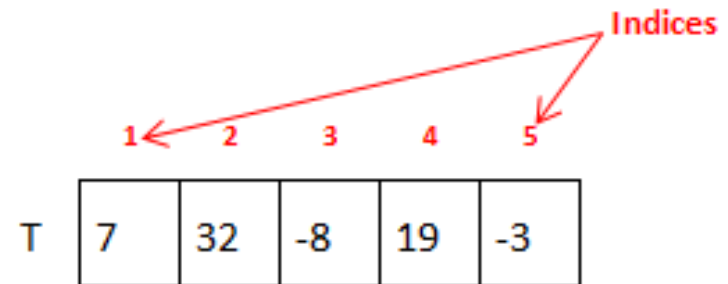
NB: on peut déclarer un tableau de *N valeurs* comme ceci:

nom_tableau: **TABLEAU**[*N*] **DE** <type_predefini>

10.1 Définitions

Exemple de tableau de 5 entiers :

T : TABLEAU [5] d' ENTIER



- T signifie que c'est un objet de type TABLEAU.
- Les numéros en indices 1, 2, 3, 4, 5 correspondent aux valeurs colonnes.
- Le contenu de T : les 5 entiers (dans un certain ordre)
- La première valeur est T[1] où 1 correspond donc à l'indice de la première colonne.



10.2 Fonctions utilisées un tableau

Parmi les traitements opérant sur des tableaux on peut:

- **créer** des tableaux
- **ranger (Saisir)** des valeurs dans un tableau
- **récupérer, consulter** des valeurs rangées dans un tableau
- **rechercher** si une valeur est dans un tableau
- **mettre à jour** des valeurs dans un tableau
- **modifier la façon** dont les valeurs sont rangées dans un tableau (par exemple : les **trier** de différentes manières)
- **effectuer des opérations** entre tableaux
- **comparaison de tableaux, addition multiplication,...**

10.2 Fonctions utilisées un tableau

Dans la suite du cours on déclare un **type** de tableau **TABL** de cette manière:

TYPE TABL = TABLEAU [TAILLE] D' ENTIER

TAILLE est la constante qui indique le nombre de case du tableau.

10.2 Fonctions utilisées un tableau

Exemple 1 :

```
PROCEDURE SAISIE_Tab (Var T: TABL; N : entier)
Var i : entier
DEBUT
    POUR i := 1 A N FAIRE
        saisir T[i]
    FINPOUR
FIN
```

Cette PROCEDURE permet de rentrer(saisir) les variables dans un tableau.

10.2 Fonctions utilisées un tableau

NB : on désigne le tableau par T et N par le nombre de colonnes, ainsi, même si l'on a déclaré auparavant un tableau à 50 colonnes et que l'on n'utilise en fait que 30 colonnes, $N = 30$ permet de définir à l'ordinateur le nombre de colonnes **réellement utilisées** et limiter la durée du traitement. N est donc indépendant de $TAILLE$ (mais $N \leq TAILLE$).

10.2 Fonctions utilisées un tableau

Exemple 2 :

PROCEDURE AFFICH_Tab (T: TABL; N : entier)

Var i : entier

DEBUT

 POUR i := 1 A N FAIRE

saisir T [i]

 FINPOUR

FIN

10.2 Fonctions utilisées un tableau

```
FONCTION Mintableau (T : TABL, N : entier) : entier
VAR i : entier
    Min : entier
Début
    Min := T[1]
    POUR i := 2 A N FAIRE
        SI T[i] < Min ALORS
            Min := T[i]
        FINSI
    FINPOUR
    retourner(Min)
FIN
```

10.3 Tri d'un tableau

Trier des objets est à la fois un **problème** fondamental de l'algorithmique, comme étape de prétraitement d'algorithmes plus complexes ou pour ordonner des structures de données (à quoi servirait un annuaire non trié ?)... et une bonne illustration de différents paradigmes de programmation (diviser pour régner)

Déterminer si un tableau est trié ??

