Université Mohamed Seddik BENYAHIA – JIJEL –

Faculté des Sciences Exactes et Informatique

Département d'Informatique

ALGORITHMIQUE ET STRUCTURES DE DONNEES 1

Cours

Auteur : Dr. KARA Messaoud

Version du 16/10/2021

Table des matières

Avant-propos

Ce cours "Algorithmique et Structures de Données 1" est destiné aux étudiants du département MI (Socle Commun en Mathématiques et Informatique) de l'université Mohamed Seddik Benyahia de Jijel.

Les objectifs de ce cours :

1- Acquérir les notions de base de l'algorithmique.
2- Acquérir l'aptitude à écrire des algorithmes (et des programmes) simples.
3- Evoluer étape par étape à partir d'algorithme simple vers des algorithmes plus structurés et avec plus de possibilités.
4- Après la maitrise des types simples, il passe aux types structurés et personnalisés.

Le cours sur le langage C est aussi divisé en plusieurs parties, mais il intégré aux séries de TPs. Chaque série, contient la partie du cours nécessaire aux étudiants pour pouvoir travailler leurs séries sereinement.

Chapitre 1 : Introduction à l'informatique et a l'algorithmique

Dans ce premier chapitre introductif nous nous intéressons à quelques définitions comme l'informatique, à l'ordinateur et à l'algorithmique.

Définitions

Informatique

C'est la science du traitement automatique de l'information.

Information + Automatique ➔ Informatique.

On définit aussi l'informatique comme étant l'ensemble des applications mettant en œuvre deux éléments distincts mais indissociables :

- **Le hardware (le matériel)** : qui est l'ensemble du matériel constitutif de l'ordinateur et de ses périphériques.
- **Le software (le logiciel)** : qui comprend l'ensemble des programmes qui s'exécutent sur le matériel. Ces programmes peuvent être répartis en deux sous-ensembles :
  - ✓ Les programmes de base : ils constituent le système d'exploitation (DOS, Windows, Linux, Unix, Mac OS, …).
  - ✓ Les programmes d'application : Selon les besoins des utilisateurs, sont installés des programmes supplémentaires. Exemples : Word (pour rédiger un document, une lettre, un rapport, …), Excel (Pour faire des calculs : facture, moyennes des notes, …), CodeBlocks (pour programmer en langage C), AutoCAD (pour créer des plans d'architecte, …), …

Système informatique

On appelle système informatique l'ensemble des moyens logiciels (software) et matériels (hardware) nécessaires pour satisfaire les besoins informatiques des utilisateurs.

Ordinateur

L'ordinateur est la machine qui se charge du traitement automatique des informations. Il peut traiter divers types d'informations (textes, dessins, images, sons) mais de manière interne toutes ces informations sont converties sous forme numérique binaire (0 et 1).

Un ordinateur est généralement composé d'une unité centrale (comprenant un processeur et une mémoire) et des périphériques d'entrée (Clavier, souris, microphone, webcam, scanner, …) et des périphériques de sortie (Ecran, imprimante, haut-parleurs, …).

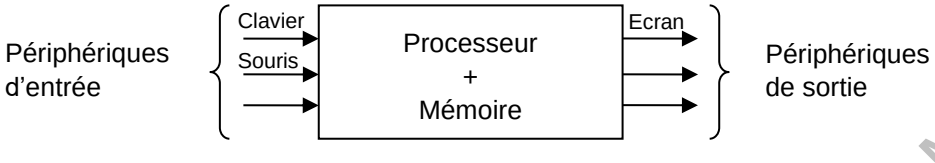
Le cœur d'un ordinateur est constitué de :

- ❖ **Processeur :** chargé de l'exécution des programmes. C'est le cerveau de l'ordinateur. Il contient les différents composants responsables de l'interprétation des instructions et des calculs.
- ❖ **Mémoires :** Chargées de stocker les programmes et les données et organisées en cases dans lesquelles on peut déposer des informations. Les informations sont très simples : 0 et

1. Chaque case élémentaire capable de mémoriser 0 ou 1 est appelée un **bit (BInary digiT)**. La capacité d'une mémoire peut se mesurer en nombre d'octets disponibles tels que : 1 octet (o) = 8 bits ; 1 kilo-octet (ko) = 2^{10} = 1024 octets ; 1 Méga-octet (Mo) = 2^{10} = 1024 ko ; 1 Giga-octet (Go) = 1024 Mo ; 1 Téra-octet (To) = 1024 Go.

- Les périphériques d'entrée/sortie : ce sont les composants qui permettent à l'ordinateur de communiquer avec l'extérieur (avec un utilisateur et/ou un autre ordinateur).

On peut schématiser un ordinateur comme suit :



Qu'est-ce que l'algorithmique ?

L'algorithmique est la logique d'écrire des algorithmes.

Pour pouvoir écrire des algorithmes, il faut connaître la résolution manuelle du problème, connaître les capacités de l'ordinateur en termes d'actions élémentaires qu'il peut assurer et la logique d'exécution des instructions.

Origine du nom algorithme

Le mot algorithme vient du nom du mathématicien musulman perse du 9ième siècle Abu Abdullah Muhammad ibn Musa al-Khwarizmi originaire de l'ancienne ville KHAWARISM, aujourd'hui KHIVA située en ex-URSS.

Le mot algorithme se référait à l'origine uniquement aux règles d'arithmétique utilisant les chiffres indo-arabes numérales mais cela a évolué par la traduction en latin européen du nom Al-Khawarizmi en algorithme au 18ième siècle. L'utilisation du mot a évolué pour inclure toutes les procédures définies pour résoudre un problème ou accomplir une tâche.

Définitions d'un algorithme

Définition 1

Un algorithme est une suite d'actions (ordonnées) que devra effectuer un automate pour arriver en un temps fini à un résultat déterminé à partir d'une situation donnée.

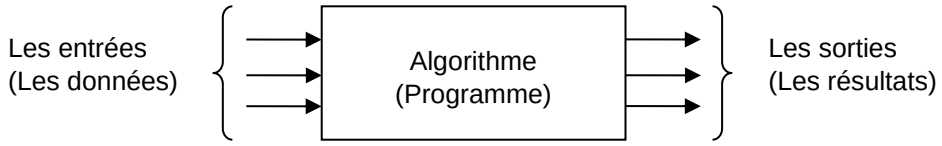
La suite d'actions est composée d'opérations élémentaires appelées instructions.

Définition 2

Un algorithme peut être considéré comme une machine fonctionnant en trois étapes :

1- Introduire les données nécessaires : **Les entrées (ou les données)**.
2- Exécuter séquentiellement des instructions sur ces données : **Les traitements (Les actions)**.
3- Afficher les résultats obtenus : **Les sorties (Les résultats)**.

Les entrées et les sorties forment la partie déclaration d'un algorithme ; La partie actions contient la liste des instructions.

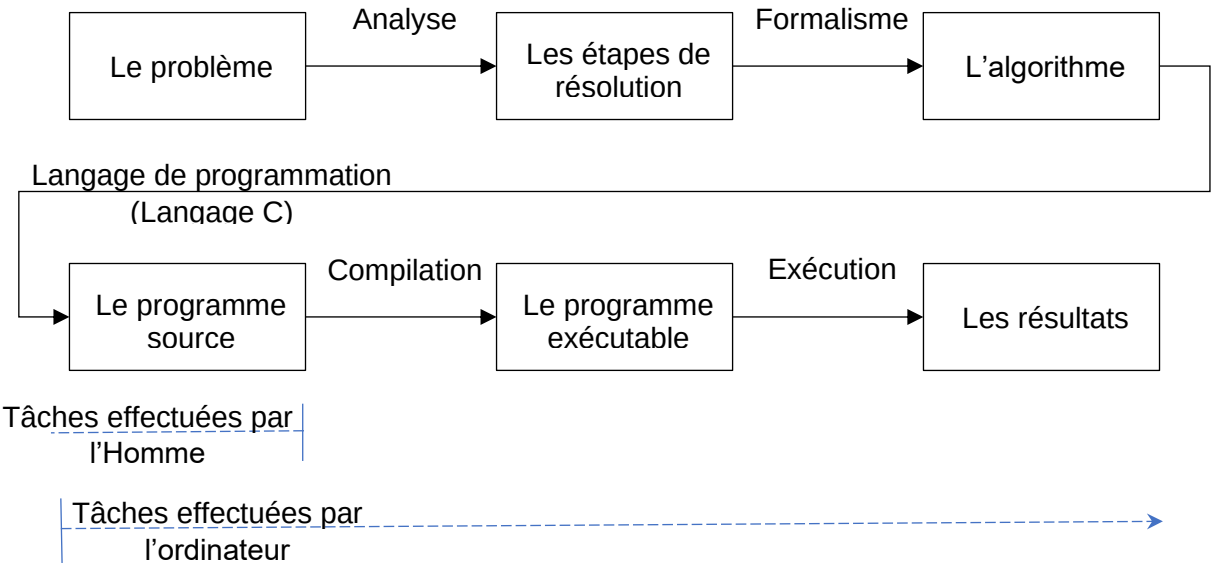


Exemples d'algorithmes (de la vie courante)

1. Une recette de cuisine : contient deux parties : Les ingrédients et le mode de préparation. La recette précise clairement, les ingrédients nécessaires à la préparation d'un repas ou d'un gâteau ainsi que les opérations de préparation par ordre chronologique.
2. Une notice de montage d'un meuble livré en kit (Bureau, armoire, …). Cette notice montre comment monter le meuble livré en montrant les étapes, en général, par des schémas évolutifs.
3. Manuel d'utilisation d'un appareil électronique, électroménager, …
4. Résolution d'une équation du second degré ($\mathbf{Ax^2 + Bx + C} = 0$).

