UNIVERSITE DE DJIBOUTI

FACULTE DE SCIENCES

DEPARTEMENT INFORMATIQUES

COURS D'INTRODUCTION A L'ALGORTIHME

2022-2023

❖ Planning de l'UE

Semestre étudié	Titre des Chapitres
S1	**Chapitre 1** : **Introduction à l'algorithme** ***a- Structure general d'un algorithme*** ***b- Algorithme sequentiel***
	Chapitre 2 : **Structure Conditionnelle**
	Chapitre 3 : **Structure répétitive**
	Chapitre 4 : **Tableau à un dimension**

❖ Objectif général de l'UE :

Obtenir de la machine qu'elle effectue un travail à notre place :
Expliquer à la machine comment elle doit s'y prendre
Savoir écrire un algorithme complet

✓

Objectif de l'enseignement :

- ✓ Mémoriser les notions essentielles en algorithme
- ✓ Ordonner la méthodologie d'écriture de l'algorithme
- ✓ Maitriser 2 types de structures (structure conditionnelle et structure répétitive)
- ✓ Déduire le tableau à une dimension

❖ Pré-requis

- ✓ Etre rigoureux
- ✓ Avoir une certaine intuition

Sommaire générale

Chapitre1 : INTRODUCTION À L'ALGORITHME

I- *Notion d'un programme*

Un programme est un assemblage et un enchaînement d'instructions élémentaires écrit dans un langage de programmation, et exécuté par un ordinateur afin de traiter les données d'un problème et renvoyer un ou plusieurs résultats.

Ainsi pour programmer un programme il faudra utiliser un moyen qui nous permettra d'écrire nos code en un langage compréhensible par un ordinateur. Alors une question simple qui est qu'est ce qu'un ordinateur ? Qu'est ce qu'il fait exactement ? Comment sa fonctionne ?

Un ordinateur est une machine sans aucune forme d'intelligence qui sert à programmer des traitements d'information codée en format binaire. L'ordinateur est la machine qui permet de traiter l'information représentée sous sa forme binaire. Il travaille donc sur des données codées de façon BINAIRE. Chaque variable binaire peut être représentée par deux états 0 et 1.

Pourtant, contrairement aux autres machines qui sont dédiées à un nombre limité de tâches, l'ordinateur est potentiellement capable d'effectuer une infinité de tâches concernant le traitement rationnel de l'information. On dit que c'est une machine universelle.

Alors comment une machine stupide peut traiter autant de problèmes différents?

C'est que, grâce aux actions de base qu'elle sait réaliser, il est possible en les assemblant de façon pertinente, de résoudre la plupart des problèmes concernant le traitement de l'information. Il suffit de lui indiquer l'ordre dans lequel il faut qu'il effectue ces actions basiques et avec quelles données. Ces ordres élémentaires sont appelés instructions et sont rassemblées au sein d'un programme. Comme l'ordinateur a l'avantage d'exécuter très rapidement et sans erreurs les ordres qu'on lui donne (les instructions), il exécute beaucoup de traitements complexes plus vite et plus sûrement qu'un homme. Pour donner des ordres à l'ordinateur, il est nécessaire de pouvoir communiquer avec lui. Cette communication passe par un langage de programmation, dans lequel est écrit le programme.

II- Notion d'un algorithme:

A. Définition

Un algorithme un ensemble de règles opératoires dont l'enchaînement permet de RESOUDRE un problème au moyen d'un nombre FINI d'actions ou opérations (instructions) .

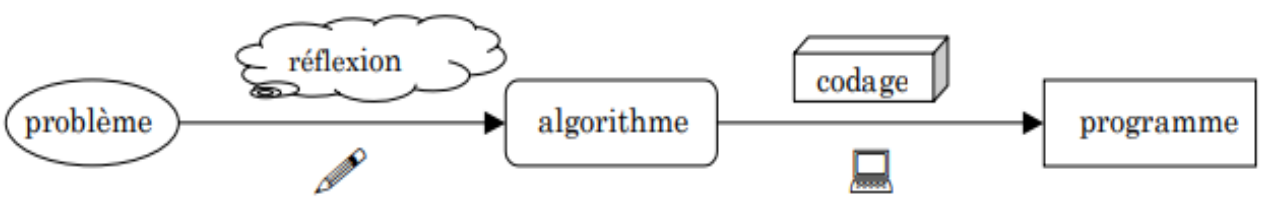
L'utilisateur d'un algorithme n'aura qu'à suivre toutes les instructions, dans l'ordre pour arriver au résultat que doit donner l'algorithme

Autre définition :

Un algorithme représente l'enchaînement des actions (instructions) nécessaires pour faire exécuter une tâche à un ordinateur (résoudre un problème)

Un algorithme n'est donc exécutable directement par aucune machine. Mais il a l'avantage d'être traduit facilement dans tous les langages de programmation.

Pour résoudre un problème, il est vivement conseillé de réfléchir d'abord à l'algorithme avant de programmer proprement dit.



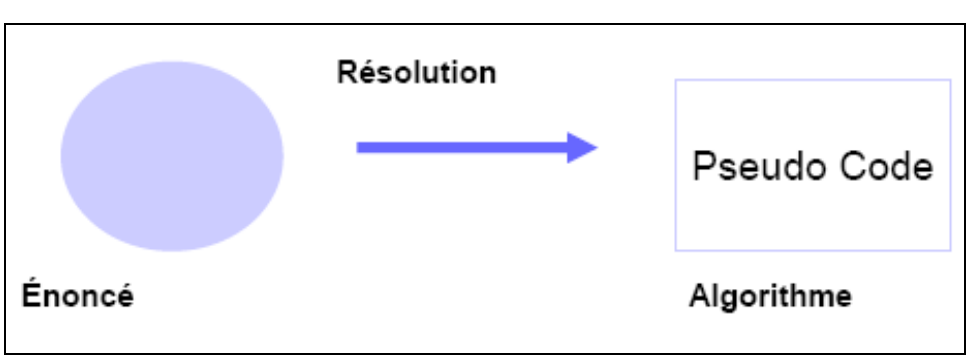
❖ L'expression d'un algorithme nécessite un langage :

- ✓ clair (compréhension)
- ✓ structuré (enchaînements d'opérations)
- ✓ universel (indépendants du langage de programmation choisi)

❖ Quel langage utilise l'algorithme ?