5	8	8	12	15	3	2	1
---	---	---	----	----	---	---	---

10.3 Tri d'un tableau

Première proposition :

```
FONCTION est_trié (T : TABL N : entier) : BOOLEEN
VAR i : entier
DEBUT
i := 1
TANTQUE (i < N) ET (T[i] ≤ T[i+1])) FAIRE
    i := i+1
FINTANTQUE
    retourner (i=N)
FIN
```

10.3 Tri d'un tableau

Remarque :

Il y a un problème quand i vaut N : on évalue la condition $i < N$ ET $(T[i] \leq T[i+1])$ or la case $T[N+1]$ n'existe pas.

La première partie ($i < N$) de la condition renvoie faux --> la condition va renvoyer aussi faux ; Mais certains langages évaluent quand même la deuxième condition (ce n'est pas le cas du C++) : Donc à éviter de l'appliquer en algorithmique.

10.3 Tri d'un tableau - Tri par sélection

L'idée du tri consiste à **chaque étape** à **rechercher le plus petit élément** non encore trié et à le *placer à la suite des éléments déjà triés*.

A une étape i , les $i - 1$ plus petits éléments sont en place, et il nous faut sélectionner le i ème élément à mettre en position i .

L'analyse de cet algorithme donne :

A chaque étape i

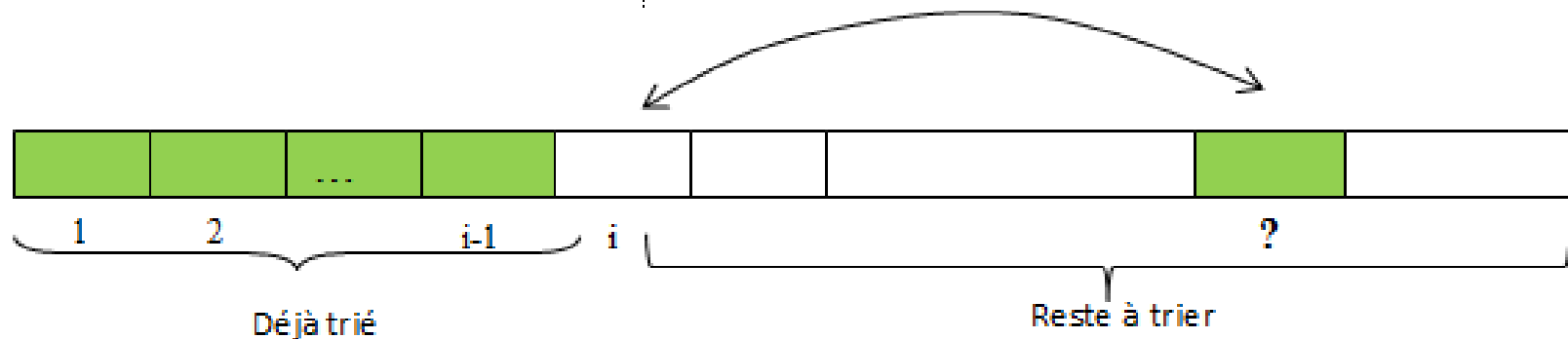
- **Rechercher** le i ème élément
- **Le placer** en position i

10.3 Tri d'un tableau - Tri par sélection

Méthode :

1. Chercher le plus petit élément du tableau restant

2. Échanger cet élément avec l'élément en position i



10.3 Tri d'un tableau - Tri par sélection

```
CONST TAILLE = 50
TYPE TABL=TABLEAU [TAILLE] DE REEL
PROCEDURE TriSélection (VAR T: TABL; TAILLE: ENTIER)
VAR i, pos: ENTIER
DEBUT
POUR i := 1 A TAILLE - 1 FAIRE
    ! Plus petit élément du tableau restant
    pos := trouverMin(T, i)
    ! Echanger avec l'élément en position i
    echanger(T, i, pos)
FINPOUR
FIN
```


