Afpa
se former, avancer

L'ALGORITHME

Écrire du pseudo-code

Concepteur Développeur d'Applications - 2024 - JBO_v3

ALGORITHME : RECETTE DE CUISINE ?

Un algorithme est une suite logique d'instructions qui, quand elles sont exécutées correctement aboutissent au résultat attendu.

On peut les décrire de manière générale, identifier des procédures, des suites d'actions ou de manipulations précises à accomplir séquentiellement.

C'est cela, un algorithme : *le concept qui traduit la notion intuitive de procédé systématique, applicable mécaniquement, sans réfléchir, en suivant un mode d'emploi précis.*

Un algorithme c'est donc « presque » comme une recette de cuisine.

EXEMPLE

On fait tous de l'algorithme sans le savoir. Par exemple en cuisine, si on souhaite faire une omelette, on a des étapes à passer avant de pouvoir la manger…

Algorithme de l'omelette :

Prendre 6 œufs, un saladier, une fourchette et une poêle
Pour chaque nombre entre 1 et 6 (inclus tous les deux) :
    Casser un œuf dans le saladier
    Jeter la coquille à la poubelle
Saler, poivrer et fouetter le contenu du saladier
Faire cuire à la poêle

COMPÉTENCES POUR L'ALGORITHMIQUE !!

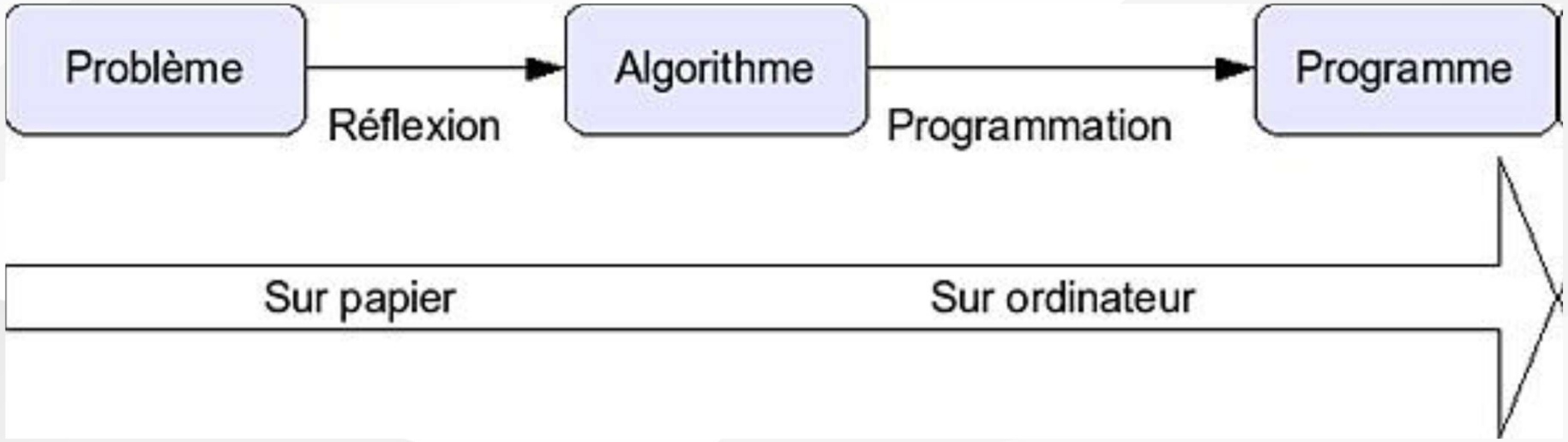
Être bon en mathématique ?

- *oui* *et* *non* *!!* Cela permet plus facilement d'arriver au résultat d'une problématique donnée.
- mais il faut simplement :
  - avoir de l'intuition : elle s'apprends par l'expérience
  - être logique : on ne fait pas d'omelettes sans casser les œufs au préalable.
  - être méthodique et rigoureux : toujours se mettre à la place de la machine!
- Enfin, la pratique vous permettra de gagner en efficacité.

Et La programmation ?

➢ Monsieur le formateur, c'est quand qu'on code ?... Pourquoi ne pas directement coder ?

1. Attention : dans les tests techniques, c'est très souvent demandé d'écrire de l'algorithme !!!
2. Faire de l'algorithme permet d'apprendre sans les particularités d'un langage
   - En effet, moins de contraintes car c'est dans un langage naturel qu'on pratique tous les jours.



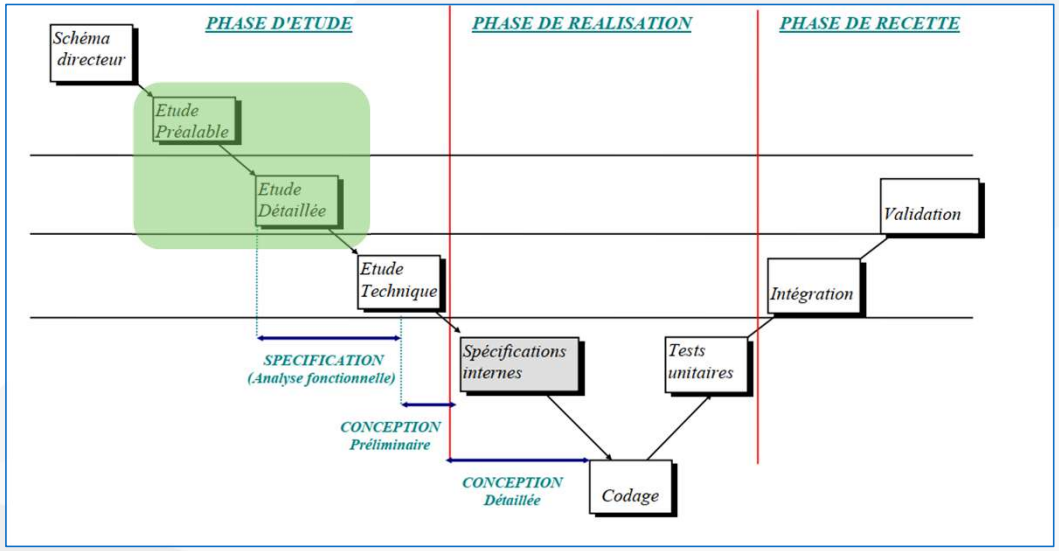
DE LA RÉFLEXION À LA PROGRAMMATION

CYCLE DE VIE D'UN PROJET

Dans le cycle de vie d'un projet (ici le cycle en V), le processus d'écriture algorithmique vient dans la conception préliminaire - conception détaillée.