Un algorithme s'écrit le plus souvent en pseudo-langage de programmation (*appelé langage algorithmique*). C'est un langage naturel qui utilise le langage français mais qui suit un certain ordre d'instruction pour arriver à un résultat fini.

- ✓ L'algorithme suit une règle syntaxique prédéfinie
- ✓ c'est écrire en français ce qu'il faut faire



Le "langage algorithmique" que nous utilisons est un compromis entre un langage naturel et un langage de programmation

Exemple

Énoncé: un algorithme qui demande à un utilisateur un nombre et qui l'affiche si ce dernier est un nombre pair ou impaire.

Donnée : un nombre

Afficher : Un nombre paire ou impaire

Traitement : expliquer en détail comment trouver un nombre paire ou impaire

- *A retenir*
- *Ce n'est pas un langage comme les autres*
- *Il n'est donc exécutable directement par aucune machine.*
- *Il faudra le traduire dans un langage compréhensible par l'ordinateur*

B. Formalisation d'un algorithme

Un algorithme doit être lisible et compréhensible par plusieurs personnes.
Il doit donc suivre des règles. Il est composé d'une entête et d'un corps. L'entête comprend :

- Nom : le nom de l'algorithme
- Rôle : ce que fait l'algorithme
- Données : les données fournies à l'algorithme
- Résultat : ce que l'on obtient à la fin du traitement - Principe : le principe utilisé dans l'algorithme

Le corps :

- il est délimité par les mots clés début et fin.
- il se termine par un lexique, décrivant les variables utilisées

Ainsi, tous les identifiants de variables seront notés en minuscule ou en majuscule et auront un nom significatif et qui explicite sur son rôle

Exemple :

Nom : AddDeuxEntiers.
Rôle : additionner deux entier et mémoriser le résultat
Données : les valeurs à additionner.
Résultat : la somme des deux valeurs.
Principe : Additionner deux entiers a et b et mettre le résultat dans c.

****

```
Algorithme Somme
a : entier
b : entier
 c : entier
```

```
début
a←5 ;
b←3 ;

 c ← a + b ;
Afficher (c) ;

 fin
```

Dans cet exemple simple apparaissent les trois étapes qui caractérisent la résolution d'un problème sur ordinateur:

-Comprendre la nature du problème posé et préciser les données fournies ("entrées" ou "input" en anglais)

-Préciser les résultats que l'on désire obtenir ("sorties" ou "output" en anglais)

-Déterminer le processus de transformation des données en résultats.

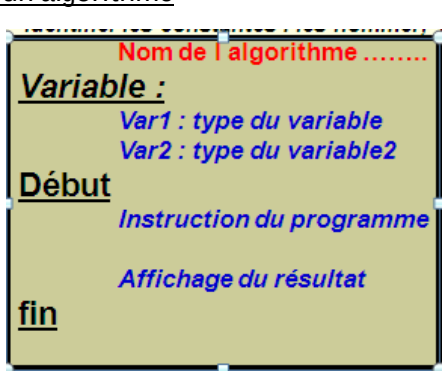
❖ Construction d'un algorithme simple

Construire un algorithme consiste par convention :

- ✓ *à lui donner un nom,*
- ✓ *identifier les constantes : les nommer, les initialiser et indiquer leurs types,*
- ✓ *identifier les variables : les nommer et indiquer leurs types,*
- ✓ *écrire le corps de l'algorithme encadré par les mots clé « début » et « fin ».*

Le corps de l'algorithme contiendra les opérations de *lecture* et d'*écriture* des *variables* qui le nécessitent. L'usage veut que l'on indente toute séquence d'instructions en décalant le début de chaque ligne pour faciliter la compréhension de l'algorithme.

❖ Comment ressemble un algorithme



III- Notion de variables et déclarations

A. Les variables

Déclaration

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. Il peut aussi s'agir de résultats obtenus par le programme, intermédiaires ou définitifs. elles peuvent être des nombres, du texte. Toujours est-il que dès que l'on a besoin de stocker une information dans un programme, on utilise une variable**.**

Une variable est une entité qui contient une information, elle possède :

- ✓ un nom, on parle d'identifiant
- ✓ une valeur
- ✓ un type qui caractérise l'ensemble des valeurs que peut prendre la variable

L'ensemble des variables est stocké dans la mémoire de l'ordinateur

Exemple :

```
Toto : entiers ;

Tata : caractère ;
```

En gros, une variable est comme une boîte, repérée par une étiquette. Pour avoir accès au contenu de la boîte, il suffit de la désigner par son étiquette.

La première chose à faire avant de pouvoir utiliser une variable est de créer la boîte et de lui coller une étiquette. Ceci se fait tout au début de l'algorithme, avant même les instructions proprement dites. C'est ce qu'on appelle la déclaration des variables.

❖ Déclaration des variables

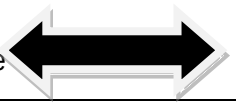
Que veut dire déclarer une variable ? En faite le programme réserve un espace dans la mémoire et lui attribue un identificateur à cet espace. Mais toutes les variables n'ont pas besoin de la même place en mémoire. Un grand nombre prend plus de place qu'un caractère. Selon le type de l'objet, il faudra lui réserver plus ou moins de place: c'est pourquoi il faut déclarer le type des variables et pas seulement leur nom. Par ailleurs, selon le type des variables, les opérations possibles seront différentes. Donc la déclaration d'une variable indique deux choses:

- son **identificateur** (son nom)

- son **type** (sa taille)

Comment déclarer une variable ?

Pour déclarer une variable, il faudra choisir un nom (identificateur) puis un type pour définir dans la mémoire quel donnée doit recevoir cette espace.

Nom-de-la-variable : Type  Nombre : entier

Exemple :

Alors combien de type existe-t-il ?