10.3 Tri d'un tableau - Tri par sélection

! Fonction retournant l'indice du plus petit élément dans le tableau entre les indices i et la fin du tableau

```
FONCTION trouverMin (T: TABL i: ENTIER ) : ENTIER
VAR i_Min, j : ENTIER
DEBUT
  i_Min := i
  POUR j := i_Min + 1 A TAILLE FAIRE
    SI T [j] < T[i_Min] ALORS
      i_Min := j
    FINSI
  FINPOUR
  retourner( i_Min)
FIN
```

10.3 Tri d'un tableau - Tri par sélection

! procédure échangeant les éléments d'indices i et j dans un tableau

PROCEDURE *Echanger* (Var T: TABL; i, j: entier)

Var Aux : entier

DEBUT

Aux := T[i]

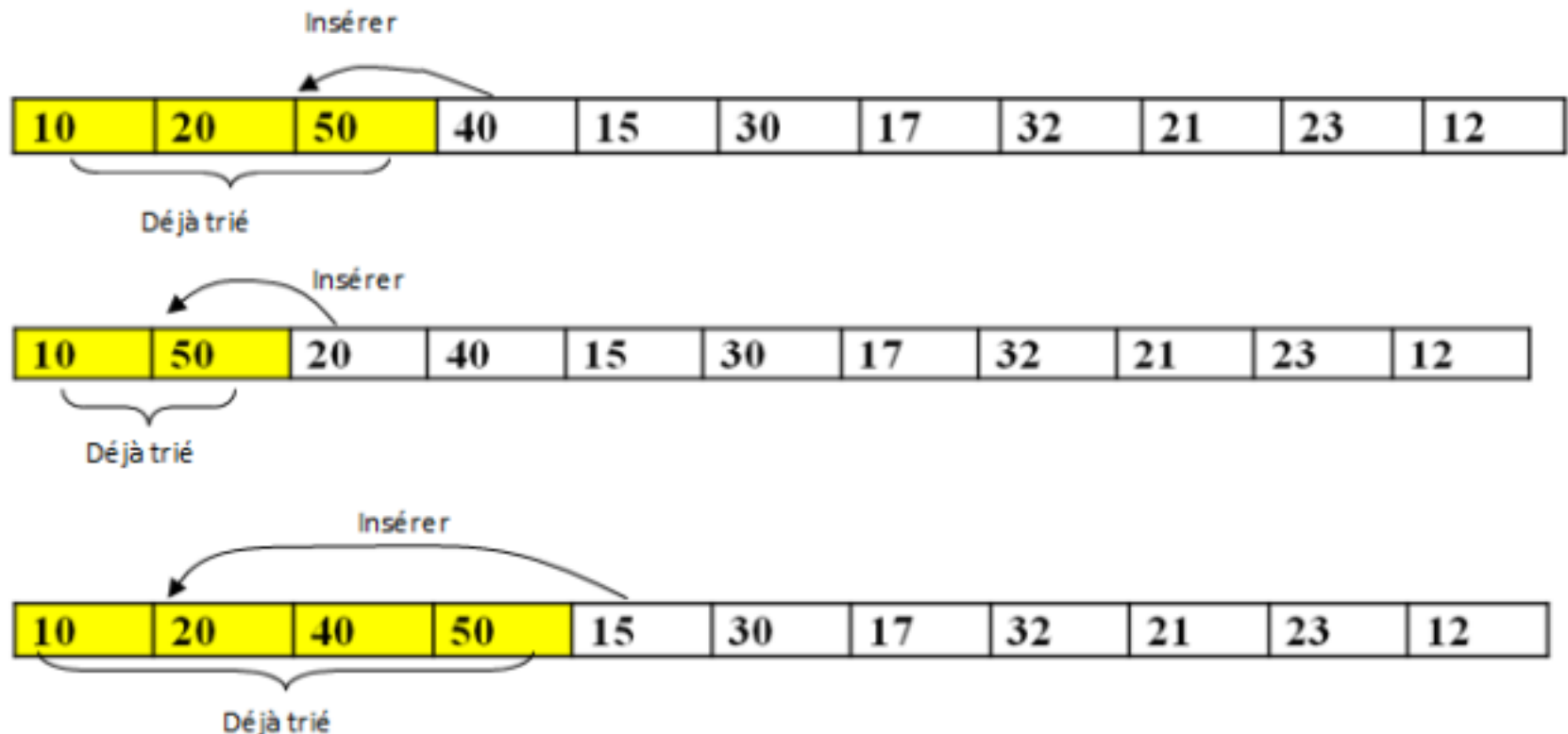
T[i] := T[j]