Construction d'un programme

Un programme peut être construit en plusieurs étapes :



- ✓ La première étape : consiste en l'analyse du problème posé. Le résultat de cette étape est la décomposition du problème en ses composants élémentaires qu'on appelle aussi des opérations.
- ✓ La deuxième étape : est l'établissement d'un algorithme qui n'est autre qu'une présentation des étapes de résolution du problème analysé en respectant un formalisme (un ensemble de règles d'écriture) bien déterminé.

- ✓ La troisième étape : est la traduction de l'algorithme en programme en utilisant un langage de programmation choisi (Le langage C, par exemple). Une fois le programme écrit, il va falloir le vérifier et le corriger en lançant la compilation.
  Le compilateur est un logiciel qui détecte les erreurs de syntaxe du programme, mais ne détecte pas les erreurs de logique.
- ✓ La dernière étape : consiste en l'exécution du programme compilé. Cette étape aboutit à l'exploitation du programme et la vérification des résultats.

Exemple : Ecrire un algorithme qui fait l'addition de deux nombres réels A et B.

Analyse : S = A + B. Comme A et B sont des réels, alors S est réel.

Opérations :

① Donner la valeur de A | Entrée
② Donner la valeur de B | Entrée
③ S = A + B | Calcul
④ Afficher S | Sortie

Algorithme & programme

```
Algorithme    Addition
Var    A, B, S : Réel
Début
    Lire( A )
    Lire( B )
    S ← A + B
    Ecrire( S )
Fin
```

```
// Programme qui fait l'addition
#include <stdio.h>

int main( )
{
        float A, B, S ;

        scanf("%f", &A) ;
        scanf("%f", &B) ;
        S = A + B ;
        printf("La somme = %f\n", S) ;

        return 0 ;
}
```

Processus de résolution d'un problème - Synthèse

En d'autres termes, la résolution d'un problème passe par les étapes suivantes :

1- Comprendre l'énoncé du problème.
2- Décomposer le problème en sous problèmes plus simples à résoudre.
3- Associer à chaque sous problème une spécification :
   - ✓ Les données nécessaires,
   - ✓ Les données résultantes,
   - ✓ La démarche à suivre pour arriver au résultat en partant d'un ensemble de données.
4- Élaboration d'un algorithme.
5- Traduction de l'algorithme en programme.

6- Compilation du programme pour détecter s'il y a des erreurs.
7- Exécution du programme compilé et exploitation des résultats.

Le langage algorithmique

Algorithme - Définition 3

L'algorithme est une description des étapes de résolution d'un problème particulier. Il permet d'expliciter clairement les idées de la solution indépendamment d'un langage de programmation.
L'utilisateur de l'algorithme n'aura qu'à suivre toutes les instructions dans l'ordre pour arriver au résultat recherché.

Le langage algorithmique

Le langage algorithmique est un pseudo-langage qui tient en compte les caractéristiques de la machine, tout en étant plus souple qu'un langage de programmation. C'est, donc, un compromis entre le langage naturel et un langage de programmation caractérisé par une liste de mots clés.

Le mot clé

Un mot clé est un identificateur (mot) qui a une signification particulière (Début, Fin, Si, Pour, Répéter, Tantque, …) au langage.

Le programme

Le programme est une traduction de l'algorithme dans un langage de programmation (en langage C, par exemple).

Chapitre 2 : Algorithme séquentiel simple

Dans ce chapitre nous nous intéressons à la présentation de la structure générale d'un algorithme et de sa partie réservée aux déclarations.

Structure générale d'un algorithme

Un algorithme se présente comme suit :

```
Algorithme   nom de l'algorithme          // L'en-tête
Const        Liste_des_constantes
Type         Liste_des_types
Var          Liste des variables          // Les déclarations
             Liste des procédures et fonctions
Début
    Instruction 1
    Instruction 2
    …                                     // Le corps de l'algorithme
    Instruction N                         // Les traitements
Fin
```

- ✓ **L'en-tête** permet d'identifier l'algorithme. C'est-à-dire, elle sert à lui donner un nom.
- ✓ **Les déclarations :** C'est une liste de tous les objets (constantes, variables, ...) utilisés et manipulés dans le corps de l'algorithme.
- ✓ **Le corps :** Dans cette partie sont placées les opérations et les instructions à exécuter.

Exemple : L'algorithme qui permet de permuter (échanger) les valeurs de deux nombres entiers X et Y. C'est-à-dire la valeur de X va à Y et la valeur de Y va à X.

Exemple : Si, au départ, X= 10 et Y=17, alors à la fin de l'algorithme on aura X=17 et Y=10.

```
Algorithme Permutation
Var  X, Y, Z : Entier
Début
    Lire( X, Y )
    Z ←X
    X← Y
    Y ← Z
    Ecrire ( X, Y )
Fin
```

La traduction de l'algorithme Permutation en programme C est donnée ci-dessous :

```
// Programme qui permute les valeurs de deux entiers X et Y.
#include <stdio.h>
int main( )
{
        int X, Y, Z ;
        scanf("%d", &X) ;
        scanf("%d", &Y) ;
        Z = X ;
        X = Y ;
        Y = Z ;
        printf("X = %d et Y = %d\n", X, Y) ;
        return 0 ;
    }
```

Variables et constantes

Définition d'une variable

Une variable est un espace mémoire identifié par un nom, destiner à stocker une valeur, qui peut être modifiée durant les traitements. Les variables d'un algorithme contiennent les informations nécessaires à son déroulement.

Définition d'une constante

une constante ne prend qu'une unique valeur au cours de l'exécution de l'algorithme.

Définition d'un identificateur

Un identificateur (identifiant) est un nom qui respecte une syntaxe particulière :

- ✓ Il est constitué d'une suite de lettres de l'alphabet (latin) et de chiffres.
- ✓ Il commence, obligatoirement, par une lettre.
- ✓ Le caractère souligné "_" (Le tiret bas, underscore) peut jouer le même rôle qu'une lettre de l'alphabet. Sur le clavier, le caractère "_" se trouve sur la même touche qui permet d'écrire le chiffre 8.

Exemples : x, y, PI, rayon_du_cercle, _x1, _y2, …

Remarques

- ✓ Pour faciliter la lisibilité d'un algorithme, il est préférable d'utiliser des noms significatifs.
- ✓ L'identificateur doit être différent de tous les mots clés (Début, Fin, Si, Alors, Sinon, …).
- ✓ Les caractères de l'alphabet grec sont interdits. Si on a besoin d'utiliser α, β, δ, ou Δ, on doit les écrire en alphabet latin comme alpha, béta, gamma, delta et ainsi de suite.
- ✓ Il est interdit d'utiliser des caractères accentués (é, è, î, ç, ', ", …) dans un identifiant.

✓ Il est interdit d'utiliser des indices ou des exposants.

Exemples : ne pas utilser x_1, x_2 mais plutôt utiliser x1 et x2.

ne pas utilser x^1, x^2 mais plutôt utiliser x1 et x2 ou Xp1 et Xp2.

Le type : Le type correspond au genre d'information utilisé. Les types standard (prédéfinis) en langage algorithmiques sont : Entier, Réel, Booléen, Caractère, Chaîne (de caractères). Chaque type est muni (dispose) d'un ensemble d'opérations.