Même si avec l'expérience, l'écriture d'algorithmique peut sembler inutile, cela reste un bon moyen de mettre en place une documentation technique pour d'autres personnes et vous fera gagner du temps sur le codage en évitant les régressions du code.

Toujours penser à ceux qui vont relire même vous !!!



L'ALGORITHME TEXTUEL

On utilise le **pseudo-code** pour exprimer clairement et formellement un algorithme :

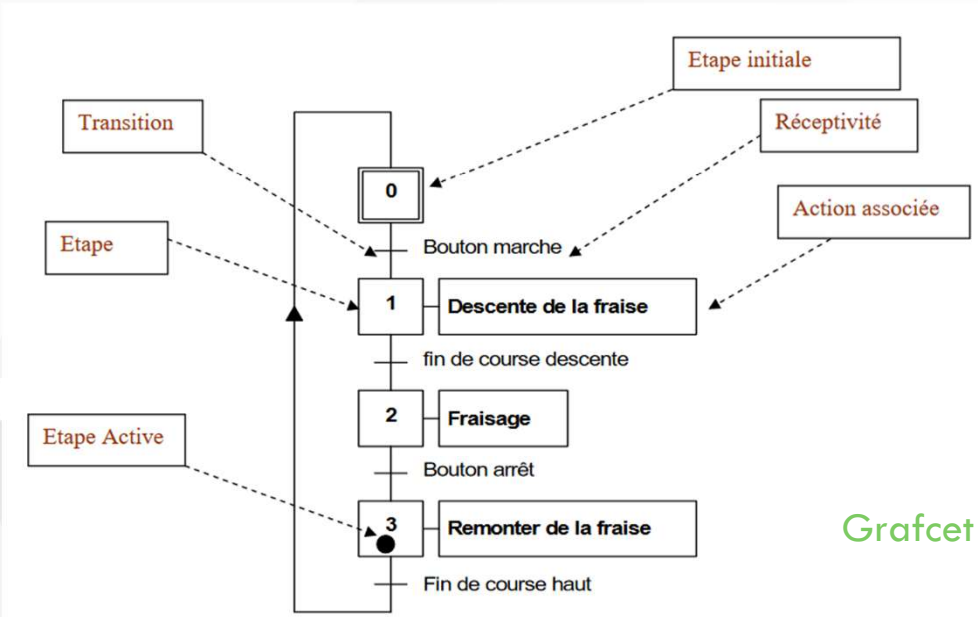
- Proche d'un langage de programmation mais ce n'est pas du codage.
- Expression d'idées formelles dans la langue naturelle de ses usagers en imposant une forme rigoureuse (règles et normalisation).
- Bien adapté aux problèmes informatiques et réponds bien à un type de problèmes donnés.
- Son formalisme a été repris dans de nombreux langages de programmation.

L'algorithme se définie suivant :

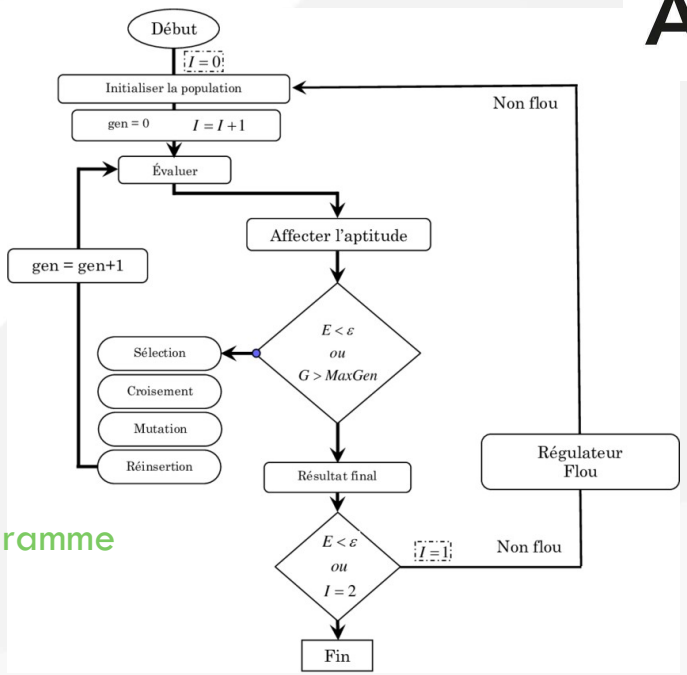
- La définition des "entités"
- L'utilisation des "entités" définies les actions
- La présentation
  - Commentaires et l'indentation
- L'esprit dans lesquels les algorithmes sont construits.

Autres formalismes :

- *Graphiques : organigrammes, réseaux de Pétri, Grafcet.*



Grafcet



Organigramme

REPRÉSENTATION EN GRAPHE

ÉCRIRE DU PSEUDO-CODE

DÉBUTONS PAR DE LA DÉFINITION

STRUCTURE D'UN ALGORITHME TEXTUEL

3 parties principales : Définitions / Déclarations / Instructions (actions)

Définitions

Déclarations

Instructions

```
// Définition des objets du bloc
Interface de bloc     // « Programme » ou « Procédure » ou « Fonction »
// Commentaire sur le rôle du bloc
Constantes            // Définition des constantes du bloc
Types                 // Définitions des types du bloc
Variables             // Définition des variables du bloc
Procédures ...        // Définitions des procédures appelées par le bloc
Fonctions ...         // Définition des fonctions appelées par le bloc
// Définition des actions du bloc
début
   actions            // Actions qui seront effectuées lors de l'activation du bloc
fin
```

NOTIONS D'ENTITÉS ET D'ACTIONS

Les entités forment l'ensemble des éléments qui sont manipulés dans un algorithme.

Plusieurs types d'entités :

- Les types :
  - Défini par un indicateur nom et une référence à une famille (entier, réel etc..).
  - Permet ainsi de classer les entités dans une famille, ainsi que les manipulations possibles sur ce type.
- Les constantes :
  - Défini par un identificateur, représentée par une valeur et défini par un type. La constante est invariante au cours du temps. Quel est l'intérêt d'une constante ? Faire en sorte que l'entité devienne invariante, empêchant toutes modifications.
- Les variables :
  - Défini par un identificateur, un type et un contenu qui va évoluer au cours du temps.