T[j] := Aux

FIN

10.3 Tri d'un tableau - Tri par insertion

Méthode : Trier le tableau de gauche à droite en insérant à chaque fois l'élément $i+1$ dans le tableau(déjà trié) des premiers éléments.



10.3 Tri d'un tableau - Tri par insertion

PROCEDURE INSERER(Var T : TABL; indice, i : entier)

Var j : entier

Aux : réel

Début

Aux := T[i]

POUR j := i - 1 A indice Par pas -1 FAIRE

T[j+1] := T[j]

FINPOUR

T[indice] := Aux

Fin

10.3 Tri d'un tableau - Tri à bulle

Le **tri à bulle** consiste à parcourir le tableau, par exemple de gauche à droite, en **comparant les éléments côte à côte** et en les permutant s'ils ne sont pas dans le bon ordre. Au cours d'une passe du tableau, les plus grands éléments remontent de proche en proche vers la droite comme des **bulles vers la surface**.

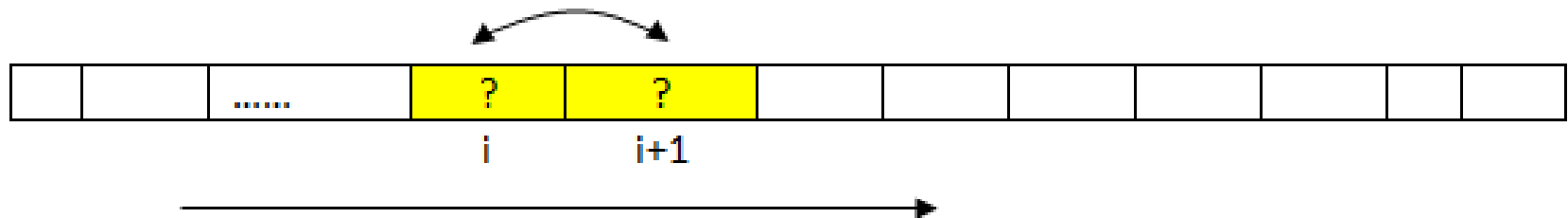
On s'arrête dès que l'on détecte que le tableau est trié : si aucune permutation n'a été faite au cours d'une passe.

