UNIVERSITE MUSTAPHA STANBOULI MASCARA

Faculté des Sciences Economiques, Commerciales et des Sciences de Gestion

Département des sciences commerciales

Polycopié de cours

Algorithmique

Cours et travaux dirigés

Dr .SAHRAOUI Mustapha

Année 2018

Avant propos

Ce polycopié est rédigé à l'intention des étudiants de la deuxième année du premier cycle universitaire (licence en sciences commerciales, licence en économie, licence en gestion) de la Faculté des Sciences Economiques, Commerciales et des Sciences de Gestion (FSECSG). Il constitue un manuel de cours et d'exercices sur une partie du domaine de l'algorithmique. Les lecteurs ne nécessitent aucun pré-requis sur l'algorithmique.

Ce polycopié est structuré en six chapitres comme suit :

- Dans le premier chapitre, nous avons donné les notions de base sur la structure globale d'un algorithme, ainsi que les différentes parties qui le composent suivie par les instructions de base les plus élémentaires.
- Le deuxième chapitre décrit les différents types d'expression qu'on peut rencontrer dans les algorithmes et la façon de les évaluer, ainsi que les notions de variable, de constante et de type, et les règles d'écriture d'un identificateur.
- Le troisième chapitre décrit les différentes instructions dans un algorithme telles que les instructions de traitement de base qui sont *Ecrire*, *Lire*, et l'instruction d'*affectation*, les instructions de contrôle qui permettent de choisir entre deux traitements ou de répéter plusieurs fois un même traitement.
- Le quatrième chapitre est consacré aux tableaux où nous allons nous familiariser avec les tableaux à une dimension (les vecteurs) et les tableaux à deux dimensions (les matrices).
- Dans le cinquième chapitre les sous-programmes (procédures et fonctions) sont donnés.
- Enfin, dans le sixième chapitre nous traitons l'utilisation des enregistrements et des fichiers dans l'algorithmique.

Nous trouverons en fin de ce manuscrit quelques sujets d'examens et leurs corrigés, et nous donnons également une liste de références bibliographiques.

Sommaire

Contexte scientifique

Le terme informatique est un néologisme proposé en 1962 par Philippe Dreyfus pour Caractériser le traitement automatique de l'information : il est construit sur la contraction de l'expression « information automatique ». Ce terme a été accepté par l'Académie française en avril 1966, et l'informatique devint alors officiellement la science du traitement automatique de l'information, où l'information est considérée comme le support des connaissances humaines et des communications dans les domaines techniques, économiques et sociaux (figure 1.1). Le mot informatique n'a pas vraiment d'équivalent aux Etats-Unis où l'on parle de *Computing Science* (science du calcul) alors que *Informatics* est admis par les Britanniques.

Définition de l'informatique

L'informatique est la science du traitement automatique de l'information. L'informatique traite de deux aspects complémentaires : les programmes immatériels (logiciel, software) qui décrivent un traitement à réaliser et les machines (matériel, hardware) qui exécutent ce traitement. Le matériel est donc l'ensemble des éléments physiques (microprocesseur, mémoire, écran, clavier, disques durs. . .) utilisés pour traiter les données. Dans ce contexte, l'ordinateur est un terme générique qui désigne un équipement informatique permettant de traiter des informations selon des séquences d'instructions (les programmes) qui constituent le logiciel.

Le terme ordinateur a été proposé par le philologue Jacques Perret en avril 1955 en réponse à une demande d'IBM France qui estimait le mot « calculateur » (computer) bien trop restrictif en regard des possibilités de ces machines.

Définition du matériel

Le matériel informatique est un ensemble de dispositifs physiques utilisés pour traiter automatiquement des informations.

Définition du logiciel

Le logiciel est un ensemble structuré d'instructions décrivant un traitement d'informations à faire réaliser par un matériel informatique.