En algorithmique, on distingue 5 types principaux:

- les caractères (lettres, chiffres, ponctuation, code des opérations, espace, retour chariot,... et plus généralement toutes les touches que l'on peut trouver sur une machine à écrire)

- les chaînes de caractère (suites de caractères)

- les entiers (les nombres sans virgule)

- les réels (les nombres à virgule et sans virgule)

- les booléens (qui n'ont que deux valeurs possibles: soit VRAI, soit FAUX)

1. Type des variables

Il ne suffit pas de créer une boîte (réserver un emplacement mémoire) ; encore doit-on préciser ce que l'on voudra mettre dedans, car de cela dépendent la taille de la boîte (de l'emplacement mémoire) et le type de codage utilisé.

1.1 Types numériques

Commençons par le cas le plus fréquent, celui d'une variable destinée à recevoir des nombres.

>>>> Entier simple, entier long , réel simple, réel double ;

Mais généralement nous déclarons les valeurs numériques soit en entier soit en réel sans différencié.

Exemple 2 :

Variable g en Entier

Variables PrixHT, TauxTVA, PrixTTC en Réel

Remarque :

Certains langages autorisent d'autres types numériques, notamment :

>Le type monétaire (avec strictement deux chiffres après la virgule)

>Le type date (jour / mois / année).

1.2 Types non numériques

On dispose donc également du type alphanumérique (*également appelé type caractère*) : dans une variable de ce type, on stocke des caractères, qu'il s'agisse de lettres, de signes de ponctuation, d'espaces, ou de chiffres. Le nombre maximal de caractères pouvant être stockés dans une seule variable dépend du langage utilisé.

Il y a aussi les chaînes de caractère car une telle chaîne de caractères est toujours notée entre guillemets.

Remarque :

Un autre type est le type booléen : on y stocke uniquement les valeurs logiques VRAI et FAUX.

2. L'instruction d'affectation

o Syntaxe et signification

Une fois qu'on a vue les variables nous allons voir les affectations car La seule chose qu'on puisse vraiment faire avec une variable, c'est de l'affecter, c'est-à-dire lui attribuer une valeur. En algorithmique, cette instruction se note avec le signe ← .

Ainsi : Toto ←24

On peut attribuer à une variable la valeur d'une autre variable, telle quelle ou modifiée. Par exemple :

Tutu ← Toto

Signifie que la valeur de Tutu est maintenant celle de Toto.

Notez que cette instruction n'a modifié en rien la valeur de Toto : une instruction d'affectation ne modifie que ce qui est situé à gauche de la flèche.

Tutu ←Toto + 4

Si Toto contenait 12, Tutu vaut maintenant 16. De même que précédemment, Toto vaut toujours 12.

❖ Ordre des instructions

Il va de soi que l'ordre dans lequel les instructions sont écrites va jouer un rôle essentiel dans le résultat final.

3. Expressions et opérateurs

Si on fait le point, on s'aperçoit que dans une instruction d'affectation, on trouve :

- à gauche de la flèche, un nom de variable, et uniquement cela.
- à droite de la flèche, ce qu'on appelle une expression. C'est-à-dire un ensemble de valeurs liées par des opérateurs, et dont le résultat final est obligatoirement du même type que la variable située à gauche.

❖ *Définition :*

Un opérateur est un signe qui peut relier deux valeurs, pour produire un résultat. Les opérateurs possibles dépendent du type des valeurs qui sont en jeu

❖ Opérateurs numériques :

Ce sont les quatre opérations arithmétiques tout ce qu'il y a de classique.

+ Addition
- soustraction
* multiplication
/ division

Mentionnons également le ^ qui signifie " puissance ". 45 au carré s'écrira donc 45 ^ 2.

❖ Opérateur alphanumérique : &

Cet opérateur permet de concaténer, autrement dit d'agglomérer, deux chaînes de caractères.

Exemple

Variables A, B, C en Caractère

Début

A ←"toto"

B ←"ettata"

C ← A & B

```
Fin

La valeur de C à la fin de l'algorithme est "totoettata"
```

❖ Opérateurs logiques :

Il s'agit du ET, du OU et du NON. Nous les laisserons de côté… provisoirement, soyez-en sûrs.

Deux remarques pour terminer :

✓ J'attire votre attention sur la trompeuse similitude de vocabulaire entre les mathématiques et l'informatique. En mathématiques, une " variable " est généralement une inconnue. Lorsque j'écris que y = 3 x + 2, les " variables " x et y satisfaisant à l'équation existent en nombre infini (graphiquement, l'ensemble des solutions à cette équation dessine une droite). Lorsque j'écris $ax^2 + bx + c = 0$, la " variable " x désigne les solutions à cette équation, c'est-à-dire 0, une ou deux valeurs à la fois… En informatique, une variable a toujours une valeur et une seule. A la rigueur, dans certains langages, mais pas tous, elle peut ne pas avoir de valeur du tout (tant qu'on ne l'a pas affectée). Dans les autres langages, les variables non encore affectées sont considérées comme valant zéro. Et cette valeur justement, ne " varie " pas à proprement parler (du moins ne varie-t-elle que lorsqu'elle est l'objet d'une instruction d'affectation).

✓ La deuxième remarque concerne le signe de l'affectation. En algorithmique, comme on l'a vu, c'est le ←. Mais en pratique, la quasi totalité des langages emploient le signe égal. Et là, pour les débutants, la confusion est facile avec les maths. En maths, A = B et B = A sont deux propositions strictement équivalentes. En informatique, absolument pas, puisque cela revient à écrire A ← B et B ←A, deux choses bien différentes. Donc, attention ! ! !

IV. Lecture et écriture

```
Exemple 3 :

Variable A : Numérique

Début

A ← 12^2

Fin
```

L'inconvénient c'est que Ce gentil programme nous donne le carré de 12. Mais si l'on veut le carré d'un autre nombre que 12, il faut réécrire le programme.

Dans un sens, cela permet à l'utilisateur de rentrer des valeurs au clavier pour qu'elles soient utilisées par le programme. Cette opération est **l'écriture**.

Dans l'autre sens, cela permet au programme de communiquer des valeurs à l'utilisateur en les affichant à l'écran. Cette opération est **lecture**.