Syntaxe de déclaration des constantes

```
CONST   nomConstante1 = Valeur1
        nomConstante2 = Valeur2
        …
        nomConstanteN = ValeurN
```

Exemples :

```
Const   Nombre = 15   // Constante de type ENTIER
        Pi = 3.14     // Constante de type REEL
        Rep = Vrai    // Constante de type BOOLEEN
        Reponse = 'N' // Constante de type CARACTERE
        Univ = "Université de Jijel"   // Constante de type CHAINE
```

Remarque : Pour déclarer une constante on définit son nom et sa valeur.

Syntaxe de déclaration des variables

```
VAR   nomVariable1, …, nomVariableN1 : Type1
      nomVariable2, …, nomVariableN2 : Type2
      nomVariableM, …, nomVariableNM : TypeM
```

Exemples :

```
Var   A, B, C, D : Entier      // Déclaration de 4 variables de type Entier.
      X, Y, Z : Réel           // Déclaration de 3 variables de type Réel.
      Stop : Booléen           // Déclaration d'une variable de type Booléen.
      Reponse : Caractère      // Déclaration d'une variable de type Caractère.
      Nom, prenom : Chaine     // Déclaration de 2 variables de type Chaine.
```

Remarque : Pour déclarer une variable on définit son nom et son type.

Les types

Le type correspond au genre d'information utilisé. Les types standard (prédéfinis) en langage algorithmiques sont : Entier, Réel, Booléen, Caractère, Chaîne (suite de caractères). Chaque type est muni (dispose) d'un ensemble d'opérations.

1- Le type Entier

Le type Entier est utilisé pour manipuler des nombres entiers : -17, -8, 0, 13, 22, …

Le type entier est muni des opérateurs suivants :

- ✓ Les opérateurs arithmétiques: + (addition), - (soustraction), * (multiplication).
- ✓ La division entière, notée **DIV** , tel que n div p donne **la partie entière du quotient** de la division entière de n par p. Exemple : 23 Div 5 = 4.
- ✓ Le modulo, noté **MOD**, tel que n mod p donne **le reste** de la division entière de n par p. Exemple : 23 Mod 5 = 3.
- ✓ Les opérateurs de comparaison classiques : <, ≤, =, ≠, ≥, >
- ✓ D'autres opérations sont définies par des fonctions dont on peut citer :
  - **abs**(n) : Cette fonction fournit la valeur absolue de l'entier n.

2- Le type Réel

Le type Réel est utilisé pour manipuler des nombres réels : -3.7, -1.5, 0, 3.0, 18.25, …

Le type Réel est muni des opérateurs suivants :

- ✓ Les opérations arithmétiques classiques : + (addition), - (soustraction), * (multiplication), / (division).
- ✓ Les opérateurs de comparaison classiques : <, ≤, =, ≠, ≥, >
- ✓ D'autres opérations sont définies par des fonctions dont on peut citer :
  - **trunc(x)** : Cette fonction donne la partie entière du nombre réel x.
  - **round(x)** : Cette fonction donne l'entier le plus proche du nombre réel x.
  - **abs(x)** : Cette fonction donne la valeur absolue du nombre réel x.
  - **sqrt(x)** : Cette fonction donne la racine carrée du nombre réel x.
  - Les fonctions trigonométriques : **sin(x)**, **cos(x)**, **arctg(x)**, …

Remarques

- ✓ On peut affecter à une variable de type réel une autre variable de type entier. C'est la machine qui prend en charge la conversion de la valeur entière en valeur réelle.
- ✓ Les variables de type entier et les variables de type réel peuvent être combinées dans des opérations définies sur les réels. Les entiers sont alors automatiquement convertis en réels.

3- Le type Booléen

Il s'agit du domaine dont les seules valeurs sont ***VRAI*** ou ***FAUX***. Les opérateurs booléen (logiques) définis et les plus utilisés sont : le **NON**, le **ET** et le **OU**. Ils sont définis par les tables de vérité ci-dessous.

Soient A et B deux variables de type Booléen.

A	NON A
Faux	Vrai
Vrai	Faux

A	B	A ET B
Faux	Faux	Faux
Faux	Vrai	Faux
Vrai	Faux	Faux
Vrai	Vrai	Vrai

A	B	A OU B
Faux	Faux	Faux
Faux	Vrai	Vrai
Vrai	Faux	Vrai
Vrai	Vrai	Vrai

4- Le type Caractère

Le type caractère est un ensemble fini et totalement ordonné de caractères (symboles). Il comporte :

- ✓ Les lettres de l'alphabet latin : a .. z, A .. Z.
- ✓ Les chiffres : 0 .. 9.
- ✓ Les symboles utilisés en tant qu'opérateurs : + - * / < = > …
- ✓ Les caractères de ponctuation : . , ; ! ? …
- ✓ Les caractères spéciaux : @ % & # …
- ✓ Et d'autres.

La figure ci-dessous montre la table ASCII comportant les 256 caractères. Ils sont numérotés de 0 à 255. Chaque caractère est stocké en mémoire d'un ordinateur sur un octet.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
32	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
96	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
128	Ç	ü	é	â	ä	à	å	ç	ê	ë	è	ï	î	ì	Ä	Å	É	æ	Æ	ô	ö	ò	û	ù	ÿ	Ö	Ü	ø	£	Ø	×	ƒ
160	á	í	ó	ú	ñ	Ñ	ª	º	¿	®	¬	½	¼	¡	«	»	¦	¦	¦	¦	¦	Á	Â	À	©	¦	¦	+	+	¢	¥	+
192	+	-	-	+	-	+	ã	Ã	+	+	-	-	¦	-	+	¤	ð	Ð	Ê	Ë	È	i	Í	Î	Ï	+	+	¦	_	¦	Ì	¯
224	Ó	ß	Ô	Ò	õ	Õ	µ	þ	Þ	Ú	Û	Ù	ý	Ý	¯	´	-	±	=	¾	¶	§	÷	¸	°	¨	·	¹	³	²	¦	

Pour connaître le rang d'un caractère ('A' par exemple) dans l'ensemble des caractères représenté par cette table, il faut faire la somme des deux nombres se trouvant sur la même ligne (dans la première colonne à gauche) et la même colonne (dans la première ligne en haut). Exemple : Le rang (l'ordre) du caractère 'A' est égal à 64 + 1 = 65.

- ✓ Une constante de type caractère est représentée par un et un seul caractère encadré par deux apostrophes simples. Exemples : ',a', 'b', 'C', '!', '#', …

Les opérations définies par des fonctions sur le type Caractère sont :

- ✓ **ORD**(c) : Cette fonction renvoie un entier positif correspond au rang du caractère c, dans l'ensemble des caractères. C'est le numéro (le rang) du caractère c. Exemples : Ord('!') = 33, Ord ('A') = 65, Ord ('a') =97.
- ✓ **CHR**(i) : C'est la fonction inverse de Ord. Pour un entier positif i, elle renvoie le caractère de rang i. Exemples : Chr(33) = '!', Chr(65) = 'A', Chr(97) = 'a'.
- ✓ **SUCC**(c) : Cette fonction fournit le caractère qui suit immédiatement le caractère c dans l'ensemble des caractères. Exemples : Succ('a') = 'b', Succ('3') = '4', Succ('%') = '&'.
- ✓ **PRED**(c) : Cette fonction fournit le caractère qui précède immédiatement le caractère c dans l'ensemble des caractères. C'est la fonction inverse de Succ. Exemples : Pred('b') = 'a', Pred ('4') = '3', Pred ('&') = '%'.

Tous les caractères respectent les principes suivants :

- ✓ Les caractères alphabétiques (majuscules et minuscules) se suivent et sont ordonnés dans l'ordre alphabétique ; C'est-à-dire 'A' < 'B' < … < 'Z' < … < 'a' < 'b' < … < 'z'.
- ✓ Les caractères numériques (les chiffres) se suivent et sont ordonnés dans l'ordre croissant ; C'est-à-dire '0' < '1' < … < '9'.
- ✓ La relation d'ordre est définie sur le type caractère tel que : Si x et y sont deux variables de type caractère alors : x < y si ord (x) < ord (y). A partir de la table ASCII, on peut voir facilement que '0' < … < '9' < 'A' < … < 'Z' < 'a' < … < 'z'.

5- Le type Chaîne

Une chaîne est une suite de caractères. Une chaîne est encadrée par deux apostrophes doubles. Exemples : "Université de Jijel", "Département MI", "Algorithmique", …

- ✓ On appelle **longueur** d'une chaîne le nombre de caractères de cette chaîne. "Algorithmique" est une chaîne de longueur 13.
- ✓ "" : représente la chaine vide de longueur 0. Elle ne contient aucun caractère.