Un ordinateur n'est rien d'autre qu'une machine effectuant des opérations simples sur des séquences de signaux électriques, lesquels sont conditionnés de manière à ne pouvoir prendre que deux états seulement (par exemple un potentiel électrique maximum ou minimum). Ces séquences de signaux obéissent à une logique binaire du type « tout ou rien » et peuvent donc être considérés conventionnellement comme des suites de nombres ne prenant jamais que les deux valeurs 0 et 1 : un ordinateur est donc incapable de traiter autre chose que des nombres binaires. Toute information d'un autre type doit être convertie, ou codée, en format binaire.

C’est vrai non seulement pour les données que l’on souhaite traiter (les nombres, les textes, les images, les sons, les vidéos, etc.), mais aussi pour les programmes, c’est-à-dire les séquences d’instructions que l’on va fournir `a la machine pour lui dire ce qu’elle doit faire avec ces données.

L’architecture matérielle d’un ordinateur repose sur le modèle de Von Neumann (figure 1) qui distingue classiquement 4 parties (figure 1) :
1. L’unité arithmétique et logique, ou unit´e de traitement, effectue les opérations de base.
2. L’unit´e de contrôle séquence les opérations.
3. La mémoire contient `a la fois les données et le programme qui dira `a l’unit´e de contrôle quels calculs faire sur ces données. La mémoire se divise entre mémoire vive volatile (programmes et données en cours de fonctionnement) et mémoire de masse permanente (programmes et données de base de la machine).
4. Les entrées-sorties sont des dispositifs permettant de communiquer avec le monde extérieur (écran, clavier, souris. . .).

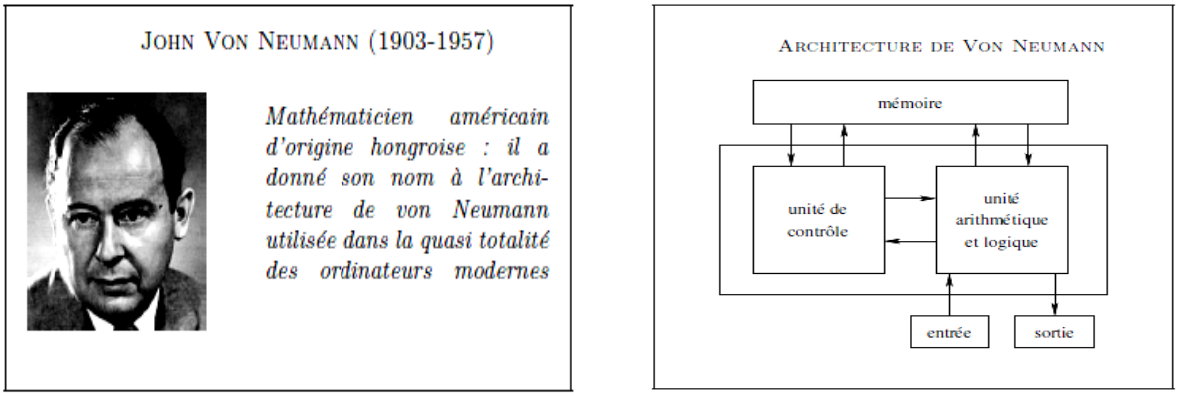


Figure 1

Objectifs du cours

- **Objectifs thématiques**

L’objectif principal des enseignements d’informatique est l’acquisition des notions fondamentales de l’algorithmique. Plus précisément, nous étudierons successivement :
1. les instructions de base permettant de décrire les algorithmes : affectation, tests, boucles ;
2. les procédures et les fonctions qui permettent de structurer et de réutiliser les algorithmes ; on parlera alors d’encapsulation, de pré-conditions, de portée des variables, de passage de paramètres, d’appels de fonctions, de récursivité et de jeux de tests ;
3. les structures de données linéaires : tableaux, listes, piles, files, qui améliorent la structuration des données manipulées par les algorithmes.

- **Objectifs pédagogiques**

Au cours du semestre S1, nous nous positionnerons principalement sur les 3 premiers niveaux de la taxonomie de Bloom qui constitue une référence pédagogique pour la classification des niveaux d'apprentissage : connaissance, compréhension, application. Les 3 derniers niveaux seront plutôt abordés au cours du semestre S2 (analyse, synthèse, évaluation).