Ecrire ("Entrez votre nom : ") ;

Lire (NomFamille) ;

❖ Instruction d'écriture

Cette instruction permet de restituer une valeur. Généralement, ça consiste à afficher sur l'écran.

Donc on a :

1. Ecrire (valeur)

   Ex: Ecrire (4)

2. Ecrire (variable)

   Ex: Ecrire(Note)

3. Ecrire (expression)

   Ex: Ecrire ('La moyenne=', (Note1+Note2)/2)

Remarque: *Ecrire(Note) n'est pas la même chose que Ecrire('Note')*

Lorsque nous utilisons l'instruction écrire, cela veut dire que soit il faudra afficher un message à l'écran pour que l'utilisateur puisse comprendre ce qu'on attend de lui soit afficher à l'écran le résultat du programme.

❖ Instruction de lecture

Cette instruction permet d'introduire une donnée au programme. Généralement, on tape la valeur sur le clavier

Donc on a :

1. Lire(variable)

   Ex: Lire(Note)

- Effet:

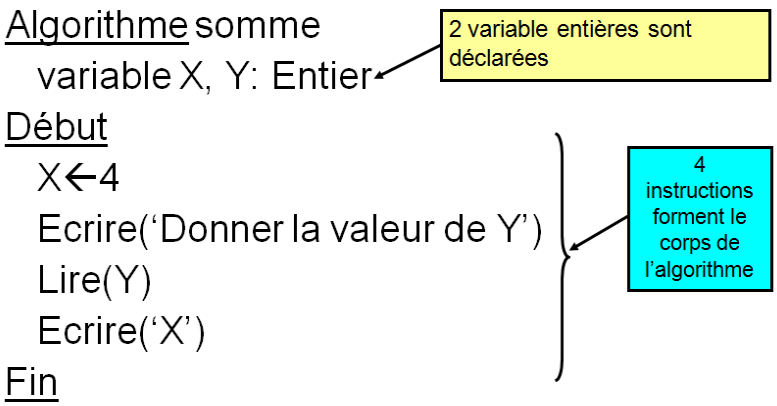
– à la rencontre de cette instruction, l'ordinateur arrête l'exécution du programme et attend qu'on tape une valeur.

– On termine la saisie en appuyant sur la touche Entrée.

– La valeur qu'on tape est affectée à la variable lue

Remarque: Lire(valeur) et Lire(expression) n'ont pas de sens

Lorsque nous utilisons l'instruction écrire cela veut dire qu'on attend de l'utilisateur qu'il rentre une valeur qui sera affecté à un variable puis il sera réutilisé afficher à l'écran le résultat du programme



Fin du chapitre 1

Chapitre 2 : LES STRUCTURES CONDITIONNELLES

Introduction :

Un algorithme doit résoudre des problèmes très divers. En effet on est amené souvent à faire des choix pour prendre des décisions.

Définition :

La structure conditionnelle permet à un programme de réaliser un traitement en fonction d'une condition, il existe trois formes de structures conditionnelles :

- ✓ Structure simple
- ✓ Structure imbriquée
- ✓ Structure de choix

I- Structure simple

1 Forme simple réduite:

Si l'élève aura une moyenne annuelle générale >= 10 Alors il passera à l'année suivante.

Si ..... Alors ..... Finsi

Syntaxe : Analyse et Algorithme

```
SI Condition Alors
Instruction 1
Instruction 1
...
Instruction 1
FinSi
```

Remarque:

Si la condition est Vrai alors l'instruction vrai est exécuté et si la condition est Faux alors il n'y rien qui sera affiché.

2 Forme simple complète:

Si il fait beau Alors j'irai au Cinéma Sinon je reste à la maison.

Si ..... Alors ..... Sinon ..... Finsi

```
Syntaxe: Analyse et Algorithme
SI Condition Alors
Instruction 11
Instruction 12
...
Instruction 1N
Sinon
Instruction 21
Instruction 22
...
Instruction 2N
FinSi
```

Remarque:
Si la condition est Vrai alors l'instruction vrai est exécuté et si la condition est Faux alors l'instruction fausse est exécuté.

Activité02 :
Le programme suivant permet de saisir les moyennes des deux semestres puis calcule la moyenne générale d'un élève, ensuite il affiche sa moyenne et la décision (REDOUBLE ou ADMIS) relative à sa moyenne.

```
Algorithme:
0) Début
1) Ecrire ("donner la moyenne de Semestre1") , lire (.....) ;
2) Ecrire ("donner la moyenne de Semestre2") , lire (.....) ;
3) Moy ??( MT1 + MT2 ) / 2 ;
4) SI( Moy ≥ 10 ) Alors
D ??"Admis." ;
Sinon
D ??"Redouble" ;
FinSi
5) Ecrire(" La moyenne =", Moy, " La décision :", D) ;
6) Fin.
```

II- Structure imbriqué

On fait appel à cette forme lorsque le problème présente plus que deux cas de traitements possibles.

```
Si condition_1  Alors
                Traitement_1
        Sinon Si condition_1  Alors
                        Traitement_2
                ...
                                Sinon
                                Traitement_n
Fin si
```

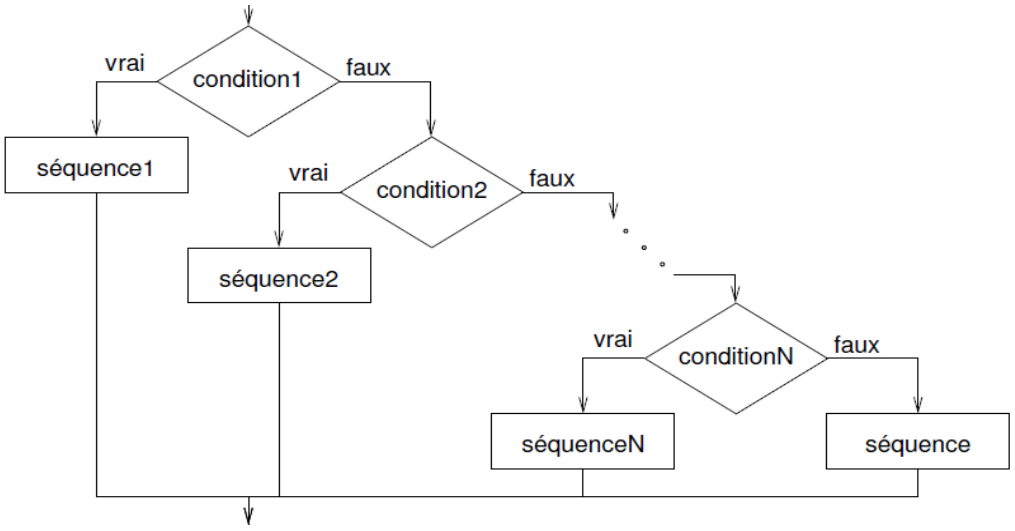
```
If condition_1  then
        BEGIN  Traitement_1;  END
        else if condition_2 then
        BEGIN  Traitement_2 ;  END
        ...
        else
        BEGIN  Traitement_n;  END;
```