Les fonctions prédéfinies sur les chaînes sont :

- ✓ **Length**(str) : elle fournit la longueur de la chaîne str.
- ✓ **Concat**(str1, str2) : elle fournit la chaîne obtenue par concaténation des deux chaînes str1 et str2. Exemple : Concat("Module", " Algorithmique") = "Module Algorithmique"

Les opérateurs de relation (<, ≤, =, ≠, ≥, >) sont définis sur le type chaîne tels que :

- ✓ Deux chaînes sont égales si elles sont identiques.
- ✓ La chaîne vide est inférieure à toute autre chaîne.
- ✓ Les chaînes sont ordonnées selon l'ordre lexicographique.
  Exemples : "Cours" < "cours", "cour" < "cours", "courage" < "cours".

Remarque sur les types : Chaque type a une taille et une représentation particulière en mémoire (de l'ordinateur). Il ne faut pas confondre les différentes formes de constantes.
Exemples : 3 (type entier), 3.0 (type réel), '3' (type caractère) et "3" (type chaine).

Chapitre 3 : Expressions et instructions élémentaires

Les expressions

- Une expression est une suite d'opérations appliquées sur un ensemble de facteurs (arguments ou paramètres). Chaque expression a une valeur et un type.

Exemple : **a + 5** est une expression qui représente une addition entre les deux facteurs **a** et **5**.

- Les opérateurs sont :
  - ✓ Les opérateurs algébriques : +, -, *, /, DIV, MOD.
  - ✓ Les opérateurs logiques : ET, OU, NON.
  - ✓ Les opérateurs relationnels : <, ≤, =, ≠, ≥, >.
- Un opérateur qui s'applique sur deux facteurs est dit binaire (+, -, *, ET, OU …).
- Un opérateur qui s'applique sur un seul facteur est dit unaire (+, -, NON).

Les règles d'évaluation d'une expression

- Le calcul de la valeur d'une expression comportant plus d'un opérateur dépend du sens que l'on donne à cette expression.

Exemple : X + Y * Z est une expression ambigüe.

- Les parenthèses permettent de lever l'ambiguïté : (X + Y) * Z ou X + (Y * Z).
- En absence de parenthèses, pour éviter toute ambiguïté, des règles d'évaluation ont été établies. Un ordre de priorité entre les opérateurs est introduit de façon décroissante comme suit :
  - ✓ Opérateurs unaires : + (exemple : +X) , - (exemple : -X), NON (NON X).
  - ✓ Opérateurs multiplicatifs : *, /, Div, Mod, ET.
  - ✓ Opérateurs additifs (binaires) : +, -, OU.
  - ✓ Opérateurs relationnels : : <, ≤, =, ≠, ≥, >.
  - ✓ Si une expression comporte plusieurs opérateurs de même priorité, ces opérateurs sont associatifs à gauche.
    Et donc le sens attribué à X + Y * Z est bien X + (Y * Z).
- Dans le cas où on souhaite modifier la sémantique définie par ces règles, il faut introduire des parenthèses.

Remarque : Pour éviter toute ambiguïté, il faut toujours introduire les parenthèses.

Les instructions élémentaires

En algorithmique, Il existe trois instructions élémentaires : Lire, Ecrire et l'affectation (←).

1- L'instruction de lecture (LIRE)

✓ L'instruction **LIRE** permet de donner une valeur à une variable à partir du clavier.

Syntaxe : Lire(nomVariable)

Exemples : Lire(A)
Lire(B)
Lire(C)

- ✓ On peut regrouper plusieurs instructions successives de lecture en une seule instruction. Pour l'exemple précédent, on peut remplacer les trois instructions de lecture par : Lire(A, B, C).
- ✓ Sur un ordinateur, lorsque le processeur reçoit l'ordre Lire(nomvariable) il arrête l'exécution du programme et se met en attente d'une valeur. L'utilisateur doit alors saisir une valeur à partir du clavier. Dès validation de saisie (par l'appui sur la touche Entrée ↵), l'exécution du programme se poursuit. La valeur transmise par l'utilisateur est affectée à la variable et écrase la valeur précédente de celle-ci.
- ✓ L'instruction Lire(nomVariable) provoque une erreur si la valeur saisie n'appartient pas au type de la variable, à moins que ce soit une valeur entière et que la variable est de type réel. Auquel cas la valeur entière est convertie en une valeur réelle.

2- L'instruction d'écriture (ECRIRE)

- ✓ L'instruction **ECRIRE** permet l'affichage sur écran. Il y a deux types d'affichage :
  - ➢ Soit on affiche la valeur d'une variable ou d'une expression :
    Syntaxe : Ecrire (nomVariable)
    Exemple : Ecrire(X) ou Ecrire(2 * X + Y). Si X = 10 et Y= 5 alors, la première instruction affiche 10 et la deuxième affiche 25
  - ➢ Soit on affiche un texte (un message) : **Syntaxe** : Ecrire ("un message")
    Exemple : Ecrire("Le résultat est = ")
- ✓ On peut regrouper plusieurs instructions successives d'écriture en une seule instruction.

Par exemple, la suite d'instructions :

```
Ecrire ( "Le résultat est =" )
Ecrire ( X )
```

Peut être remplacés par : Ecrire ("Le résultat est =", X). Cette instruction permet d'afficher la chaîne "Le résultat est = 10".

Remarque : La communication par messages est très utile en programmation. Elle offre la possibilité : ①- D'orienter l'utilisateur en lui indiquant ce que la machine attend de lui suite à un ordre de lecture. ②- D'expliquer les résultats d'un traitement.

3- L'affectation (←)

L'affectation est une instruction qui stocke la valeur d'une expression dans une variable.

Syntaxe : nomVariable ← expression. Cette instruction se lit : nomVariable reçoit expression.

Exemples : X ← 7 // Affecter une valeur constante à la variable X (X reçoit 7).
Y ← X // Copier la valeur de la variable X dans la variable Y (Y reçoit X).

Z ← 2 * X + T / Y // une expression formée de plusieurs opérations (Z reçoit 2 * X + T / Y).

Remarques

①- Si la variable contenait déjà une valeur, elle serait remplacée par la valeur de l'expression. L'ancienne valeur de la variable est perdue et il n'y a aucun moyen de la récupérer !

②- Une opération de contrôle de type est effectuée avant l'affectation. C'est une erreur si la valeur de l'expression n'appartient pas au type de la variable, à moins que ce soit une valeur entière et que la variable est de type réel. Dans ce cas la valeur entière est convertie en réel.

Exemple : Si, au départ, X=10 et Y=17, alors à la fin de l'algorithme X=17 et Y=10.

```
Algorithme Permutation
Var X, Y, Z : Entier
Début
    Lire( X, Y )
    Z ←X
    X← Y
    Y ← Z
    Ecrire ( X, Y )
Fin
```

Remarque

D'une manière générale, pour construire un algorithme on doit suivre trois étapes :

①- Introduire les données (Des instructions de **lectures**).

②- Résoudre le problème : C'est-à dire manipuler ces données pour obtenir la solution au problème posé (Faire des calculs avec des instructions d'**affectations**).

③- Affichage des résultats (Des instructions d'**écritures**).

Chapitre 4 : Structures de contrôle

Les **structures de contrôle** encore appelées **instructions structurées** permettent d'exprimer comment s'enchaînent les instructions d'un algorithme. Il existe trois structures : La séquence, la sélection et l'itération. Elles permettent d'exprimer tous les enchaînements possibles dans un algorithme.

1- La séquence

- ✓ La séquence exprime un enchaînement séquentiel des instructions. Elle se présente sous forme d'une suite ordonnée d'instructions regroupée en un bloc.

Syntaxe :
```
Instruction 1
Instruction 2
…
Instruction N
```

- ✓ Tout algorithme est défini par une séquence d'instructions délimitée par les mots clés **Début** et **Fin**.
- ✓ Les instructions d'une séquence sont exécutées séquentiellement (l'une après l'autre) dans l'ordre ou elles se présentent. Aucune instruction n'est ignorée.

2- La sélection

La sélection exprime un enchaînement conditionnel sélectif. Elle offre la possibilité de sélectionner pendant l'exécution la prochaine séquence (Bloc d'instructions) à exécuter. La sélection se fait sur la base d'une condition.

La sélection est utilisée pour traiter tous les cas rencontrés dans un algorithme.

2-1. L'alternative (Si)

L'alternative est une situation dans laquelle on est amené à choisir entre deux solutions possibles. Il existe deux types de la structure alternative :

- ✓ L'alternative complète.
- ✓ L'alternative réduite.

L'alternative complète (Si … Sinon … FSi)

L'alternative complète permet de sélectionner sur la base d'une condition une séquence d'instructions à exécuter parmi deux séquences.

Syntaxe : **Si** (Condition) **Alors**

    Séquence 1

Sinon

    Séquence 2

FSi

❖ La condition est formulée par une expression booléenne.

