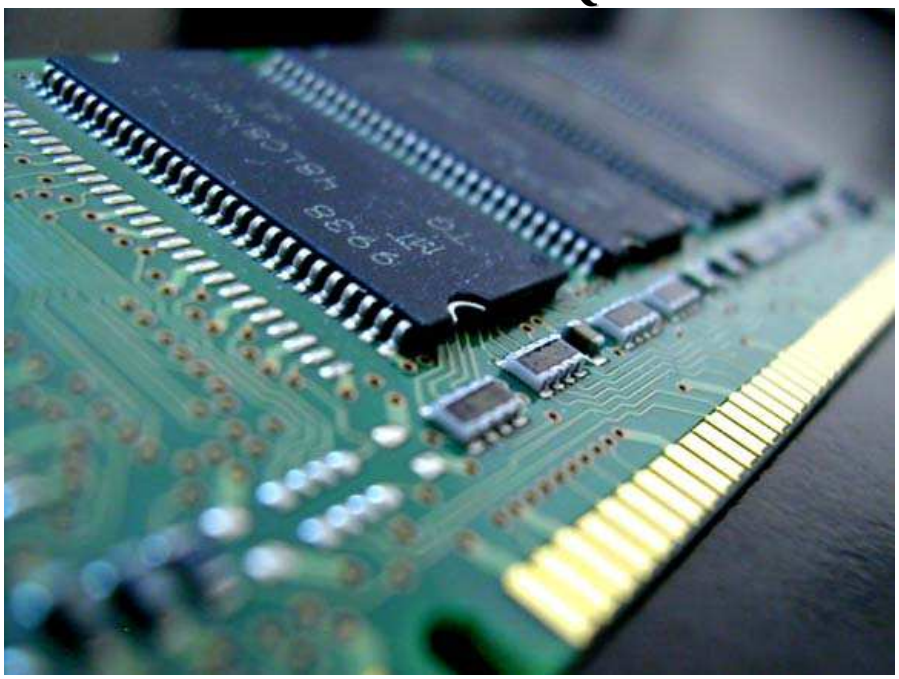
# ALGORITHMIQUE



Baccalauréat STI 2D - Systèmes Informatiques et Numériques

**Référence au programme de formation :**

- Comportement informationnel des systèmes (ET 2.3.6)

## I/ Objectifs et prérequis

### Objectifs

A la fin de la séquence, l'élève doit être capable

- de décrypter un algorithme comportant des structures itératives (boucles) et conditionnelles
- de décrire le comportement d'un système à partir de son algorithme
- d'élaborer un algorithme simple

### Pré-requis

Aucun prérequis

## II/ Expression du besoin

La complexification des systèmes techniques a rendu nécessaire l'élaboration d'***outils de descriptions*** grâce auxquels ont peut, de manière structurée et codifiée, expliciter :

- le comportement d'un système automatisé
- une démarche opératoire

- un raisonnement

L'algorithmique est un de ces outils.


Elle permet, en outre, de faciliter la gestion de projets complexes et dans le cas d'une application informatique le passage rapide vers la programmation car la plupart des langages informatiques utilisent un langage algorithmique.

Un algorithme **ne dépend pas**

- du langage terminal utilisé (souvent un langage informatique)
- de la technologie qui devrait exécuter l'enchaînement séquentiel des actions

**Un algorithme ne peut être écrit que si le problème à résoudre est parfaitement maîtrisé. De plus, une erreur dans un algorithme se traduira par une erreur dans le développement final ou dans la compréhension du produit qu'il décrit.**

# III/ Principes

Un algorithme :

- décrit une résolution de problème en mode pas à pas. A chaque pas, c'est la commande suivante qui est réalisée
- est délimité par les mots clés ***DEBUT*** et ***FIN***
- comporte des variables dont il faut définir le format (entier, caractère, chaîne de caractères...)
- se compose d'instructions basées sur des mots simples en général des verbes d'action (afficher, saisir,
- peut comporter des sous-problèmes décrits par d'autres algorithmes

## III.1/ Variables

Comme leur nom l'indique, à l'inverse des constantes, les variables sont amenées à être modifiées lors du traitement du problème.

En toute rigueur, il est nécessaire de spécifier le type des variables dans l'algorithme.

De manière courante, les types sont :

- **bit** : 0 ou 1, vrai ou faux
- **entier :** 1000, 255 par exemple
- **réel :** 10,125 par exemple

- **caractère :** b par exemple
- **chaîne de caractère :** algo25 par exemple

L'exemple précédent peut donc s'écrire :

```
Algorithme TVA

Variable prixHT, prixTTC : réels

DEBUT

afficher("Donner le prix du produit Hors-Taxes")

saisir(prixHT)

prixTTC   <-   prixHT*1,196

afficher("Le prix TTC est : ")

afficher(prixTTC)

FIN
```

## Affectation de valeur

Comme le montre l'exemple ci-dessus, l'affectation se décrit par une flèche pointant à gauche "<-".

Ainsi donc : **prixTTC <- prixHT *1,96** signifie :

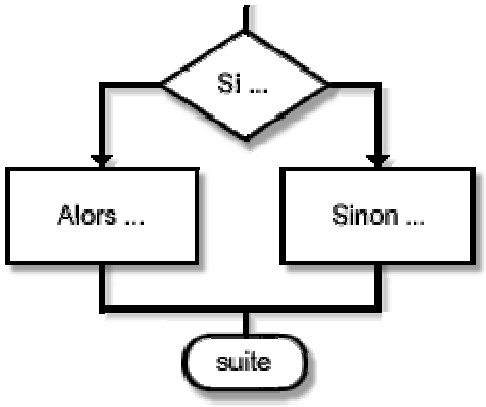
après exécution de l'opération, prixTTC vaudra 1,96 x prixHT

## III.2/ Structures conditionnelles

Les structures conditionnelles permettent de réaliser des opérations en fonction d'une condition.