1. ***Connaissance*** : mémorisation et restitution d'informations dans les mêmes termes.
2. ***Compréhension*** : restitution du sens des informations dans d'autres termes.
3. ***Application*** : utilisation de règles, principes ou algorithmes pour résoudre un problème, les règles n'étant pas fournies dans l'énoncé.
4. ***Analyse*** : identification des parties constituantes d'un tout pour en distinguer les idées.
5. ***Synthèse*** : réunion ou combinaison des parties pour former un tout.
6. ***Evaluation*** : formulation de jugements qualitatifs ou quantitatifs.

Chapitre I

Notions de base

« *Chaque programme d'ordinateur est un modèle, forgé par l'esprit, d'un processus réel ou imaginaire. Ces processus, qui naissent de l'expérience et de la pensée de l'homme, sont innombrables et complexes dans leurs détails. A tout moment, ils ne peuvent être compris que partiellement. Ils ne sont que rarement modélisés d'une façon satisfaisante dans nos programmes informatiques. Bien que nos programmes soient des ensembles de symboles ciselés avec soin, des mosaïques de fonctions entrecroisées, ils ne cessent d'´evoluer. Nous les modifions au fur et `a mesure que notre perception du modèle s'approfondit, s'´etend et se généralise, jusqu'`a atteindre un équilibre métastable aux frontières d'une autre modélisation possible du problème. L'ivresse joyeuse qui accompagne la programmation des ordinateurs provient des allers-retours continuels, entre l'esprit humain et l'ordinateur, des mécanismes exprimes par des programmes et de l'explosion de visions nouvelles qu'ils apportent. Si l'art traduit nos rêves, l'ordinateur les réalise sous la forme de programmes !* »

Abelson H., Sussman G.J. et Sussman J. [1]

I.1- Notions de processeur, environnement et action

Exemple : Extrait du mode d'emploi d'un télécopieur concernant l'envoi d'un document.

1. Insérez le document dans le chargeur automatique.
2. Composez le numéro de fax du destinataire `a l'aide du pave numérique.
3. Enfoncez la touche envoi pour lancer l'´emission.

Ce mode d'emploi précise comment envoyer un fax. Il est compos´e d'une suite ordonnée d'instructions (insérez, composez, enfoncez. . .) qui manipulent des données (document, chargeur automatique, numéro de fax, pave numérique, touche envoi) pour réaliser la tache désirée (envoi d'un document).

I.1.1- Processeur :

Nous appelons processeur toute entité (vivante ou matérielle) capable de comprendre un tel énoncé et d'exécuter le travail indiqué.

Dans notre cas, une personne sachant lire et disposant des objets nécessaires peut jouer le rôle d'un processeur.

I.1.2- Environnement :

L'ensemble des objets nécessaires à l'exécution d'un travail constitue l'environnement de ce travail.

I.1.3- Action :

L'exécution de tout travail suppose une progression étape après étape vers le but cherché. Dans l'exemple précédent, on distingue des étapes (a, b, c, …, g). Chaque étape est appelée une action. Une action est un événement de durée finie qui modifie l'environnement.

I.2- Action primitive, décomposition d'une action, algorithme

I.2.1- Action primitive :

Pour un processeur donné, une action est primitive si l'énoncé de cette action suffit au processeur pour qu'il puisse l'exécuter sans un supplément d'information.

Exemple :

L'action : "additionner 2 nombres de 2 chiffres chacun" est une action primitive pour un processeur qui est un étudiant à l'université.

I.2.2- Décomposition d'une action :

Supposons maintenant que le processeur est un petit enfant qui ne sait faire qu'additionner des chiffres. Dans ce cas, il faudra lui expliquer :

- ✓ additionner les chiffres des unités puis écrire les résultats,
- ✓ additionner les chiffres des dizaines, écrire le résultat à gauche du précédent (on suppose qu'il n y a pas de retenue).

On dit qu'on a décomposé l'action " additionner 2 nombres de 2 chiffres chacun " en 2 actions primitives.

I.2.3- Algorithme :

- Un algorithme est une suite ordonnée d'instructions qui indique la démarche à suivre pour résoudre une série de problèmes équivalents.
- Etant donné un processeur bien défini et un traitement à exécuter par ce processeur, un algorithme du traitement est une suite (ou liste) d'actions primitives réalisant ce traitement.