10.3 Tri d'un tableau - Tri à bulle

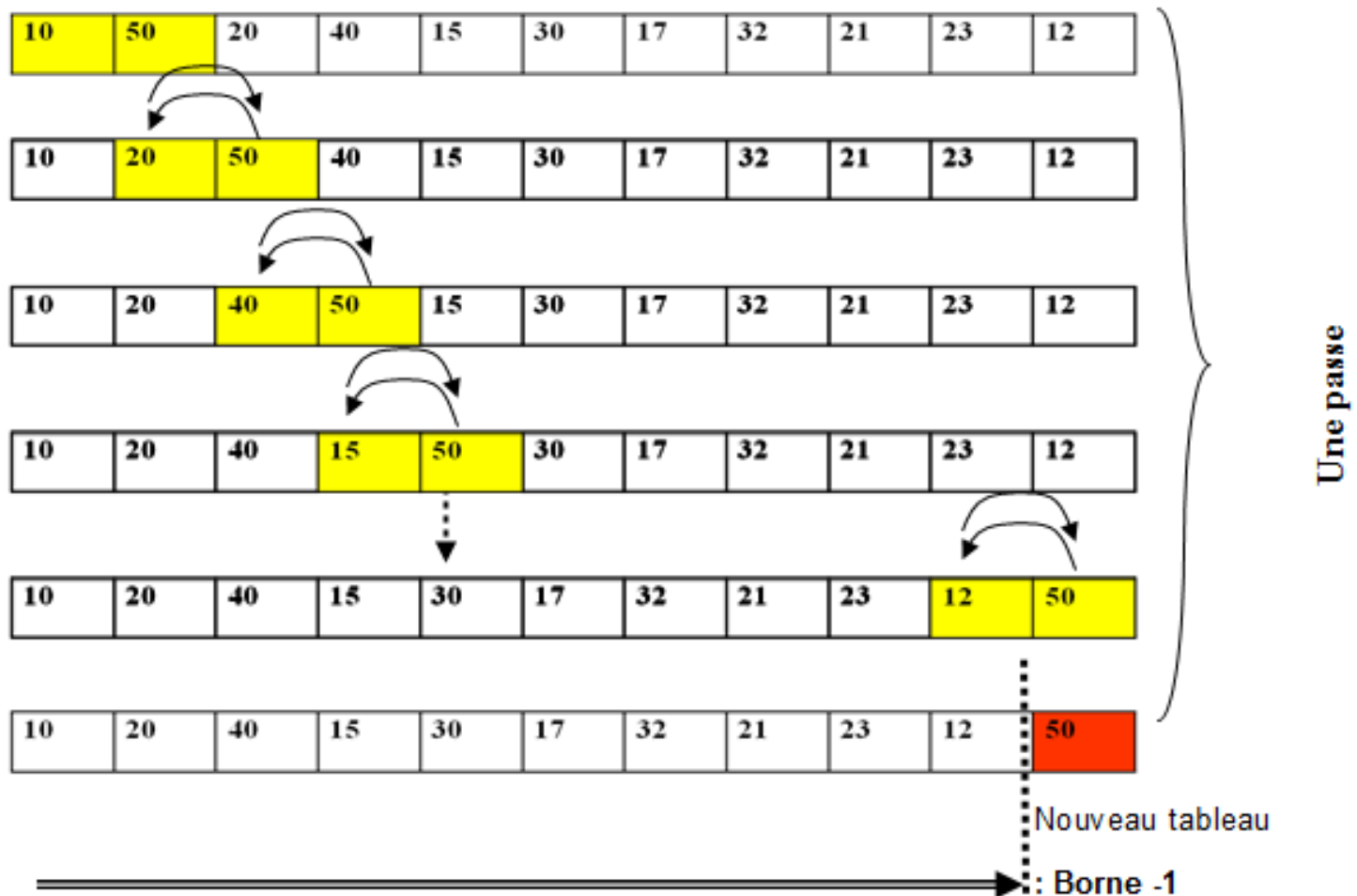
Tant que le tableau n'est pas trié
1. Effectuer **une passe** !



a. Parcourir le tableau
b. Si 2 éléments successifs ne sont pas dans le bon ordre, **les échanger**



10.3 Tri d'un tableau - Tri à bulle



10.3 Tri d'un tableau - Tri à bulle

ALGORITHME TriBulle

CONST TAILLE = 50

TYPE TABL = TABLEAU[TAILLE] REEL

Var T : TABL

 borne : entier

 TRI : booleen

Début

 TRI := FAUX

! Détecte si le tableau est trié

 borne := TAILLE

 TANTQUE (TRI = FAUX) FAIRE

! Itération sur les passes

 TRI := VRAI

! Initialisation avant une passe

 POUR i := 1 A borne - 1 ALORS

! Effectue une passe

 SI T[i] > T[i+1] ALORS

 echanger(T, i, i+1)

 TRI := FAUX

 FINSI

 FINPOUR

 borne := borne - 1

 FINTANTQUE

Fin