### SI ALORS SINON

Une structure conditionnelle permet de traiter une opération ou une suite d'opérations selon le résultat d'un test logique.




La syntaxe normalisée est :

```
SI <condition logique>
```

```
        ALORS opération1
        SINON opération2
    FINSI
```

```
DEBUT
    afficher("Combien Arthur a-t-il mangé de pommes ?")
    saisir(nombre)
    SI nombre>1
    ALORS pomme <-"pommes."
    SINON pomme <- "pomme."
    FINSI
    afficher("ARTHUR a mangé "+ nombre + " "+ pomme)
FIN
```

**Remarque** : S'il n'y a pas d'opérations à réaliser en cas de réponse négative à la condition logique, il est inutile de mettre le bloc **SINON.**

### SELON...

```
SELON QUE variable vaut
  1:instruction1
  2: instruction2
  3: instruction3
  4: instruction4
FINSELON
```

Lorsque la condition ne peut pas avoir de réponse binaire on utilisera la structure - plus rare - ***SELON QUE***.

```
....
DEBUT
  afficher("Donner votre âge : ")
  saisir(age)
```

```
SELON QUE age vaut
    0..3 : afficher("Bébé")
    4..12 : afficher("Enfant")
    13..18 : afficher("Adolescent")
    19..150 : afficher("Adulte")
FINSELON
FIN
```

Dans l'exemple précédent, la variable ***age*** est un entier compris entre 0 et 150. Selon sa valeur, le message affiché sera différent.

On remarquera la notation utilisée pour définir un intervalle de valeurs.

# III.3/ Structures itératives - Boucles

Une ***structure itérative*** permet de répéter une ou plusieurs opérations en fonction d'une condition logique atteinte ou non.

## Comment choisir entre les trois structures proposées ?

```
SI nombre d'itérations connu
  ALORS choisir boucle POUR
  SINON
    SI traitement exécuté au moins une fois
      ALORS choisir boucle REPETER
      SINON choisir boucle TANT QUE
    FINSI
FINSI
```

## POUR... FAIRE

Cette structure permet de répéter une suite d'opérations un certain nombre de fois. Elle n'a de sens que si le nombre d'itération est connu à l'avance.

```
POUR <iteration> <- valeur_initiale A valeur_finale
  FAIRE instructions
FINPOUR
```

### TANTQUE.... FAIRE

Cette structure répète une suite d'action tant que la condition est vérifiée.

```
....
TANT QUE <condition>
  FAIRE
    instructions
    mettre à jour <condition>
  FINTQ
....
```

**REMARQUE :** si la condition est fausse dès le départ, les instructions ne seront jamais exécutées.

**ATTENTION :** il est indispensable que la condition soit mise à jour dans la boucle et que la condition finisse par ne pas être remplie sinon on ne sortira jamais de cette boucle.

### REPETER.... JUSQU'A

Cette structure répète une suite d'actions jusqu'à ce que la condition soit vérifiée.

```
...
REPETER
  instructions
  mettre à jour <conditions>
JUSQU'A  <condition>
```

**REMARQUE :** les instructions seront exécutées au moins une fois.

**ATTENTION :** il est indispensable que la condition soit mise à jour dans la boucle et que la condition finisse par être remplie sinon on ne sortira jamais de cette boucle.

## IV/ Conclusions

Pour réaliser un algorithme il faut :

- avoir une écriture **rigoureuse**
- avoir un souci de présentation et **indenter**, comme dans les exemples de ce cours, les lignes

- **commenter**

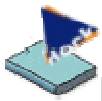
Il existe plusieurs manières de résoudre un problème, aussi faut-il toujours rechercher l'**efficacité** de l'algorithme.

# V/ Aller plus loin... Procédures et fonctions

Un algorithme, on l'a vu permet de décrire un problème simple.

Un problème compliqué n'est le plus souvent qu'une association de problèmes simples.

Les langages de développement informatique reposent généralement sur une programmation procédurale.



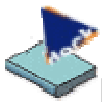
## Programmation procédurale

Avec des langages tels que le C ou le Pascal, la résolution d'un problème informatique passe généralement par l'analyse descendante qui consiste à décomposer un problème en sous-problème jusqu'à descendre à des actions primitives.
On décompose ainsi un programme en un ensemble de sous-programmes appelés procédures qui coopèrent pour la résolution d'un problème.

Les procédures et fonctions sont généralement des outils qui produisent et/ou modifient des données.

Ainsi :

**PROGRAMME = ALGORITHME + DONNEES**



## Procédure

Une procédure réalise des actions dans un programme qui apportent une modification dans les variables d'entrée-sortie.

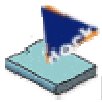
La procédure ne restitue pas de valeur.

La procédure peut recevoir des variables en paramètre.

**Exemple :**

*procedure Tform1.Afficherheure(sender:Tsender);*

Cette procédure affiche l'heure à l'écran



## Fonction

A la différence de la procédure, la fonction restitue une valeur.

**Exemple**

**Déclaration de la fonction**

function TForm1.carre(valeur:integer);

**Utilisation de la fonction :**

*y:=carre(2);*

*y* vaudra 4 après exécution si la fonction carré à pour objectif d'élever valeur au carré.