Exemple : Trouver son chemin
Extrait d'un dialogue entre un touriste égaré et un villageois.

1. Pourriez-vous m'indiquer le chemin de la gare, s'il vous plait ?
2. Oui bien sur : vous allez tout droit jusqu'au prochain carrefour, vous prenez à gauche au carrefour et ensuite la troisième à droite, et vous verrez la gare juste en face de vous.
3. Merci.

Dans ce dialogue, la réponse de l'autochtone est la description d'une suite ordonnée d'instructions (allez tout droit, prenez à gauche, prenez la troisième à droite) qui manipulent des données (carrefour, rues) pour réaliser la tache désirée (aller à la gare). Ici encore, chacun a déjà été confronté à ce genre de situation et donc, consciemment ou non, a déjà construit un algorithme dans sa tête (ie. définir la suite d'instructions pour réaliser une tache). Mais quand on définit un algorithme, celui-ci ne doit contenir que des instructions compréhensibles par celui qui devra l'exécuter (des humains dans les 2 exemples précédents).
Dans ce cours, nous devrons apprendre `a définir des algorithmes pour qu'ils soient compréhensibles (et donc exécutables) par un ordinateur.[2]

Remarque :
i) le mot "algorithme" vient du nom d'un mathématicien ouzbek du 9ème siècle : El Khawarismi.
ii) dans la définition, on dit "un" algorithme et non pas "l'" algorithme car, en général, la solution n'est pas unique.

I.3- Programmation

a) Un algorithme exprime la structure logique d'un programme informatique et de ce fait est indépendant du langage de programmation utilisé. Par contre, la traduction de l'algorithme dans un langage particulier dépend du langage choisi et sa mise en œuvre dépend également de la plateforme d'exécution.
b) La programmation consiste à établir (déterminer) une suite d'actions pouvant être exécutées par un processeur pour réaliser un travail donné.

La programmation est constituée de trois phases :

a- La résolution du problème, c'est-à-dire la recherche d'un algorithme réalisant le traitement.

Problème à résoudre

↓

Enoncé clair et précis délimitant le problème
- données du problème (que connait-on ?)

↓

Analyse dans le langage naturel (langue de tous les jours) des actions à réaliser.

↓

Algorithme (analyse dans un langage algorithmique)
- résultats cherchés ?

b- L'adaptation de l'algorithme au processeur. Elle est réalisée par la codification de l'algorithme dans un langage de programmation donné.

Algorithme ⟶ Codification Ou Traduction dans un langage de Programmation ⟶ Programme

c- L'exécution du programme sur une machine.

Programme ⟶ Tests + Corrections ⟶ Programme utilisable

Chapitre II

Les expressions et les données d'un Algorithme

II.1- Définition

Une expression désigne le calcul d'une valeur à partir d'autres valeurs (variables, constantes) et d'opérations.

Exemples :

- a + 3 * b – c
- 'INF' + '3'
- a < b

II.2- Différents types d'expressions

Suivant les types de variables et des opérations, on distingue différentes expressions :

II.2.1- Expressions arithmétiques :

Valeurs : des entiers ou des réels
Opérations : +, -, *, /, DIV, MO**D**

Exemples :

a et b étant des variables numériques :

- (a + b) / 2.5
- a – 4.75 / (2 * b)

Remarque :

Les éléments constituant une expression (variables, constantes et opérateurs) sont écrits sur la même ligne.

II.2.2- Expressions booléennes :

Valeurs : des booléens
Opérations : ET, OU, NON, OUX

Exemples :

- a **ET** (b **OU** c)
- **NON** (a) **OU** (b **ET** c)

II.2.3- Expressions caractères :

Valeurs : caractères, chaînes de caractères
Opérations : + (ou concaténation)

Exemple :

- 'PAS' + 'CAL' donnera PASCAL

II.2.4- Expressions de relation (comparaison) :

Valeurs : tous les types prédéfinis
Opérations : <, =, >, <=, >=, <>
Exemples :

- c = 'c'
- a < (b + 5)
- p <= 0.05

Remarque :
Le résultat d'une expression de ***relation*** est un booléen.