La conditionnelle **Si ... Alors ... Sinon ... FinSi** peut être complétée avec des clauses **SinonSi** suivant le schéma suivant :

```
1     Si condition_1 Alors
2         séquence_1
3     SinonSi condition_2 Alors
4         séquence_2
5     ...
6     SinonSi condition_N Alors
7         séquence_N
8     Sinon       { Expliciter la condition ! }
9         séquence
10    FinSi
```

Évaluation : Les conditions sont évaluées dans l'ordre d'apparition. Dès qu'une condition est vraie, la séquence associée est exécutée. L'instruction suivante à exécuter sera alors celle qui suit le **FinSi**. Si aucune condition n'est vérifiée, alors la séquence associée au **Sinon**, si elle existe, est exécutée.

Organigramme :



Étant donné un entier lu au clavier, indiquer s'il est nul, positif ou négatif.

```
Si n > 0 Alors
14 | | Écrire("positif");
15 | SinonSi n < 0 Alors
16 | | Écrire("positif");
17 | Sinon { Non (n > 0) Et Non (n < 0) donc N = 0 }
18 | | Écrire("nul");
19 | FinSi
```

Activité03 :

Faire l'analyse, l'algorithme et la traduction langage c d'un programme intitulé CUBIQUE qui saisit un entier X et vérifier si l'entier saisie est cubique ou non. Un entier est dit cubique s'il est égal à la somme des cubes de ses chiffres.

Exemple : 153 est un entier cubique car

153= 13+53+33

III- Structure de contrôle conditionnelle à choix:

La structure de choix permet de faire un choix parmi plusieurs possibilités. Le choix du traitement à effectuer dépend de la valeur que prendra un sélecteur. Ce sélecteur est une variable, cette variable est comparée à une série de valeurs ou à un ou plusieurs intervalles. En cas d'égalité, l'instruction qui lui est associée est exécutée. Les autres ne seront pas exécutées.

a- Syntaxe :

```
Selon expression Dans
choix1 :
séquence1
choix2 :
séquence2
...
choixN :
séquenceN
Sinon
séquence
FinSelon
```

b- Règles :

– expression est nécessairement une expression de type scalaire.

Cours Algo, Semaine 1 c INPT–PAD 22/32

ALGORITHMIQUE ET PROGRAMMATION 1 Algorithmique et programmation : les bases (Algo)

– choixi est une liste de choix séparés par des virgules. Chaque choix est soit une constante, soit un intervalle (10..20, par exemple).

L'instruction Selon peut donc être considérée comme un cas particulier de la conditionnelle

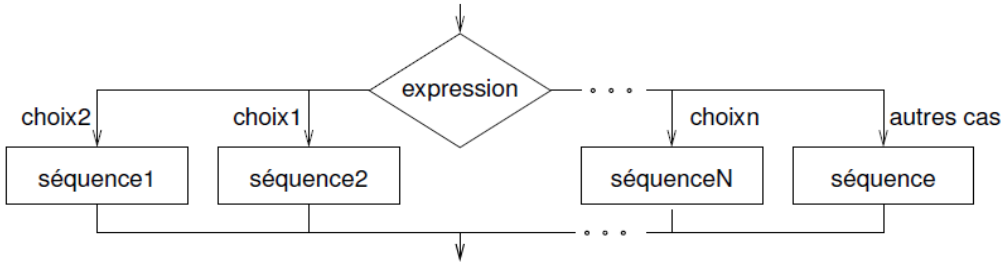
```
Au niveau de l'analyse et l'algorithme
[init] Selon sélecteur Faire
Valeur 1 : Action 1
Valeur 2 : Action 2
Action 2-1
...
Action 2-N
Valeur 3 : Action 3
Valeur 4, Valeur 6, Valeur 8 : Action 4
Valeur 5, Valeur 7, Valeur 9 : Action 5
Valeur 10 .. Valeur 19 : Action 6
...
Valeur N : Action N
SINON
Action R
Fin Selon
```

Si ... SinonSi ... FinSi.

Évaluation : L'expression est évaluée, puis sa valeur est successivement comparée à chacun des ensembles choix$_i$. Dès qu'il y a correspondance, les comparaisons sont arrêtées et la séquence associée est exécutée. Les différents choix sont donc *exclusifs*. Si aucun choix ne correspondant, alors la séquence associée au **Sinon**, si elle existe, est exécutée.

Remarque : La clause **Sinon** est optionnelle... mais il faut être sûr de traiter tous les cas (toutes les valeurs possibles de l'expression).

Organigramme :



Solution :

```
Algorithme repondre
-- Répondre par « affirmatif », « négatif » ou « ?!?!?!? ».
Variable
reponse: Charactere -- caractère lu au clavier
Début
-- saisir le caractère
Écrire("Votre réponse (o/n) : ")
Lire(reponse)
-- afficher la réponse
Selon reponse Dans
'o', 'O': { réponse positive }
Écrireln("Affirmatif !")
'n', 'N': { réponse négative }
Écrireln("Négatif !")
Sinon
Écrireln("?!?!?!?")
FinSelon
```

Fin du chapitre 2

Chapitre 3 : Structures répétitives

Les structures répétitives aussi appelées boucles, permettent de répéter un traitement (c'est à dire une instruction simple ou composée) autant de fois qu'il est nécessaire: soit un nombre déterminé de fois, soit tant qu'une condition est vraie.