A QUOI SERVENT LES VARIABLES ?

Dans un programme informatique, on va avoir en permanence besoin de stocker provisoirement des valeurs.

- Il peut s'agir de données issues du disque dur, fournies par l'utilisateur (frappées au clavier), etc.
- Il peut aussi s'agir de résultats obtenus par le programme, intermédiaires ou définitifs.
- Ces données peuvent être de plusieurs types : elles peuvent être des nombres, du texte, etc.

Toujours est-il que dès que l'on a besoin de stocker une information au cours d'un programme, on utilise une variable.

En résumé :

- une variable a un type qui permet de définir la valeur qu'elle va contenir
- une variable est définie par un indicateur permettant de la nommer et de la retrouver facilement.
- une variable sera stockée dans la mémoire de l'ordinateur
- une variable pourra évoluer dans le temps, c'est-à-dire qu'on pourra lui affecter plusieurs valeurs différentes. Le contraire d'une variable est donc une constante.

NOTIONS D'ENTITÉS ET D'ACTIONS

- Les procédures et fonctions.
  - Définie par un identificateur qui permet de l'appeler, les types des objets qu'elle va manipuler (les paramètres), les actions qu'elle effectue sur les objets et qui lui seront donnés à l'appel de la procédure.
  - Une procédure exécute des instructions sans retourner de résultats à l'appelant.
  - Une fonction exécute des instructions retournant un résultat à l'appelant.
- Les actions.
  - Opérations qui pourront être réalisées sur les entités définis dans l'algorithme.
  - Plusieurs types d'actions :
    - Observation : comparaison de deux objets.
    - Modification : donne une valeur à une variable.
    - Alternative : effectuer des actions suivant certaines conditions.
    - Répétitive : itérer des actions selon une condition de terminaison.
    - Complexe : appel d'une procédure ou une fonction.

LE LANGAGE ALGORITHMIQUE

ELÉMENTS DU LANGAGE

CARACTÈRES UTILISÉS ET MOTS RÉSERVÉS ET SYMBOLES

Les majuscules	A ... Z
Les minuscules	a ... z
Les chiffres	0 ... 9
Les signes	_ = < > ' () [ ] * + - / , : . Espace

Conventions de Nommage

alors	autrecas	booléen	caractère
choix	constantes	créer	de
début	détruire	div	écrire
enregistrement	entier	entrée	et
faire	faux	fermer	fichier
fin	finchoix	finenregistrement	finfichier
finsi	fintantque	fonction	indexé
jusquà	lire	mod	non
null	ou	ouvrir	pointeur
positionner	procédure	programme	quelconque
réel	répéter	retourner	si
sinon	sortie	sur	tableau
tantque	types	variables	vrai
:=	<>	>=	<=
//	->		

RÈGLES ET CONVENTIONS

Règles

- doit être le plus lisible possible, de manière que n'importe qui d'autre que l'auteur soit capable de comprendre en le lisant.
- ne doit pas être trop long (une page écran).
  - S'il est trop long, il faut le découper en fonctions et procédures.
- Les structures de contrôle doivent être indentées. Il en est de même pour les répétitives.
  - A chaque imbrication d'une structure de contrôle, on décale d'une tabulation.

Conventions

- Le nom des variables doit être significatif, c'est à dire indiquer clairement à quoi elles servent.
- On doit toujours être capable de donner un nom significatif à une procédure ou à une fonction.
- Le nombre de paramètres ne doit être trop grand (en général inférieur à 5) car cela nuit à la lisibilité du programme.
- Une procédure ou une fonction doit être la plus générale possible de manière à pouvoir être réutilisée dans d'autres circonstances.
- Si le but d'une procédure est de calculer une valeur simple, il est préférable d'en faire une fonction.
- Il est souvent plus clair d'écrire une fonction booléenne plutôt qu'une condition complexe.

LES FAMILLES

Pour simplifier le traitement des chaines de caractères, on peut partir du principe d'un type Chaine
texte : Chaine *// déclaration*
Texte[0] correspondra au 1er caractère de la chaine

- **Entiers**
  - Opérateurs de comparaison : =, >, <, >=, <=, <>
  - Opérateurs de calcul : + - * div mod
- **Caractères**
  - Opérateurs de comparaison : =, >, <, >=, <=, <>
  - A est différent de a
    ASCII
- **Booléens**
  - vrai / faux
  - Opérations logiques : et , ou , non
  - Opérations de comparaison : =, >, <, >=, <=, <>
- **Réels**
  - Opérateurs de comparaison : =, >, <, >=, <=, <>
  - Opérateurs de calcul : + - * /
- **Tableaux**
  - Tableau [10] de entier
  - Tableau[i] où i représente l'index de l'élément dans le tableau (0 à …)
  - Tableaux à 1 ou 2 dimensions : tab[i] ou tab[i][j]
- **Enumérations**
  - types état = (marche, arrêt, panne)
    Moteur : état

LES INSTRUCTIONS

- Déclaration du programme principal

```
Programme nomProgramme
… // section des déclarations constantes, variables, types, fonctions, procédures
Début
    …
Fin
```

- Déclaration de constantes

Constante NOM : [Type] = valeur

Constante PI : Réel = 3,141559

- Déclaration de variables

Variable nom : [Type] = valeur

Variable compteurLettres : Entier

VISIBILITÉ DES OBJETS.

Le principe est simple : un objet est visible (utilisable) dans l'unité algorithmique qui l'a défini, et dans toutes les unités algorithmiques définies dans celle-ci.

Mais il est interdit d'utiliser des variables dans une procédure ou fonction, qui ne seraient pas définies dans la procédure ou fonction ou en paramètre de celle-ci.

Notions de variables globales et locales :

- Une variable est dite **locale** si elle est définie dans l'environnement d'où on la regarde.
- Elle est dite **globale** si elle est définie dans un environnement qui est le père, ou un aïeul, de cet environnement.

➢ Donc une même variable peut être considérée globale ou locale relativement à l'endroit d'où elle est vue.