II.3- Evaluation des expressions

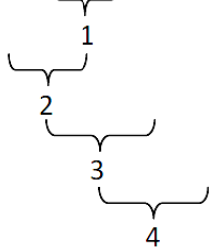
L'ordre des opérations constituant une expression obéit aux règles suivantes :
a- les opérations entre parenthèses sont évaluées les premières
b- les opérations sont évaluées de :
- de gauche à droite lorsqu'elles ont des priorités égales,
- la plus prioritaire à la moins prioritaire lorsqu'elles ont des priorités inégales.

c- les priorités des opérateurs sont résumées dans le tableau suivant :

Priorités		Opérateurs
+ ↓ -	1	NON, - (unaire)
	2	*, /, DIV, MOD, ET
	3	+, - (binaire), OU, OUX
	4	<, <=, =, >, <>

Exemples :

- 7 – 5 + 2 vaut : 7 – 5 = 2 puis 2 + 2 = 4
- 7 – (5 + 2) vaut : 5 + 2 = 7 puis 7 – 7 = 0
- 7 – 3 * 2 vaut : 3 * 2 = 6 puis 7 – 6 = 1
- (a + b * c) / d * e vaut $\frac{a+b+c}{d} * e$



II.4- Notion de variable

- Les valeurs (données, résultats intermédiaires, résultats finals) utilisées dans un algorithme sont représentés par des variables.
- Une variable est un emplacement (case) de la mémoire de l'ordinateur.
- Un algorithme peut modifier (ou varier → variable) à tout moment la valeur d'une variable.

Remarque :
Une mémoire d'ordinateur peut être vue comme un meuble de rangement avec une multitude de tiroirs ou cases.

II.5- Caractéristiques d'une variable

Une variable est caractérisée (définie) par son ***nom***, son ***type*** et sa ***valeur***.

II.5.1- Le nom (identificateur) d'une variable :

- Permet de la retrouver (repérer) dans la mémoire de l'ordinateur
- C'est une suite de caractères (lettres, chiffres, souligné ou tiret -8) dont le premier ne doit pas être un chiffre.

Exemples :
- Age, Note1, Ecart_type, … sont des identificateurs valides (corrects)
- 2Tiers, M5 ?, Ma note, … ne sont pas des identificateurs valides.

Remarques :
- Le nom d'un algorithme doit obéir aux règles d'écriture des noms de variables.
- Afin d'améliorer la lisibilité d'un algorithme, il est conseillé de choisir des noms de variables significatifs.
- Les noms de variables doivent être différents des mots clés (ou mots réservés) : Algorithme, Début, Fin, Si, Alors, … de votre pseudo-langage.

II.5.2- Le type d'une variable :

Détermine l'ensemble des valeurs qu'elle peut prendre. Les types que nous utiliserons seront décrits dans la suite du chapitre.

II.5.3- La valeur d'une variable :

C'est un élément quelconque de l'ensemble défini par le type de la variable. Les valeurs que nous traiterons seront essentiellement des nombres et du texte.

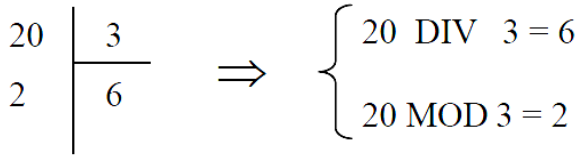
II.6- Les types prédéfinis

Sont les types de base disponibles dans la plupart des langages de programmation.

II.6.1- le type entier (Integer) :

Est une partie de l'ensemble … des entiers relatifs. Les valeurs de ce type appartiennent à l'intervalle : $[-2^{15}, +2^{15}-1]$.
Les opérations autorisées sur ce type sont : +, -, * (multiplication), **DIV** (division entière), **MOD** (reste de la division entière).

Exemple :



II.6.2- Le type réel (Real) :

Un nombre réel est un nombre qui a une partie décimale comme : -3.5, +12.35, …
Les valeurs de ce type appartiennent à l'intervalle : $[-1{,}7.10^{38}, +1{,}7.10^{38}]$
Les opérations autorisées sur ce type sont : +, -, *, / (division).