Il existe trois grands types principaux de structures répétitives:

- la structure Tant que...Faire, qui permet d'effectuer une instruction tant qu'une condition est satisfaite
- la structure Pour qui permet de répéter une instruction un certain nombre de fois
- la structure Répéter...Jusqu'à, qui comme son nom l'indique, permet de répéter une instruction jusqu'à ce qu'une condition soit satisfaite.

Seule la boucle Tant que est fondamentale. Avec cette boucle, on peut réaliser toutes les autres boucles alors que l'inverse n'est pas vrai. La boucle Pour est très utilisée aussi car elle permet de simplifier la boucle Tant que lorsque le nombre de tour de boucle est connu d'avance. La boucle Répéter, très peu utilisée, sera étudiée vers la fin de ce chapitre.

A. LA BOUCLE TANT QUE ... FAIRE

La boucle Tant que ... Faire permet de répéter un traitement tant qu'une expression conditionnelle est vraie. Si d'emblée, la condition n'est pas vraie, le traitement ne sera pas exécuté. On voit donc que la boucle Tant que a un point commun avec la structure conditionnelle où si la condition n'est pas vraie, le traitement n'est pas exécuté.

Syntaxe:

Tant que <condition d'exécution> Faire

    <traitement> // instruction simple ou bloc d'instructions

FinTantque

Supposons que l'on veuille que l'algorithme calcule le cube des nombres qu'on lui fournit et que pour arrêter, l'utilisateur doive entrer 0. Si le nombre saisi est 0, on ne veut pas afficher le cube et le traitement est terminé. Si le nombre saisi est différent de 0, on affiche son cube et on recommence (on demande d'entrer un nombre, on le saisit, etc). On veut donc exécuter les instructions dans l'ordre suivant:

- saisir un nombre
- vérifier la condition d'exécution ($x \neq 0$)
- si x vaut 0, on sort de la boucle sinon on affiche le cube et on attend que l'utilisateur entre un autre nombre
- On vérifie la condition d'exécution ($x \neq 0$)
- si x vaut 0, on sort de la boucle sinon on affiche le cube et on attend que l'utilisateur entre un autre nombre

...

On voit donc qu'après la saisie du premier nombre, on répète les trois dernières instructions. On va donc pouvoir les inscrire dans une boucle. La condition de continuation (x . 0) est inscrite après le tant que. Cette condition est vérifiée à chaque fois qu'on a terminé les traitements de la boucle.

```
Programme cube ;

Var x : Entier

Début

    Ecrire ("Ce programme calcul le cube des nombres que vous entrez. Pour
arrêter tapez 0.") ;

    Ecrire ("Entrez un nombre") ;

    Lire (x) ;
```

Le nombre de répétition du traitement n'est pas indiqué explicitement; il dépendra des données fournies au programme, en l'occurrence les nombres entrés.

Fonctionnement de ce programme

Il affiche tout d'abord le libellé de saisie et attend que l'utilisateur entre un nombre, qui est alors saisi dans la variable x. Ensuite, la condition qui suit le Tant que est évaluée. Si l'utilisateur rentre comme premier nombre 0, la condition est fausse et le corps de la boucle ne sera pas exécuté et le processeur continuera à la première instruction suivant le FinTQ (Afficher "Fin"). Si l'utilisateur entre un nombre différent de 0, son cube est calculé et affiché et un nouveau nombre est saisi. Au niveau du FinTQ, le processeur effectue un branchement, c'est à dire qu'il n'effectue pas l'instruction suivante mais retourne au début de la boucle et réévalue l'expression conditionnelle. L'utilisateur peut calculer autant de cubes qu'il désire et quand il veut arrêter, il lui suffit de taper 0. On dit que 0 est une valeur drapeau, c'est-à-dire une valeur qui indique la fin d'un traitement.

La trace d'un algorithme

La trace d'un algorithme représente la valeur des différentes informations d'un programme durant son exécution. Il est vivement conseillé d'effectuer la trace d'un algorithme afin de vérifier qu'il fonctionne. La première chose à faire est de choisir des données sur lesquelles ont va effectuer le test de l'algorithme. Pour ces données, on calcule à la main le résultat attendu. Puis on effectue la trace et on compare le résultat attendu avec le résultat de la trace qui doivent être les mêmes (sinon, il y a une erreur quelque part…).

Effectuons la trace de l'algorithme précédent avec les données suivantes

donnée x	résultat attendu
10	affichage de 100
-3	affichage de -9
0	affichage de Fin et arrêt du programme

B. LA BOUCLE POUR

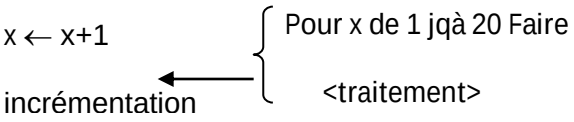
La boucle Pour permet de répéter une instruction un nombre connu de fois. Elle a le formalisme suivant:

Syntaxe :

```
Pour < compteur> de <valeur initiale> jqà <valeur finale> [pas de <incrément>] Faire
        <traitement>
FinPour
```

Elle permet de faire la même chose que la boucle Tant que mais de façon plus rapide, du moins lorsque le nombre de répétition est connu. La variable compteur est de type entier. Elle est initialisée à la valeur initiale. Le compteur augmente (implicitement) de l'incrément à chaque répétition du traitement. Lorsque la variable compteur vaut la valeur finale, le traitement est exécuté une dernière fois puis le programme sort de la boucle. Par défaut, l'incrément est de 1

Exemple:




```
Pour x de 1 jqà 20 Faire
    <traitement>
FinPour
```

Grâce à une telle structure, le traitement va être répétée 20 fois. On pourrait faire la même chose avec une boucle tant que, mais il faudrait initialiser la variable compteur et l'incrémenter explicitement.

```
X ← 1
Tant que x <= 20 Faire
        <traitement>
        x ← x+1
FinTantQue
```

La boucle Pour est en fait une simplification de la boucle TantQue.