Sémantique : L'exécution de cette instruction structurée commence par l'évaluation (calcul) de la condition :

- ✓ Si la condition est vérifiée (= Vrai), c'est la Séquence 1 qui est exécutée. La séquence 2 est ignorée (n'est pas exécutée).
- ✓ Si la condition est fausse (= Faux), c'est la Séquence 2 qui est exécutée. La séquence 1 est ignorée.

Exemple 1 : **Si** (Moyenne ≥ 10) **Alors**

    Ecrire("Etudiant ADMIS") // Passe à l'année suivante.

Sinon

    Ecrire("Etudiant Ajourné") // Redouble.

FSi

Exemple 2 : Vérifier si un entier N est positif ou négatif.

Si (N < 0) **Alors**

    Ecrire(N, " est négatif")

Sinon

    Ecrire(N, " est positi")

FSi

Exemple 3 : Résolution d'une équation du second degré.

Si (delta < 0) **Alors**

    Ecrire("Pas de solution")

Sinon

    Ecrire("Deux solutions")

FSi

Remarque : Pour certains problèmes, on doit sélectionner une séquence à exécuter parmi plus de deux séquences d'instructions. On est donc dans un cas de choix

multiples. La solution consiste à utiliser des alternatives imbriquées (l'une à l'intérieur d'une autre).

Exemple 3 : Résolution d'une équation du second degré en détails.

```
Si (delta < 0) Alors
|     Ecrire("Pas de solution")
Sinon
|     Si (delta = 0) Alors
|     |     Ecrire("Solution double")
|     Sinon
|     |     Ecrire("Deux solutions")
|     FSi
FSi
```

L'alternative réduite (Si … FSi)

L'alternative réduite exprime le cas où l'exécution d'une séquence d'instructions est soumise à une condition.

Syntaxe :

```
Si (Condition) Alors
|     Séquence
FSi
```

- La condition est formulée par une expression booléenne.

Sémantique: On commence par évaluer (calculer) la condition.

- ✓ Si la condition est vérifiée (= Vrai alors la Séquence 1 est exécutée. Sinon, elle est ignorée.

Exemple :

```
Si (Moyenne ≥ 17) Alors
|     Ecrire("Mérite un prix") // Les autres étudiants on n'en parle pas !
FSi
```

Remarque : L'alternative réduite est un cas particulier de l'alternative complète (cas ou la séquence 2 est vide.

2-2. Structure conditionnelle de choix multiples (Selon)

La structure conditionnelle de choix multiples est adaptée aux seuls cas ou les conditions testées sont toutes des égalités entre une même expression et des constantes d'un type discret (Entier ou Caractère).

Syntaxe : **Selon** (Expression) **vaut**

Liste1 : Séquence 1
Liste2 : Séquence 2
…
ListeN : Séquence N
Autre : Séquence (N+1)

FinSelon

- ✓ Liste1, Liste2, …, ListeN : Sont des listes de constantes. Chaque liste est formée d'une suite de constantes, du même type que l'expression, séparées par des virgules.

Sémantique : L'exécution de cette structure conditionnelle commence par l'évaluation de l'expression. Seulement la séquence qui contient dans sa liste la valeur de l'expression est exécutée. Et si la valeur de l'expression ne figure dans aucune liste, c'est la séquence (N+1) qui est exécutée (Elle commence par le mot clé Autre).

Exemple **:** **Selon** (mois) **vaut**

12, 1, 2 : Ecrire("C'est l'hiver")
3, 4, 5 : Ecrire("C'est le printemps")
6, 7, 8 : Ecrire("C'est l'été")
9, 10, 11 : Ecrire("C'est l'automne")
Autre : Ecrire("Ce n'est pas un mois de l'année")

FinSelon

3- L'itération (La boucle)

On appelle itération toute répétition de l'exécution d'un traitement décrit par une séquence d'instructions. A la notion d'itération est associée la notion de boucle. On appelle corps de la boucle la séquence d'instructions à répéter. Le nombre d'itérations doit être fini. Il est contrôlé soit par un compteur soit par une condition.

3-1. Itération contrôlée par un compteur (Pour)

La boucle **POUR** utilise un compteur pour contrôler le nombre d'itérations (de répétitions) de la séquence d'instructions.

Syntaxe : **POUR** i ← Valinit **À** ValFin **PAS** = n **FAIRE**

Séquence

FPour

Exemple : Pour afficher à l'écran les nombres de 1 à 10, on peut utiliser la boucle Pour suivante :

```
Pour i ← 1 à 10 Pas = 1 Faire
    Ecrire( i )
FPour
```

Sémantique :

- ✓ Le compteur d'itération i est initialisé par la valeur de l'expression Valinit.
- ✓ Si le pas n est positif, le compteur est incrémenté du pas n à chaque itération (i ← i + n). La répétition s'arrête dès que la valeur du compteur devienne supérieure à la valeur de l'expression ValFin.
- ✓ Si le pas n est négatif, le compteur est décrémenté du pas n à chaque itération. La répétition s'arrête dès que la valeur du compteur devienne inférieure à la valeur de l'expression ValFin.
- ✓ Le nombre d'itérations de la séquence d'instructions qui forme le corps de la boucle est déterminé par :
  - ➢ La valeur initiale du compteur d'itérations i déterminée par la valeur de l'expression entière Valinit.
  - ➢ La valeur finale du compteur d'itérations i déterminée par la valeur de l'expression entière ValFin.
  - ➢ La valeur et le signe de la constante entière n (Le pas d'incrémentation ou de décrémentation du compteur).

Remarques

- ➢ La variable i est un compteur donc, elle est de type entier.
- ➢ Si le pas n'est pas mentionné, il est implicitement égal à 1. C'est-à-dire, si le pas est égal à 1 on peut ne pas l'écrire.
- ➢ Dans la boucle POUR, l'initialisation du compteur i, l'incrémentation (ou la décrémentation) du compteur i et le test d'arrêt de l'itération sont à la charge du processeur.
- ➢ Dans le corps de la boucle, le compteur i ne peut recevoir aucune valeur avec une affectation ou lecture. C'est-à-dire, on ne peut pas écrire ni i ← expression, ni Lire(i) à l'intérieur du corps de la boucle.

Exemple 2 : En utilisant la boucle Pour, écrire un algorithme qui permet de calculer la somme S = 1+ 2 + 3 + … + N.

Solution :

```
Algorithme   Somme
Var     N, I, S : Entier
Début
        Lire( N )
        S ← 0
        Pour i ← 1 à N Faire
                S ← S + I
        FPour
        Ecrire("La somme = ", S )
Fin
```

Exercices : Modifier l'algorithme précédent pour :

①- Calculer la somme : $S = 1 + 2^2 + 3^2 + \ldots + N^2$.

②- Calculer X à la puissance N. C'est-à-dire : $X^N = X * X * \ldots * X$ (N fois)

③- Calculer la factorielle de N ! = N * (N – 1) * (N – 2) * … * 3 * 2 * 1

④- Calculer la somme : S = 1 ! + 2 ! + 3 ! + … + N !

(Voir le corrigé-type de la série de TD N°2)

3-2. Itération contrôlée par une condition

Il existe deux boucles contrôlées par une condition :

- ✓ La boucle **Répéter**.
- ✓ La boucle **TantQue** (**TQ**).

La boucle Répéter

La boucle Répéter permet de répéter l'exécution d'une séquence d'instructions jusqu'à ce qu'une condition soit vérifiée (= Vrai).

Syntaxe :

```
Répéter
        Séquence
Jusqu'à (Condition)
```

La condition est formulée par une expression booléenne.

Exemple : Pour filtrer les nombres entiers, afin de n'utiliser que les nombres naturels, on peut écrire :

```
Répéter
        Lire( N ) // N est déclaré Entier
Jusqu'à ( N ≥ 0 )
```

Si l'utilisateur saisis un nombre négatif, le système lui demande de relire N jusqu'à ce que la valeur lue soit positive ou nulle.

Sémantique : Le déroulement de cette boucle Répéter provoque successivement :

①- L'exécution du corps de la boucle.

②- Evaluation de la condition.

Et cela de manière répétitive jusqu'à ce que la condition soit vérifiée (devienne égale à Vrai).

Remarques

①- Dans la boucle Répéter, la séquence d'instructions qui forme le corps de la boucle est exécuté au moins une fois. La première exécution n'est soumise à aucune condition.

②- La formulation de l'expression booléenne doit permettre l'arrêt de l'itération. Dans le cas contraire, on est en présence d'une boucle infinie.