Remarque :

Entier / entier ⟶ Réel

Exemple :

7 / 2 = 3.**5**

II.6.3- Le type caractère (Char) :

Représente tous les symboles qu'on trouve sur le clavier d'un ordinateur : espace, lettres, chiffres, signes de ponctuation et caractères spéciaux (#, @, &, …).

Remarque :

Une variable de ce type doit être entre apostrophes (ou ***quotes***). Ainsi, dans un algorithme :
B : désignera un nom de variable
'B' : désignera une valeur (la lettre B majuscule)

II.6.4- Le type chaîne de caractères (String) :

Regroupe toutes les chaînes (suites) de caractères
La longueur maximale d'une chaîne est 255
L'opérateur "+" réalise la concaténation (mise bout à bout) des chaîne de caractères
'université' + 'de'+ 'Mascara' ⟶ 'université de Mascara'

II.6.5- Le type booléen (Boolean) :

Représente les 2 valeurs logiques **VRAI** et **FAUX**, Les opérateurs logiques sont :
- négation logique : NON (NOT)
- conjonction logique : ET (AND)
- disjonction logique : OU (OR)
- exclusion logique : OUX (XOR)

Remarque :

Soit a et b deux variables booléennes :

A	b	NON(b)	A ET b	a OU b	a OUX b
FAUX	FAUX	VRAI	FAUX	FAUX	FAUX
FAUX	VRAI	FAUX	FAUX	VRAI	VRAI
VRAI	FAUX	VRAI	FAUX	VRAI	VRAI
VRAI	VRAI	FAUX	VRAI	VRAI	FAUX

II.7- Définition (déclaration) des variables :

Les variables d'un algorithme sont déclarées dans la partie "définition des données" ou " partie de déclaration" comme suit :
Variables liste d'identificateurs : Type
Ou en Pascal :
Var liste d'identificateurs : Type ;
Exemple :
Var Som, somme,a : Integer ;
Taille, Moyenne : Real;
Chiffre: Char;
mot : string [20] ;
valeur : Boolean ;

II.8- Les constantes

En plus des variables, un algorithme utilise aussi des constantes. Ainsi, pour calculer la moyenne de 2 notes, on a besoin d'écrire :
(Note1 + Note2) / ***2*** où la valeur 2 est une constante.

Chapitre III

Les instructions d'un algorithme

III.1- Introduction

Dans un algorithme, on distingue deux types d'instructions :
- les instructions qui font les ***opérations*** (traitements),
- les instructions qui ***contrôlent*** (indiquent) l'ordre des opérations.

III.2- Les instructions de traitement

Ce sont la ***lecture***, ***l'écriture***, et ***l'affectation***.

III.2.1- L'instruction LIRE (READ) :

Elle permet de lire (introduire) les données du problème :
LIRE (v_1, v_2, …, v_n)
Ou en Pascal
READ (v_1, v_2, …, v_n) ;
Où : v_1, v_2, …, v_n sont des (noms de) variables.
Les ***n*** valeurs introduites au clavier sont affectées dans l'ordre (la première à v_1, la deuxième à v_2, …etc.) aux variables v_1, v_2, …, v_n.

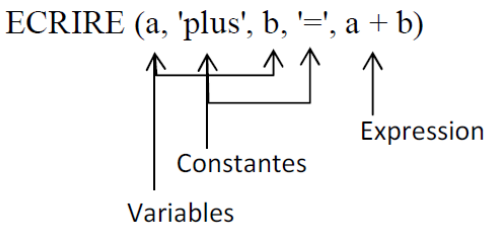
III.2.2- L'instruction ECRIRE (WRITE) :

Elle permet d'écrire (***afficher***) les résultats :
ECRIRE (v_1, v_2, …, v_n)

Ou en Pascal
WRITE (v_1, v_2, …, v_n) ;
Les valeurs des variables : v_1, v_2, …, v_n sont affichées à l'écran.

Remarque :
L'instruction ECRIRE permet aussi d'afficher :
- des constantes (en général des textes),
- des valeurs d'expressions.