Application

Affichons la table de multiplication du 7. Pour cela on va utiliser une variable a qui varie de 1 à 10 et multiplier cette variable par 7 à chaque incrémentation. Cette variable va aussi servir de compteur pour la structure Pour.

```
Programme multiplication7 ;

Var a : Entier ;

Début

    Pour a de 1 à 10 faire

        Ecrire ( a, " * 7 = ", a * 7) ;

    FinPour

Fin.
```

C. LA BOUCLE REPETER…JUSQU'A

Cette boucle sert à répéter une instruction jusqu'à ce qu'une condition (expression booléenne) soit vraie. Son formalisme est le suivant:

Syntaxe

```
Répéter

    traitement // une instruction simple ou un bloc d'instructions

Jusqu'à condition d'arrêt
```

Le traitement est exécuté, puis la condition est vérifiée. Si elle n'est pas vraie, on retourne au début de la boucle et le traitement est répété. Si la condition est vraie, on sort de la boucle et le programme continue séquentiellement. A chaque fois que le traitement est exécuté, la condition d'arrêt est de nouveau vérifiée à la fin.

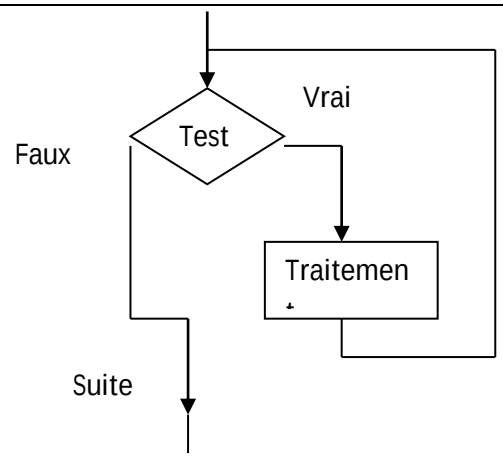
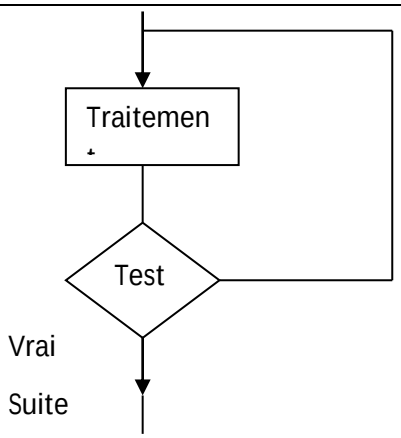
Exemple

```
Programme Aire ;

Var rayon : réel ;

    réponse : chaîne ;

Début

    Ecrire ("Calcul de l'aire d'un cercle") ;

    Répéter

        Ecrire ("Entrez le rayon d'un cercle en cm") ;

        Lire (rayon) ;

        Ecrire ("L'aire de ce cercle est ", rayon*rayon *3.14, "cm²") ;

        Ecrire (Voulez-vous l'aire d'un autre cercle? (oui/non)")
```

```
        Lire ( réponse) ;
        Jusqu'à (réponse . "oui" //si la réponse est différente de "oui", la répétition du
         traitement s'arrête) ;
        Ecrire ( "Au revoir!") ;
Fin
```

La boucle Répéter n'est pas indispensable. Elle peut toujours être remplacée par une boucle Tant que. C'est pourquoi certains langages n'ont pas d'équivalent pour la boucle Répéter.

Différences entre la boucle Répéter et Tant que

Tant que…Faire	Répéter… Jusqu'à
Condition vérifiée avant le traitement: le traitement peut ne pas être exécuté	Condition vérifiée après le traitement: le traitement est forcément exécuté une fois
condition de continuation : le traitement est répété si la condition est vraie	condition d'arrêt : le traitement est répété si la condition est fausse
	

Chapitre 4 : Tableau à une dimension

Exemple introductif

Supposons qu'on veut conserver les notes d'une classe de 12 étudiants pour extraire quelques informations. Par exemple : calcul du nombre d'étudiants ayant une note supérieure à 10

Le seul moyen dont nous disposons actuellement consiste à déclarer 12 variables, par exemple N1, …, N12. Après 30 instructions lire, on doit écrire 30 instructions Si pour faire le calcul

Exemple calcul de la moyenne

- Moy ← (N1+N2+N3+N4+N5+N6+N7+N8+N9+N10+N11+N12)/12

```
nbre ← 0
Si (N1  >10) alors nbre ←nbre+1 FinSi
….
Si (N12>10) alors nbre ←nbre+1 FinSi
```

C'est lourd à écrire, Heureusement, les langages de programmation offrent la possibilité de rassembler toutes ces variables dans une seule structure de donnée appelée tableau

- rassembler toutes ces variables en une seule, au sein de laquelle chaque valeur sera désignée par un numéro

1. Définition

Un tableau est une structure de données permettant d'effectuer un même traitement sur des données de même nature.

tableau à **une** dimension

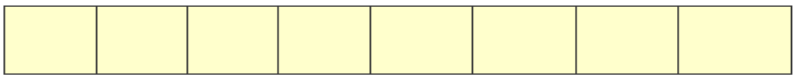
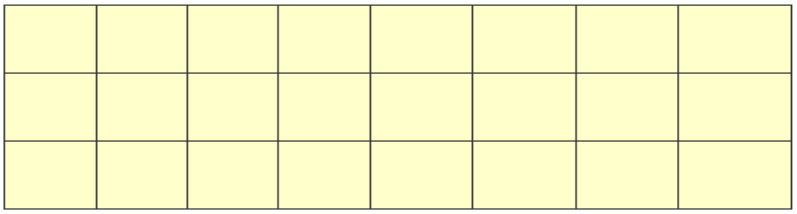
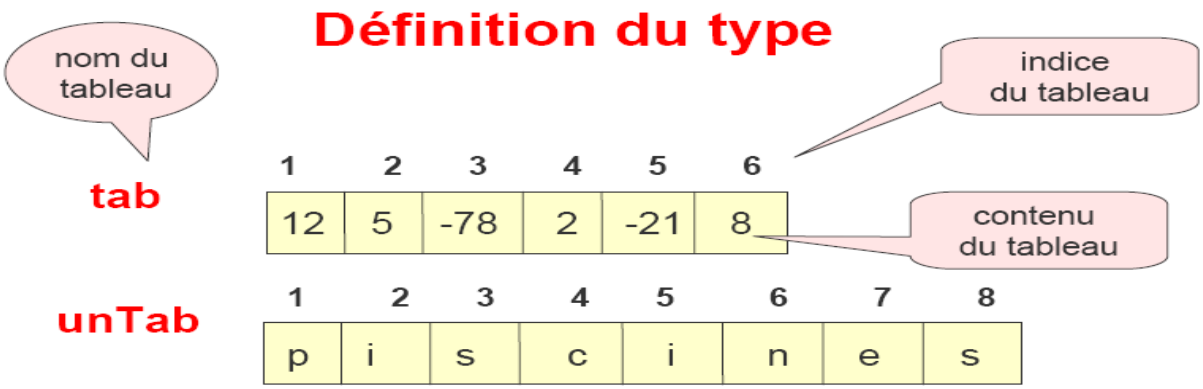