La boucle TantQue (TQ)

Avec le schéma répéter toute exécution du corps de la boucle sauf la première est soumise à condition. Le schéma tantque est un schéma plus général dans lequel toute exécution du corps de la boucle même la première est soumise à condition.

Syntaxe : **TQ** (Condition) **FAIRE**

| Séquence

FTQ

Sémantique

Le déroulement de la boucle Tantque provoque successivement et de manière répétitive :

①- L'évaluation de la condition.

②- L'exécution éventuelle (si la condition est satisfaite) du corps de la boucle (séquence d'instructions).

Jusqu'à ce que la condition soit fausse.

Remarques :

①- Dans la boucle TQ la séquence d'instructions qui forme le corps de la boucle peut ne jamais être exécuté dans le cas où la condition est fausse dès le départ.

②- Comme pour la boucle Répéter, La formulation de l'expression booléenne doit permettre l'arrêt de l'itération. Dans le cas contraire, on est en présence d'une boucle infinie. Il y a certainement une erreur !

Synthèse sur les boucles

- ✓ La boucle Pour permet d'exprimer les itérations contrôlées par un compteur. Si on connait le point de départ et le point d'arrivée et le pas qui est constant, on préfère utiliser la boucle Pour.
- ✓ La boucle Répéter permet d'exprimer les itérations dans lesquelles la première exécution du corps de la boucle n'est pas soumise à une condition.
- ✓ La boucle TQ permet d'exprimer toute itération. Elle peut remplacer les deux autres boucles Pour et Répéter.

4- Structuration d'un algorithme

✓ Les instructions structurées peuvent être imbriquées (L'une à l'intérieur d'une autre) ou disjointes (L'une à la suite d'une autre) mais ne peuvent en aucun cas se chevaucher (Se mélanger).

Exemple 1

```
Début
    …
    Répéter
        …
        Si( Condition 1 ) Alors
            Séquence 1
        FSi
        …
        Séquence 2
    Jusqu'à( Condition 2 )
    …
Fin
// Deux structures imbriquées
```

Exemple 2

```
Début
    …
    Répéter
        Séquence 1
    Jusqu'à( Condition 1 )
    …
    Si( Condition 2 ) Alors
        Séquence 2
    FSi
    …
Fin
// Deux structures disjointes.
/* Exemple d'un commentaire long */
```

✓ L'indentation (manière d'arranger ou d'organiser les instructions) est très utile pour mettre en évidence la structure des algorithmes et en faciliter la lecture.

5- Règles de base de construction d'un algorithme

- ✓ Définir clairement et sans ambiguïtés le problème posé.
- ✓ Un algorithme doit être bien structuré.
- ✓ Le résultat doit être atteint en un nombre fini d'étapes. Faites attention aux boucles infinies.

- ✓ Un algorithme doit être générique : il faut étudier tous les cas possibles de données.
- ✓ Le résultat doit répondre au problème demandé. Attention, un jeu d'essais ne prouve JAMAIS qu'un programme est correct à 100%. Il peut seulement prouver qu'il est faux.
- ✓ Un algorithme (programme) doit être commenté. Des commentaires peuvent être insérés pour expliquer certaines parties de l'algorithme.

Chapitre 5 : Types Structurés

Une variable de type structuré regroupe sous un même nom plusieurs valeurs appartenant à des types constituants déjà définis. Ces types constituants peuvent à leur tour être structurés.

1- Les tableaux

Un tableau est un ensemble de données qui sont toutes du même type et qui possèdent un identificateur unique (nom du tableau) et se différencient les unes des autres dans ce tableau par leur numéro d'indice.

Exemples : Tableau de notes d'un module, tableau de températures sur un mois.

On peut représenter un tableau par un ensemble de cases repérées par leurs indices (positions, rangs) dans le tableau.

Notes	1	2	3	…				N–1	N
	11.5	13	9.75	…				12.25	10.5

Syntaxe de déclaration d'un tableau

Var nomTableau : **Tableau[**indiceMminimal **..** indiceMaximal**] de** TypeDeDonnées

Exemple : Var Notes : Tableau[1 .. 10] de Réel

Dans cet exemple, on a déclaré un tableau appelé Notes d'indice minimal 1 et d'indice maximal 10 qui contient donc 10 cases de nombres réels.

- ✓ Pour déclarer un tableau il faut lui donner un nom, une valeur d'indice minimale et une valeur d'indice maximale.
- ✓ Il faut déclarer aussi une variable de type entier qui servira d'indice pour accéder aux différentes cases (le compteur i par exemple).

Accès aux éléments d'un tableau

On accède à un élément d'une variable Tab de type tableau par une variable indicée qui s'écrit : Tab[Expression]

Expression: est une expression de type entier qui détermine l'indice (le rang, la position) de l'élément sélectionné dans le tableau.

Exemple **:** Notes[5] : désigne l'élément de rang 5 dans le tableau Notes.

Manipulation des tableaux

Chaque élément du tableau se comporte comme toute variable de même type de données. Notes[5] : est un réel. Cette case peut être lue à partir du clavier, affichée à l'écran, être utilisée dans une comparaison ou dans une affectation.

✓ **L'écriture dans le tableau (La modification)**

```
Lire( Notes[ 5 ] )
Notes[ 5 ] ← 11.75
```

✓ **La lecture d'un élément du tableau (La consultation)**

```
X← (Notes[ 5 ] + Notes[ 6 ]) / 2   // Utilisation dans une expression
Si ( Notes[ 5 ] ≥ 10) alors   // Comparaison
    Ecrire("Admis")
Sinon
    Ecrire("Ajourné")
FSi
```

✓ **Le parcours**

parcourir un tableau consiste à passer par chaque élément du tableau une et une seule fois pour effectuer le même traitement sur chacun des éléments. En particulier on ne peut pas lire ou écrire un tableau globalement, on doit le parcourir pour saisir ou afficher ses éléments un par un.

Remarques

①- Le nom d'un tableau n'est jamais utilisé seul. Dans toutes les instructions (saisie, affichage, calcul, test, …), il est toujours suivi d'un indice entre crochets.

②- En plus de la lecture et l'écriture, toute autre opération sur un tableau doit être réalisée par algorithme. Exemple : comparaison entre deux tableaux : Il faut comparer les éléments des deux tableaux deux à deux pour savoir si les deux tableaux sont identiques ou différents.

Exemple 1 : Ce premier exemple permet de montrer comment lire les éléments d'un tableau Notes de 10 éléments puis les afficher par la suite.

```
Algorithme    LectureAffichageTableau
Const  N = 10
Var    Notes : Tableau[ 1 .. N ] de Réel
       i : Entier
Début
    // Lecture des notes – Saisie du tableau
    Pour i ← 1 à N Faire
        Ecrire ("Donner la note de l'étudiant N° ", i )
        Lire( Notes[ i ] )
    FPour
```

```
        // Affichage des notes saisies – Affichage du tableau
        Pour i ← 1 à N Faire
            Ecrire( Notes[ i ] )
        FPour
Fin
```

Exemple 2 : Un algorithme qui calcule la moyenne de 10 notes.

```
Algorithme    Moyenne
Const  N = 10
Var    Notes : Tableau[ 1 .. N ] de Réel
       i : Entier
       S, Moy : Réel
Début
        // Lecture des notes – Saisie du tableau
        Pour i ← 1 à N Faire
            Ecrire ("Donner la note de l'étudiant N° ", i )
            Lire( Notes[ i ] )
        FPour
        S ← 0
        // Calcul de la somme des notes
        Pour i ← 1 à N Faire
            S ← S + Notes[ i ]
        FPour
        Moy ← S / N
        Ecrire("Moyenne = ", Moy)
Fin
```

2- Les Matrices

Lorsqu'un traitement utilise plusieurs tableaux à une dimension qui ont le même nombre d'éléments et subissent le même traitement, on utilise un seul tableau à deux dimensions appelé **matrice**.

Cette nouvelle forme de tableau possède un identifiant unique. Chaque élément est identifié par deux indices :

①- indice de ligne.