LES INSTRUCTIONS

- Déclaration de fonctions

```
Fonction nomFonction(param1 : [Type], param2 : [Type]...) : [TypeRésultat]
Constante ...
Variables
    resultat : TypeRésultat
Début
    ...
    // renvoi le résultat à l'appelant
    retour resultat
Fin
```

- Appel de fonction

variable ← nomFonction(param1, param2…) résultat ← racine(69) ou Afficher(Racine(69))

Affectation ou assignation

Le symbole ← représente une affectation qui consiste d'attribuer une valeur à une variable.
On peut aussi utiliser := voir = mais peut être confondu avec un test

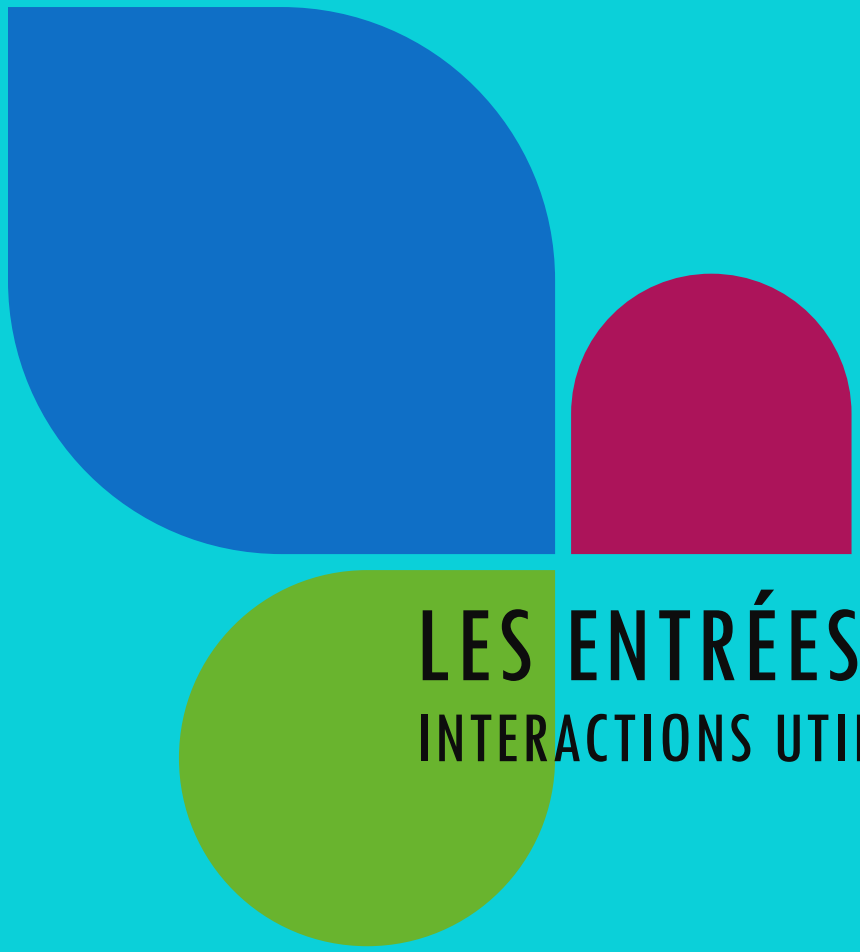
LES INSTRUCTIONS

- Déclaration de procédures

```
Procédure nomProcédure(param1 : [Type], param2 : [Type]…)
    Constante …
    Variable …
    Début
        Actions
    Fin
```

- Appel de procédure

nomProcédure(param1, param2…) afficher(Bonjour)



LES ENTRÉES ET SORTIES

INTERACTIONS UTILISATEUR AVEC LE PROGRAMME

ENTRÉE / SORTIE

- À la console

Il existe des actions complexes qui permettent :

- Lire des informations au clavier.
  - Lire(nombre) où nombre est une variable qu'on aura définie préalablement.
  - *Affectera le résultat de la lecture dans la variable*
- Ecrire des informations à l'écran.
  - Ecrire('le résultat est : ', nombre)

• Fichier

Nous pouvons ouvrir un fichier, lire un élément du fichier, écrire un élément dans le fichier, fermer le fichier, et tester la fin du fichier :

- Déclarer un fichier
  - nomfic : fichier de type élément
- Ouvrir un fichier
  - ouvrir('fichier.txt', nomfic)
- Lire un fichier
  - lire(nomfic, variable)
- Écrire dans un fichier
  - écrire(nomfic, valeur)
- fermer un fichier
  - fermer(nomfic)
- tester un fichier
  - Si finfichier(nomfic) alors …

QUELQUES EXEMPLES

Algorithme par la pratique.
apprendre à lire et à écrire nos premiers algorithmes

PARTIE 1

Exercice 1.1

Quelles seront les valeurs des variables A et B après exécutions des instructions suivantes ?

```
Variable A, B : entier

Début
   A := 1
   B := A + 3
   A := 3
Fin
```

Exercice 1.2

Quelles seront les valeurs des variables A, B et C après exécutions des instructions suivantes ?

```
Variable A, B, C : entier

Début
   A := 5
   B := 3
   C := A + B
   A := 2
   C := B - A
Fin
```

PARTIE 1

Exercice 1.3

Quelles seront les valeurs des variables A et B après exécutions des instructions suivantes ?

```
Variable A, B : entier

Début
  A := 5
  B := A + 4
  A := A + 1
  B := A - 4
Fin
```

Exercice 1.4

Quelles seront les valeurs des variables A, B et C après exécutions des instructions suivantes ?

```
Variable A, B, C : entier

Début
  A := 3
  B := 10
  C := A + B
  B := A + B
  A := C
Fin
```