tableau à **deux** dimensions

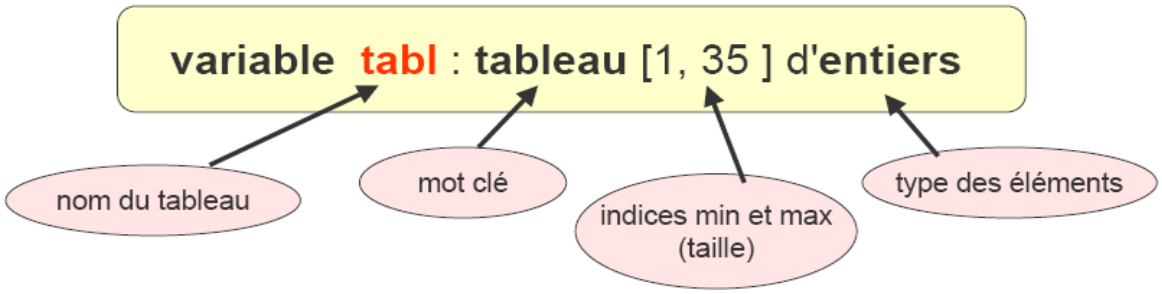




Remarques :
- Indices : en général, démarrage à 1, **mais en C++, démarrage à 0**
- Nombre d'octets occupés : dépend du type des valeurs enregistrées

2. *Déclaration d'un tableau*

Exemple : déclaration d'un tableau pouvant contenir jusqu'à 35 entiers



Autre exemple : déclaration d'un tableau qui contiendra les fréquences des températures comprises entre –40°C et 50°C

variable températures : tableau [-40, 50] de réels

- ✓ Un *tableau* possède un nom (ici note) et un nombre d'éléments (de cases) qui représente sa taille (ici 12).
- ✓ Tous les éléments d'un tableau ont le même type (c'est normal car ils représentent des valeurs logiquement équivalentes).
- ✓ Pour designer un élément, on indique le nom du tableau suivi son indice (son numéro) entre crochets:

3. *Les variables d'un tableau*

La notion de ≪case contenant une valeur ≫doit faire penser a celle de variable. Et, en effet, les cases du tableau, encore appelées éléments du tableau, sont des variables, qualifiées d'indices.

- ✓ *Différence entre variables classiques et variables indices:*

_ Les *variables classiques* sont déclarées individuellement et ont un nom distinct ;
_ Les *variables indices* (constituant le tableau) sont implicitement déclarées lors de la déclaration du tableau. Pour bien montrer que ces l'endroit qu'il faudra remplir.

❖ *Utilisation d'un tableau : par les indices*

- Accès en lecture :
  - **afficher**(tabl[4]) *{le contenu du tableau à l'indice 4 est affiché à l' écran}*
- Accès en écriture :
  - tabl[3] ←18 *{la valeur 18 est placée dans le tableau à l'indice 3}*
  - **saisir**(tabl[5]) *{la valeur entrée par l'utilisateur est enregistrée dans le tableau à l'indice 5}*
  - attention!

~~tabl ← 18~~ ~~nom[2] ← 3~~

Remarque:
Dans un programme, chaque élément d'un tableau est repéré par un indice. Dans la vie courante, nous utilisons souvent d'autres façons de repérer une valeur. Par exemple, au lieu de parler de note [1], note [2], note [3], nous préférons parler des notes des étudiants. Le tableau ne permet pas de repérer ses valeurs autrement que par un numéro d'indice. Donc si cet indice n'a pas de signification, un tableau ne permet pas de savoir a quoi correspondent les différentes valeurs.

L'indice d'un élément peut être:

- *directement une valeur ex: Note [10]*
- *une variable ex: note [i]*
- *une expression entière ex: note [k+1] avec k de type entier*

Quelque soit sa forme, la valeur de l'indice doit être :

- *entière*
- *comprise entre les valeurs minimales et maximales déterminées à la déclaration du tableau.*

4. *Méthode d'écriture et de lecture d'un tableau*

✓ Méthode d'écriture :

Pour saisir (remplir) un tableau, il il existe deux façon de le faire :

SOIT JE CONNAIS LE NOMBRE DE CASE (DIMENSION DU TABLEAU) DANS CE CAS:

```
Ecrire (veuillez entrer les 12 notes dans le tableau)
Pour i allant de 1 à 12 faires
Lire (Note[i])
Fin pour
```

II. *SOIT JE NE CONNAIS PAS LE NOMBRE DE CASE* (DIMENSION DU TABLEAU) ET DANS CE CAS :

```
//Demande de saisir la dimension du tableau
Ecrire (veuillez entrer la dimension du tableau)
```

```
Lire (N)
//Ensuite on demande de saisir les valeurs du tableau
Ecrire (« veuillez entrer les 12 notes dans le tableau »)
Pour i allant de 1 à N faire
Lire (Note[i])
Fin pour
```

Fin du chapitre

Bibliographie

Web :

http://pise.info/algo/boucles.htm

https://algo.developpez.com/tutoriels/initiation/

Livre :