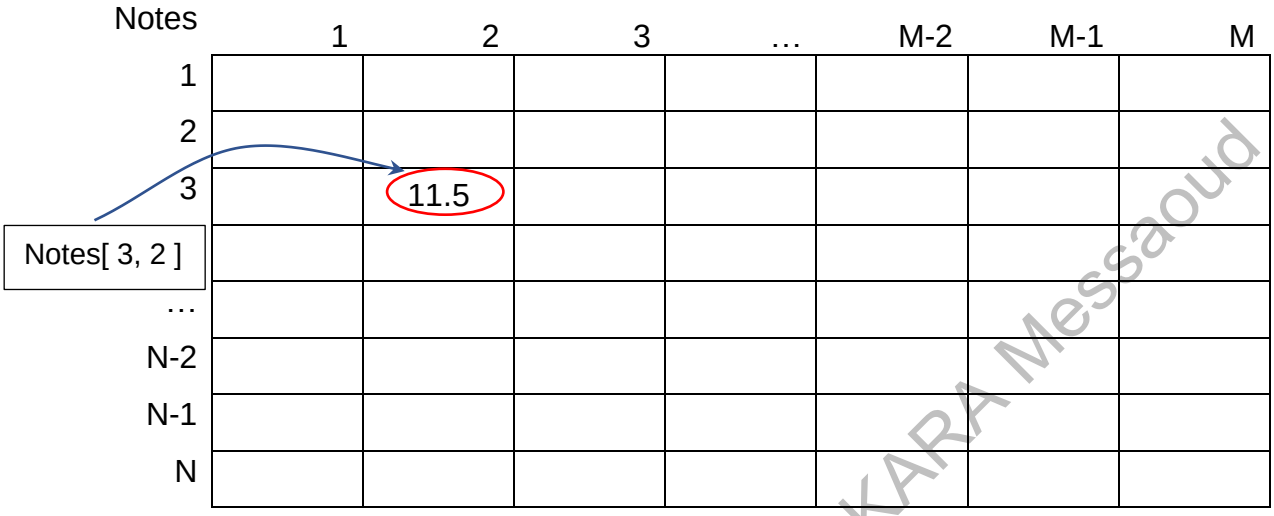
②- indice de colonne.

Exemples **:** ①- Les notes des étudiants dans tous les modules. ②- Un système d'équations linéaires. ③- Représentation d'une image formée par des pixels (points).

On représente une matrice (un tableau à deux dimensions) par un ensemble de cases repérées par leurs positions dans la matrice :

①- Numéro de ligne,

②- Numéro de colonne.



Remarque :

Si le nombre de lignes est égal au nombre de colonne (égal à N), on dit que la matrice est carrée de taille (ou d'ordre) N.

Syntaxe de déclaration d'une matrice

Un tableau à deux dimensions (une matrice) est déclaré comme suit :

Var nomTableau :**Tableau[** indiceLigneMminimal **..** indiceLigneMaximal,
indiceColonneMminimal **..** indiceColonneMaximal **] de** TypeDeDonnées

Exemple **:** Notes : Tableau[1 .. 100, 1..10] de Réel.

Dans cet exemple, on a déclaré un tableau appelé Notes de 100 lignes et de 10 colonnes. Contenant des valeurs réelles.

Remarque : Les tableaux les plus fréquemment utilisés sont les tableaux à une et à deux dimensions, mais il est possible de définir des tableaux à trois ou quatre dimensions, voire plus.

Accès aux éléments d'une matrice

On accède à un élément d'une variable Tab de type matrice (tableau à deux dimensions) par une variable indicée dont la syntaxe est : Tab[Exp1, Exp2]

Exp1 et Exp2 sont deux expressions, de type entier, qui déminent le numéro de ligne et numéro de colonne de l'élément sélectionné dans le tableau.

Exemple **:** Notes[3, 2] : désigne l'élément qui se trouve à la 3ième ligne et la 2ième colonne dans la matrice Notes. Cet élément désigne, par exemple, la note du 3ième étudiant au 2ième module.

Manipulation des Matrices

Chaque élément de la matrice se comporte comme toute variable de même type de données. Notes[3, 2] : est un réel. Cette case peut être lue à partir du clavier, affichée à l'écran, être utilisée dans une comparaison ou dans une affectation.

- ✓ **L'écriture dans la matrice (La modification)**

```
Lire( Notes[ 3, 2 ] )
Notes[ 3, 2 ] ← 11.75
```

- ✓ **La lecture d'un élément de la matrice (La consultation)**

```
X← (Notes[ 3, 1 ] + Notes[ 3, 2 ]) / 2   // Utilisation dans une expression
Si ( Notes[ 3, 2 ] ≥ 10) alors   // Comparaison
    Ecrire("Module acquis")
Sinon
    Ecrire("Module n'est pas acquis")
FSi
```

- ✓ **Le parcours**

Parcourir une matrice (tableau à deux dimensions) consiste à passer par chaque élément de la matrice une et une seule fois (en général, ligne par ligne et colonne par colonne) pour effectuer le même traitement sur chacun des éléments. En particulier on ne peut pas lire ou écrire une matrice globalement, on doit la parcourir pour saisir ou afficher ses éléments un par un.

Remarques

①- Le nom d'une matrice n'est jamais utilisé seul. Dans toutes les instructions (saisie, affichage, calcul, test, …), il est toujours suivi de deux indices entre crochets séparés par une virgule.

②- En plus de la lecture et l'écriture, toute autre opération sur une matrice doit être réalisée par algorithme. Exemple : Comparaison entre deux matrices : Il faut comparer les éléments des deux matrices deux à deux pour savoir si les deux matrices sont identiques ou différentes.

Exemple 1 : Ce premier exemple permet de montrer comment lire les éléments d'une matrice Notes de 40 éléments (10 lignes et 4 colonnes) puis les afficher par la suite.

```
Algorithme     LectureAffichageMatrice
Const  N = 10
       M = 4
Var    Notes : Tableau[ 1 .. N, 1 .. M ] de Réel
       i, j : Entier
Début
       // Lecture des notes – Saisie de la matrice
       Pour i ← 1 à N Faire                  // Répéter pour chaque ligne
              Pour j ← 1 à M Faire           // Répéter pour chaque colonne
                     Ecrire ("Donner la note de l'étudiant N° ", i, " au module ", j )
                     Lire( Notes[ i, j ] )
              FPour
       FPour
       // Affichage des notes saisies – Affichage de la matrice
       Pour i ← 1 à N Faire  // Ligne par ligne
              Pour j ← 1 à M Faire  // Colonne par colonne
                     Ecrire( Notes[ i, j ] )
              FPour
       FPour
Fin
```

Exemple 2 : Un algorithme qui calcule la moyenne générale de 10 étudiants qui ont 7 modules. On suppose que tous les modules ont le même coefficient.

```
Algorithme MoyenneGénérale
Const  N = 10
       M = 7
Var    Notes : Tableau[ 1 .. N, 1 .. M ] de Réel
       i, j : Entier
       S, Moy : Réel
Début
       // Lecture des notes – Saisie de la matrice
       Pour i ← 1 à N Faire           // Répéter pour chaque ligne (Chaque étudiant).
              Pour j ← 1 à M Faire  // Répéter pour chaque colonne (Chaque module).
                     Ecrire ("Donner la note de l'étudiant N° ", i, " au module ", j )
                     Lire( Notes[ i, j ] )
              FPour
       FPour
```

```
    // Calcul des moyennes
    Pour i ← 1 à N Faire  // Pour chaque étudiant
        // Calcul de la moyenne.
        S ← 0
        // Calcul de la somme des notes de l'étudiant i
        Pour j ← 1 à M Faire
            S ← S + Notes[ i, j ]
        FPour
        Moy ← S / M
        Ecrire("Moyenne de l'étudiant ", i , " = ", Moy)
    FPour
Fin
```

Chapitre 6 : Les types personnalisés

En plus des types prédéfinis (standards), l'utilisateur peut définir de nouveaux types. Nous nous intéressons principalement aux types : énumération et enregistrement.

1- Enumération (Type énuméré)

On construit un nouveau type énuméré en explicitant son domaine de valeurs par énumération de la liste ordonnée de ses constantes. Les constantes du type sont des constantes nommées.

- ✓ Tout type doit être déclaré avant son utilisation (sauf les types prédéfinis : Entier, …).
- ✓ Une énumération est un nouvel objet de nature type défini par un nom identifiant et un domaine de valeurs.

Exemples **:**

Type Jour = (Samedi, Dimanche, Lundi, Mardi, Mercredi, Jeudi, Vendredi)
Mois = (Janvier, Février, Mars, Avril, Mai, Juin, Juillet, Août, Septembre, Octobre, Novembre, Décembre)

Var J1, J2 : Jour
M : Mois

- Les variables J1 et J2 ne peuvent prendre que l'une des valeurs : Samedi, ..., Vendredi.
- La variable M ne peut prendre que l'une des valeurs : Janvier, …, Décembre.

- ✓ Les constantes d'une énumération sont liées par une relation d'ordre définie par la position des valeurs dans l'énumération. Donc, l'ordre dans lequel sont listés les identificateurs est significatif. Exemples : Samedi < lundi et Décembre > Janvier.
- ✓ Les noms attribués aux différentes constantes d'une énumération ne peuvent en aucun cas être réutilisés.
  Var Samedi : entier erreur !!!