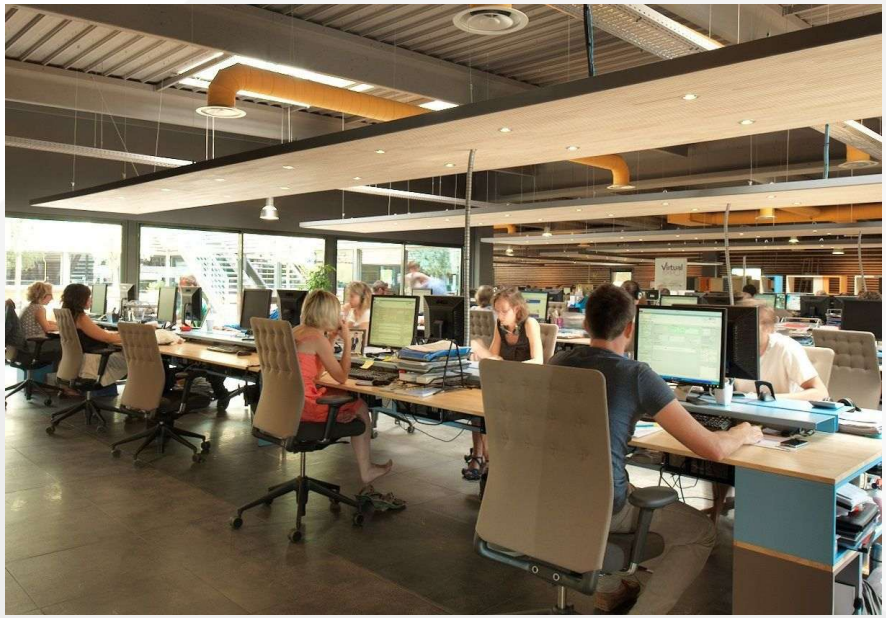
PARTIE 1

Exercice 1.5

Que produit l'algorithme suivant ?

```
Variable A, B, C : caractères

Début
   A := "423"
   B := "12"
   C := A + B
Fin
```



PARTIE 1

A réaliser

Exercice 1.6

C'est un classique, écrire un algorithme permettant d'échanger les valeurs de deux variables A et B, et ce quel que soit leur contenu préalable (Exemple : des entiers) ?

Exercice 1.7

Variante du 1.6, on dispose de trois variables A, B et C.

Ecrivez un algorithme transférant à B la valeur de A, à C la valeur de B et à A la valeur de C (toujours quels que soient les contenus préalables de ces variables) (Exemple : des entiers) ?

QUELQUES EXEMPLES

Algorithme par la pratique.
apprendre à écrire nos premiers algorithmes
Utilisation de fonctions et de procédures

PARTIE 2

A réaliser

Exercice 2.1

Ecrire un algorithme qui demande à l'utilisateur le prix Hors taxe d'un objet et qui donne sa valeur TTC (multiplier le prix par 1.196).

Exercice 2.2

Ecrire un algorithme qui demande à l'utilisateur son prénom et son nom et qui affiche ensuite la phrase "Bonjour *prénom* votre nom est *nom*"

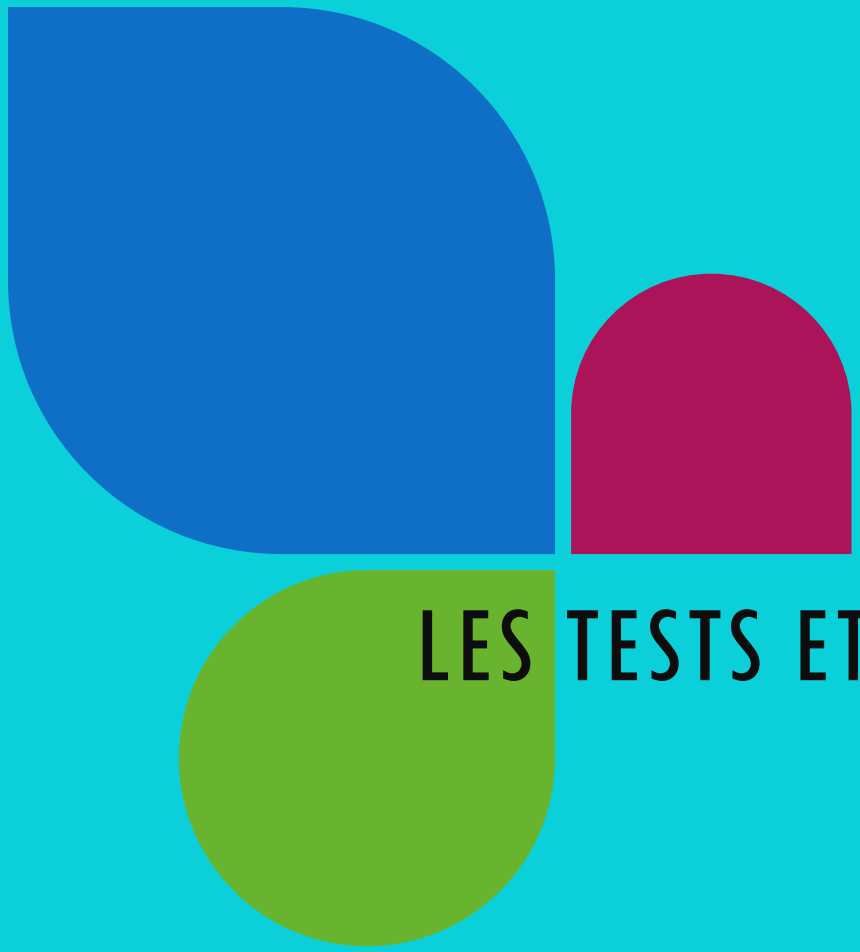
Variante : avec une fonction et une procédure

LES OPÉRATEURS

AGIR AVEC NOS ENTITÉS

LES OPÉRATEURS

Nature	Variables utilisées	Notation	Signification
Opérateurs arithmétiques	Entier Réel	+	Addition
		-	Soustraction
		*	Multiplication
		/	Division (réelle)
		DIV	Division entière
		MOD	Reste de la division entière
Opérateurs logiques	Booléen Entier	et	Fonction ET
		ou	Fonction OU
		ouex	Fonction OU EXCLUSIF
		non	Fonction NON
Opérateur de concaténation	Chaîne de caractères	+	Concaténation
Opérateurs de comparaison	Booléen Entier Réel Caractère Chaîne de caractères	=	Egal
		≠	Différent
		<	Inférieur
		>	Supérieur
		≤	Inférieur ou égal
		≥	Supérieur ou égal



LES TESTS ET CONDITIONS

LES CONDITIONS

Une condition est une comparaison

Elle signifie qu'une condition est composée de trois éléments :

- une valeur
- un opérateur de comparaison
- une autre valeur.

Les valeurs peuvent être a priori de n'importe quel type (numériques, caractères…).

- Mais si l'on veut que la comparaison ait un sens, il faut que les deux valeurs de la comparaison soient du même type

Conditions composées

Certains problèmes exigent parfois de formuler des conditions qui ne peuvent pas être exprimées sous la forme simple exposée ci-contre.

Prenons le cas Toto est inclus entre 5 et 8.

- En fait cette phrase cache non une, mais deux conditions. Car elle revient à dire que Toto est supérieur à 5 et Toto est inférieur à 8.
- Il y a donc bien là deux conditions, reliées par ce qu'on appelle un opérateur logique, le mot ET.

L'informatique met à notre disposition quatre opérateurs logiques : ET, OU, NON, et XOR.

ALGÈBRE DE BOOLE

L'algèbre de Boole permet d'effectuer des opérations entre des entrées et des sorties, et utilise des portes logiques pour faire modifier les signaux et avoir des sorties potentiellement différentes des entrées.

Elle utilise principalement les portes logiques :

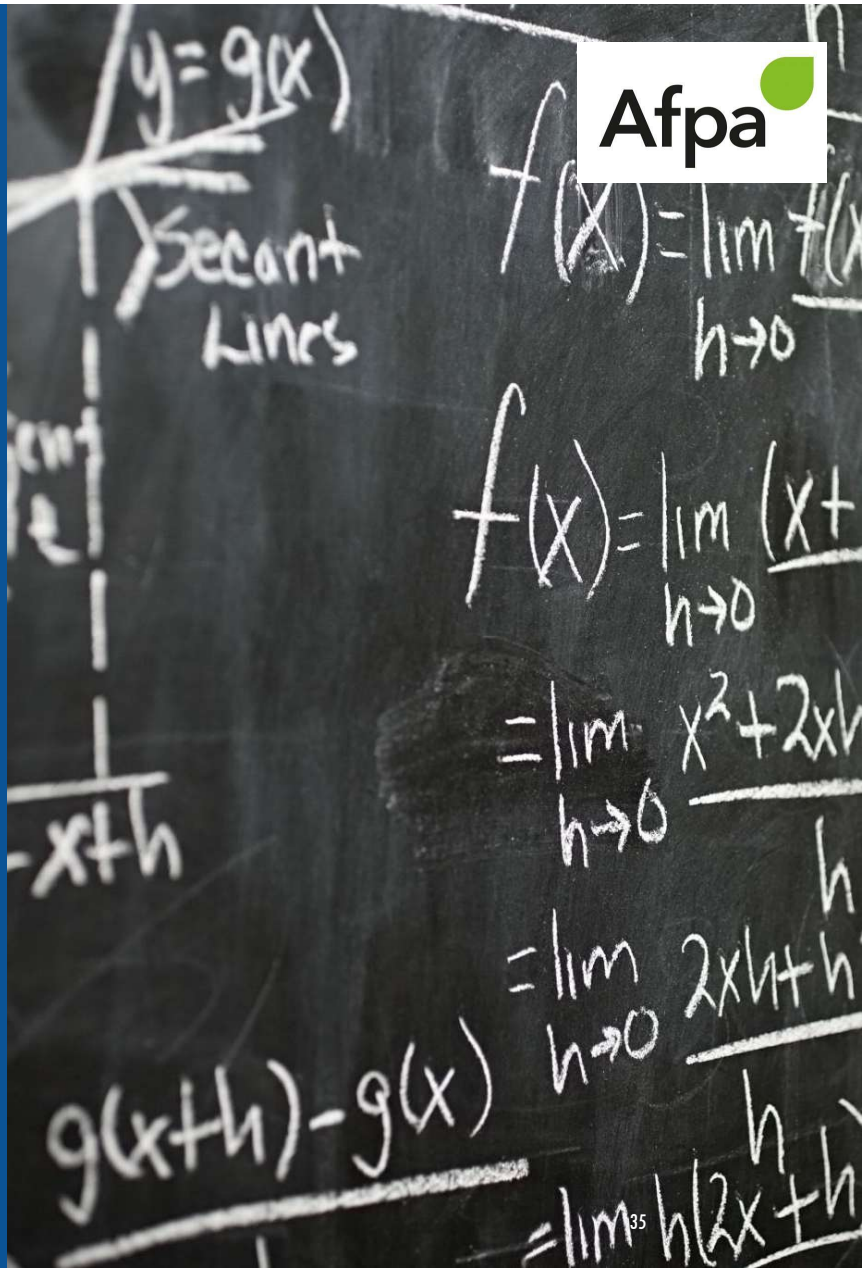
- ET : Les deux entrées doivent être vérifiées pour que la sortie se déclenche.

- OU : Au moins une des deux entrées doit être vérifiée pour que la sortie se déclenche.

- NON : La sortie et l'inverse de l'entrée.

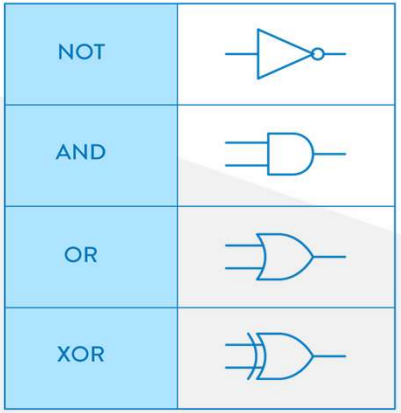
- OU EXCLUSIF : Moins souvent utilisée en programmation classique (mais plutôt en électronique), c'est une combinaison de ET, de OU et de NON.

Une fois le schéma logique fait, on peut facilement passer au code sans trop de difficultés.



LES TABLES DE VÉRITÉ

Symboles des portes logiques



ET (AND)		
A && B		
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

XOU (XOR)		
A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

OU (OR)		
A || B		
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

NON (NOT)	
!A	
A	$\bar{A}$
1	0
0	1

MISE EN PLACE DES TESTS

Les tests vont servir à conditionner nos actions.

- **Si** une condition du test est respectée **alors** l'action peut se dérouler.

Il existe plusieurs types de structures de contrôle permettant de mettre en place tout un ensemble d'enchainements d'actions conditionnés avec les tests.

- **Si** une condition du test est respectée **alors** l'action peut se dérouler **Sinon** j'effectue une autre action.

C'est à cela que vont nous servir les tables de vérité car on va pouvoir composer des conditions.