Les fonctions définies sur les types énumérés sont :

- ✓ **Ord**(x) : Cette fonction renvoie un entier positif correspondant au rang de x dans la liste.
- ✓ **Succ**(x) : Cette fonction fournit la constante qui suit immédiatement la valeur de x dans l'énumération. Le successeur de la dernière valeur n'est pas défini.
- ✓ **Pred**(x) : Cette fonction fournit la constante qui précède immédiatement la valeur de x dans l'énumération. Le prédécesseur de la première valeur n'est pas défini.

  Exemples :

- Ord(Samedi) = 1, Ord(Dimanche) = 2, .. Ord(Vendredi) = 7.
- Succ(Samedi) = Dimanche, Succ(Dimanche) = Lundi, …, Succ(Vendredi) = ? (n'est pas défini).
- Pred(Vendredi) = Jeudi, Pred(Jeudi) = Mercredi, …, Pred(Samedi) = ? (n'est pas défini).

2- Les enregistrements (Les structures)

Définition

Un enregistrement (structure) est une structure de données permettant de regrouper dans une seule entité un ensemble de données de types différents associées à un même et seul objet.

Un enregistrement (appelé aussi structure ou article) est constitué de composants appelés champs.

Chaque champ est identifié par un nom qui permet d'y accéder directement et un type.

Le type d'un champ est quelconque : simple ou structuré.

Déclaration

```
Type  Nom_enregistrement = Enregistrement
                             Nom_champ1 : Type1
                             Nom_champ2 : Type2
                             …
                             Nom_champN : TypeN
                           Fin
```

- ✓ Nom_champ1, Nom_champ2, …, Nom_champN : Sont les identifiants des champs de l'enregistrement.
- ✓ Type1, Type2, …, TypeN : Sont les types associés aux champs.
- ✓ Une fois le type de l'enregistrement est défini, on peut déclarer des variables de ce type :

  Var Nom_variable : Nom_enregistrement

Exemple **:** Les informations concernant un étudiant : nom, prénom, age, sexe, Moyenne du BAC peuvent être représentées à l'aide d'un enregistrement comme suit :

```
Type  Etudiant = Enregistrement
                   Nom : Chaine
                   Prenom : Chaine
                   Age : Entier
                   Sexe : Caractère // 'M' : Masculin, 'F' : Féminin
                   MoyenneBac : Réel
                 Fin
Var   Etudiant1, Etudiant2 : Etudiant      // Deux structures (variables) de type Etudiant
      e1, e2, e3 : Etudiant                // Trois structures (variables) de type Etudiant
```

- ✓ Un enregistrement peut être représenté par un ensemble de cases. Ces cases peuvent être des tailles différentes, car les types d'un enregistrement ne sont pas forcément les mêmes comme pour un tableau.

	Nom	Prenom	Age	Sexe	MoyenneBAC
Etudiant1	**"LAKAB"**	**"Ism"**	**19**	**'M'**	**12.5**

Accès aux champs d'un enregistrement

On accède à une information en précisant le nom de la variable de type enregistrement suivie du nom du champ séparé par un point(.) :

Nom_variable . Nom_champ

Exemple : Pour modifier l'âge de l'étudiant 1 en utilisant l'affectation on doit écrire :

Etudiant1.Age ← 20

- ✓ Le point qui figure dans la syntaxe indique le chemin d'accès : On accède d'abord à la variable Etudiant1 puis on sélectionne le champ Age (ou la partie Age).

Remarque : L'accès aux champs se fait par les noms des champs, donc, l'ordre de déclaration de ces champs n'est pas important.

3- Cas des structures imbriquées

Un enregistrement peut être imbriqué dans une structure de type tableau ou enregistrement, comme il peut avoir des champs de type structuré quelconque. La notation utilisée pour sélectionner les champs reste la même (L'utilisation du point).

Tableaux d'enregistrements

Il est possible de déclarer un tableau dont les éléments sont de type enregistrement. On définit d'abord le type enregistrement, puis on déclare un tableau dont les éléments sont de ce type d'enregistrement.

Type Nom_enregistrement = **Enregistrement**
    Nom_champ1 : Type1
    Nom_champ2 : Type2
    …
    Nom_champN : TypeN
Fin

Var Nom_tableau : **Tableau [** 1 .. N **] de** Nom_enregistrement

Accès aux éléments

On accède d'abord à une case tableaux, en utilisant les crochets [ ], puis on accède au champ en utilisant le point (.).

Pour sélectionner le deuxième champ du troisième élément du tableau on utilise la syntaxe :

Nom_tableau[3] . Nom_champ2

Exemple : Pour déclarer un tableau d'enregistrements pour manipuler les informations de 100 étudiants, on doit écrire :

Type Etudiant = **Enregistrement**

Nom : Chaine
Prenom : Chaine
Age : Entier
Sexe : Caractère // 'M' : Masculin, 'F' : Féminin
MoyenneBac : Réel

Fin

Var Tab : Tableau[1 .. 100] d'Etudiant // Tableau de 100 enregistrements.

Pour modifier les champs de l'étudiant 2, on peut écrire, par exemple :

```
Tab[ 2 ] . Nom ← "Mohamed"
Tab[ 2 ] . Prenom ← "Lakabahou"
Tab[ 2 ] . Age ← "18"
Tab[ 2 ] . Sexe ← 'M'
Tab[ 2 ] . MoyenneBac ← 11.7
```

Manipulation des enregistrements :

L'écriture (La modification)

```
Lire ( Tab[ 2 ] . Age )
Tab[ 3 ] . MoyenneBAC ← 13
```

La lecture (La consultation)

```
X← Tab[ 5 ] . Prenom
Si ( Tab[ 2 ] . MoyenneBAC < 12 ) alors      // Comparaison
        Ecrire("Admis en MI")
FSi
```

Remarques

Toute opération sur les enregistrements doit être effectuée par un algorithme : La lecture, l'écriture, la comparaison ne peuvent se faire globalement, il faut lire, écrire ou comparer chaque champ individuellement (un par un).

Une exception : On peut affecter une variable de type enregistrement à une autre variable du même type.

Exemple : **Var** e1, e2 : Etudiant

On peut écrire indifféremment : e2 ← e1 ou champ par champ :

```
e2 . Nom ← e1 . Nom
e2 . Prenom ← e1 . Prenom
e2 . Age ← e1 . Age
e2 . Sexe ← e1 . Sexe
e2. MoyenneBac ← e1. MoyenneBac
```

Chapitre 7 : Cours sur le langage C

Remarque : Le cours sur le langage C est divisé en plusieurs parties intégrées aux séries de TPs. Chaque série, contient la partie nécessaire pour pouvoir la travailler sereinement.

La série N°1 est consacrée à l'outils utilisé (Code::Blocks), la traduction d'un algorithme en langage C, les types standards et les instructions élémentaires.

La série N°2 est réservée aux structures conditionnelles.

La série N°3 est dédiée aux boucles.

La série N°4 est dédiée aux tableaux (et aux matrices) ainsi qu'aux chaines de caractères.

Références Bibliographiques

[1] M. BELAID, Algorithmique & Programmation en PASCAL, Cours, Exercices, Travaux Pratiques, Corrigés, Bouira-Algérie: Eurl Pages Bleues Internationales, 2008.

[2] D.-E. ZEGOUR, Structures de Données et de Fichiers Programmation PASCAL et C, Alger-Algérie: Editions Chihab, 1996.

[3] J. COURTIN, I. KOWARSKI et J. ARSAC, Initiation à l'algorithmique et aux structures de données 1. Programmation structurée et structures de données élémentaires, Paris - France: Dunod, 1994.

[4] J. COURTIN et I. KOWARSKI, Initiation à l'algorithmique et aux structures de données 2. Récursivité et structures de données avancées, Paris - France: Dunod, 1995.

[5] C. DELANNOY, Apprendre à programmer en Turbo C, Alger-Algérie: Chihab - Eyrolles, 1994.

[6] K. KHALFAOUI, «Introduction à la programmation, Support de Cours,» Université de Jijel, Jijel, 2018.

[7] V. L. Damien Berthet, Algorithmique & programmation en langage C vol.1 : Supports de cours., Istanbul, Turquie : Université Galatasaray, 2014.

[8] V. L. Damien Berthet, Algorithmique & programmation en langage C vol.2 : Sujets de travaux pratiques., Istanbul, Turquie: Université Galatasaray, 2014.

[9] V. L. Damien Berthet, Algorithmique & programmation en langage C vol.3 : Corrigés de travaux pratiques., Istanbul, Turquie: Université Galatasaray, 2014.