Quelques exemples :

Si Toute valeur supérieure à 18 alors…

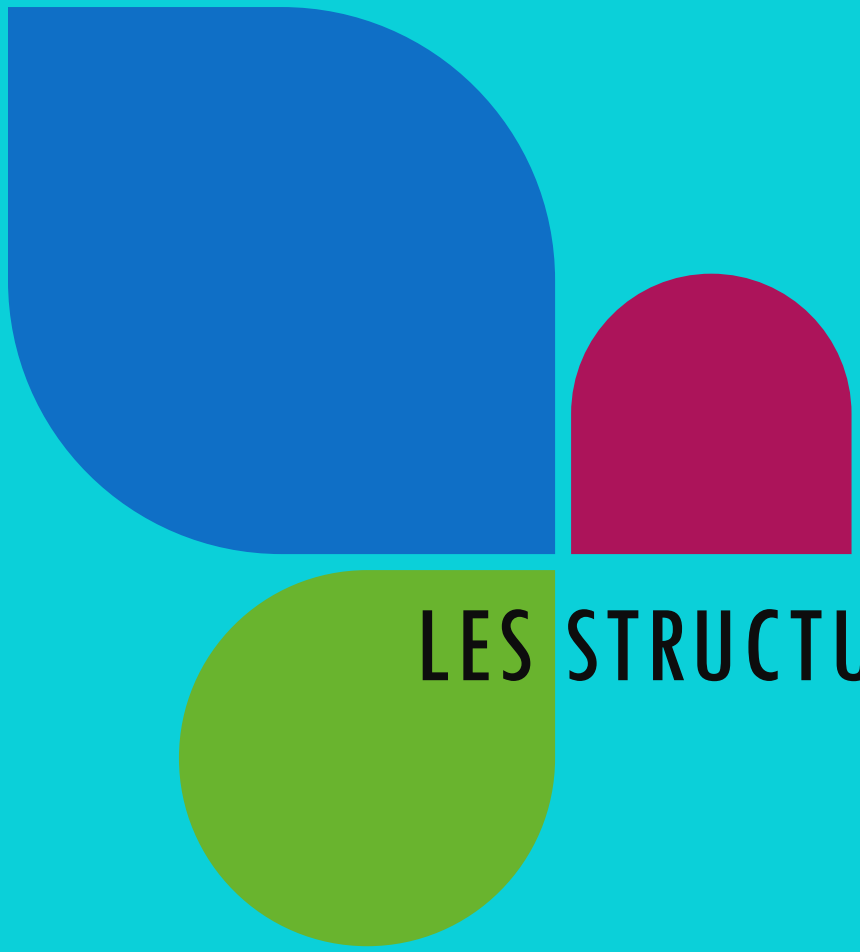
Valeur > 18

Si Toute valeur inferieur à 18 ***ou*** *supérieur à 25 alors…*

Valeur < 18 **ou** valeur > 25

Si Le courant est présent et le bouton Power est sur marche alors…

Courant = alimenté **et** Power = marche

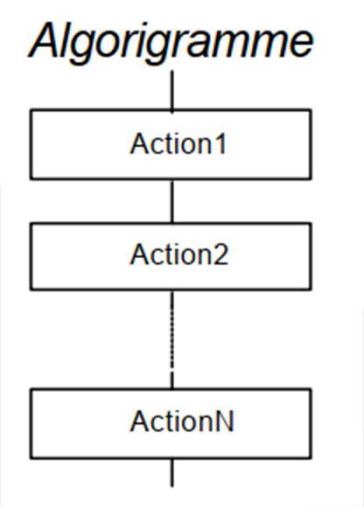


LES STRUCTURES ALGORITHMIQUES

LES STRUCTURES ALGORITHMIQUES

Séquence linéaire :

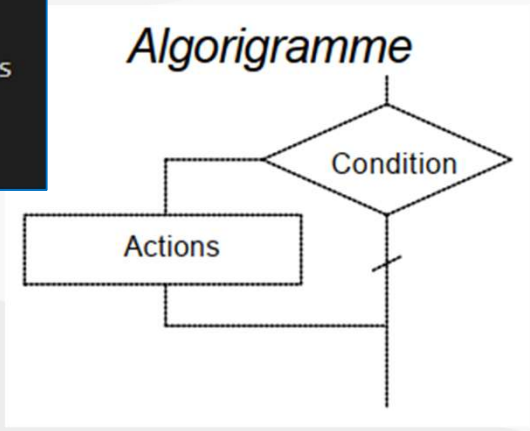
```
Action1
Action2
…
Action N
```



Structure Si … Alors … :

```
Si condition
   Alors Actions
FinSi

Si condition Alors
    Actions
FinSi
```

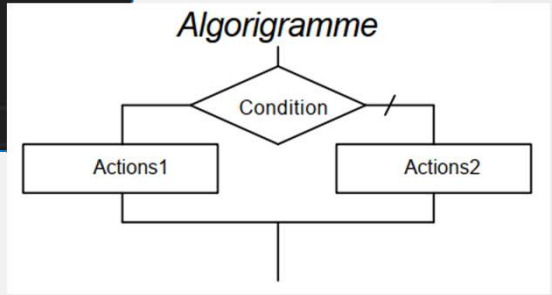


LES STRUCTURES ALGORITHMIQUES

Structure Si … Alors … Sinon … :

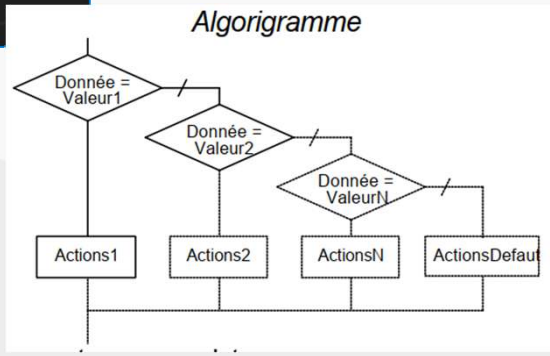
```
Si condition
   Alors Action1
   Sinon Action2
FinSi

Si condition Alors
   Action1
Sinon
   Action2
FinSi
```



Structure Choix multiple :

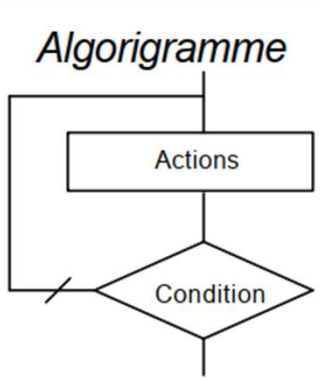
```
Cas où donnée Vaut
   Valeur1 : Actions1
   Valeur2 : Actions2
   …
   ValeurN : ActionsN
   Autre : ActionsDéfaut
FinCas
```



LES STRUCTURES ALGORITHMIQUES

Structure Répéter … Jusqu'a :

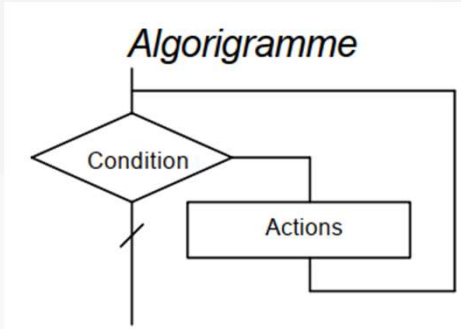
```
Répéter
    Actions
Jusqu'à Condition
```



La vérification de la condition s'effectue après les actions. Celles-ci sont donc exécutées au moins un fois.

Structure Tant que … Faire …

```
TantQue Condition
    Faire Actions
FinTantQue
```



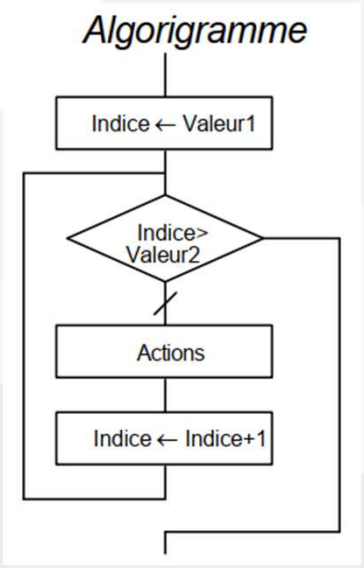
La vérification de la condition s'effectue avant les actions. Celles-ci peuvent donc ne jamais être exécutées.

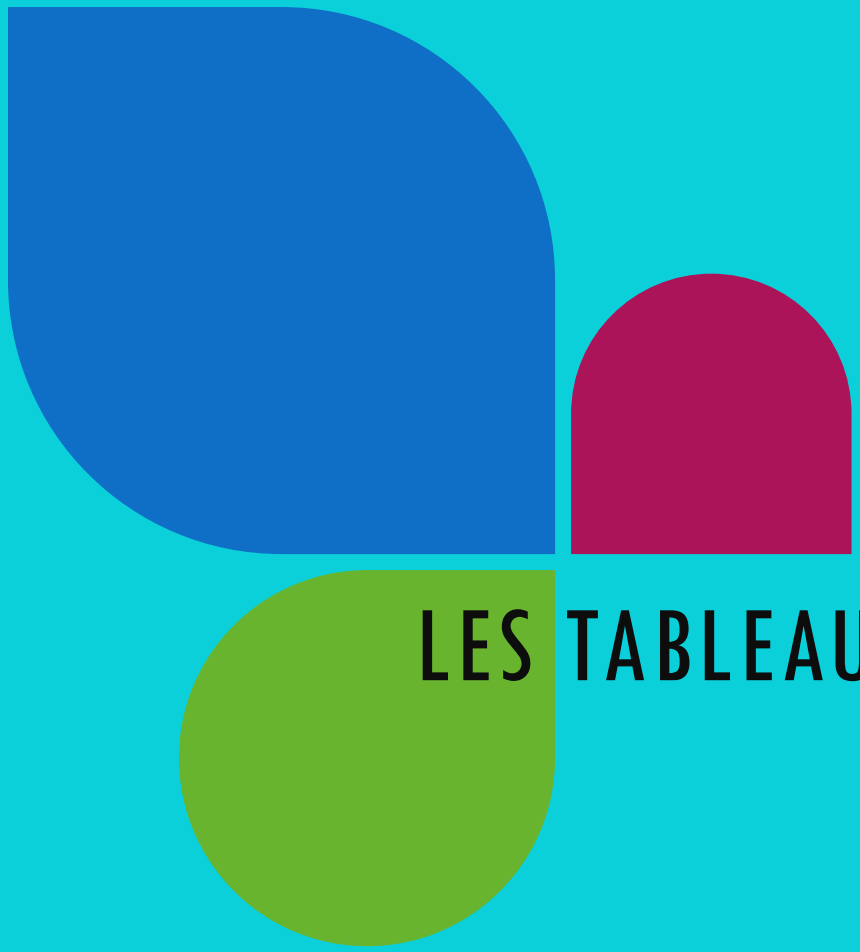
LES STRUCTURES ALGORITHMIQUES

Structure Pour indice allant de … a … faire :

```
Pour indice Allant de valeur1 à valeur2
    Faire
        Actions...
        indice := indice + 1
    FinFaire
FinPour
```

Dans ce type de boucle, on inclue un pas d'incrémentation (+1, +2 …)





LES TABLEAUX

MANIPULATION TABLEAU, LISTE…

Pour rappel, la création d'un tableau :

```
Variable tab : Tableau[10] d'Entier
```

- tab sera un tableau qui va contenir 10 entiers.
- Le premier élément de notre tableau se trouvera à l'indice 1
- Le dernier élément de notre tableau se trouvera à l'indice 10

Nous sommes en pseudo- code, d'où volontairement je commence à 1 comme indice du 1er élément de mon tableau.

Accès à un élément de notre tableau :

- Tab[indice] où indice correspondra à l'emplacement de l'élément ciblé.

Parcours de notre tableau :

- Pour parcourir l'ensemble de notre tableau, nous allons utiliser la structure Pour…

```
Pour indice Allant de 1 à longueur_tableau
    Faire
        // ex : si on souhaite écrire dans le tableau
        Lire tableau[indice]

        // ex : si on souhaite lire le contenu du tableau
        Ecrire tableau[indice]
    FinFaire
FinPour
```

QUELQUES EXEMPLES

Algorithme par la pratique.

PARTIE 2

A réaliser

Exercice 2.3

Ecrire un algorithme qui demande deux nombres à l'utilisateur et l'informe ensuite si le produit est négatif, positif, ou nul et afficher le résultat du produit obtenu ?

PARTIE 3

A réaliser

Exercice 3.1

Ecrire un algorithme qui demande un nombre compris entre 10 et 20, jusqu'à ce que la réponse convienne. En cas de réponse supérieure à 20, on fera apparaître un message : "plus petit" et inversement "plus grand" si le nombre est inférieur à 10.

Exercice 3.2

Ecrire un algorithme qui demande un nombre de départ et qui calcule sa factorielle.

Note : la factorielle de 8 vaut : 1x2x3x4x5x6x7x8