On peut par exemple avoir des commentaires :



III.2.3- L'affectation :

Elle permet d'attribuer (de donner) une valeur à une variable.

Nom de variable ⟵ ***e***

Ou en Pascal

Nom de variable := ***e*** ;

Où ***e*** peut être un ***constante***, une autre ***variable*** ou une ***expression***.

Exemples :

Nombre ⟵ 10 ; ***constante***
Som ⟵ Nombre ; ***variable***
Moyenne := (note1 + note2)/2 ; ***expression***

Remarques :

- Dans l'instruction : Som ⟵ nombre ; la valeur de la variable **nombre** ne change pas.
- REEL ⟵ ENTIER est ***autorisé***
- ENTIER ⟵ REEL ***n'est pas autorisé***

Exercice III.1 :

Etant donné l'algorithme suivant :

Algorithme abc ;
Var a, b : entier ;
Debut
Ecrire ('Introduire deux nombre entier SVP ') ;
Lire (a, b) ;
a ⟵ a+b ;
b ⟵ a-b ;
a ⟵ a-b ;
Ecrire ('a=', a);
Ecrire ('b=', b);
Fin.

1- Dérouler cet algorithme pour a=5, b=3.
2- Que fait cet algorithme.

Solution :

1- Les variables recevant des valeurs au cours de l'exécution de l'algorithme sont a et b. la méthode la plus pratique pour dérouler un algorithme et de tracer un tableau avec comme champs de colonnes les variables de l'algorithme (dans notre cas : a et b):

a	b
5	**3**
8	**5**
3	

2- L'algorithme précédent permet d'inter changer (de permuter) les valeurs de deux variables a et b.

III.2.4- Trace d'un algorithme :

La trace (le déroulement) permet de simuler l'exécution d'un algorithme instruction après instruction. On présente la trace sous forme d'un tableau où chaque colonne représente une variable et chaque ligne l'exécution d'une instruction.[4]

Exemple :

Faire la trace de l'algorithme "Echange" pour les valeurs : a = 8 et b = 5.

Variables / Instructions	a	b	c
Au départ	0	0	0
LIRE (a, b)	8	5	
c ← a			8
a ← b	5		
b ← c		8	
ECRIRE (a, b)	5	8	

III.3- les instructions de contrôle

Ces instructions permettent de :

- Choisir entre deux traitements,
- Répéter plusieurs fois un même traitement.

III.3.1- L'instruction SI (IF) :

Elle exprime un choix (une alternative) entre deux possibilités :

SI condition **ALORS**
    Traitement 1
SINON
    Traitement 2
FINSI

Ou en Pascal :

IF condition **THEN**
    Traitement 1
ELSE
    Traitement 2 ;

Cela veut dire que si la condition :

- est satisfaite (***VRAI***) : on exécute traitement 1 et on saute traitement 2
- n'est pas satisfaite (***FAUX***) : on saute Traitement 1 et on exécute Traitement 2.

Remarques :
1- La condition est souvent une comparaison. Si a et b sont des variables, on peut avoir

SI a = 0 **ALORS**
    Traitement 1
SINON
    Traitement 2
FINSI
Ou :
SI a < b **ALORS**
    Traitement 1
SINON
    Traitement 2;
FINSI
2- Les opérateurs de comparaison sont : =, ≠, <, >, ≤, ≥ qu'on traduit en Pascal par : =, <>, <, >, <=, >=.
3- En Pascal, lorsque traitement 1 et/ou traitement 2 sont formés de plusieurs instructions, on doit les insérer entre les mots clés BEGIN et END.
4- Lorsqu'on ne fait rien dans l'un des deux cas déterminés par la condition, l'instruction SI s'écrit :
SI condition **ALORS**
    Traitement;
FINSI
Ou en Pascal :
IF condition **THEN**
    Traitement ;
Attention :
Il ne faut donc jamais mettre un point-virgule (;) avant le ELSE dans la forme complète de l'instruction IF.
Exercice :
Ecrire un algorithme (ou programme) qui lit la moyenne d'un étudiant et qui suivant que celle-ci soit inférieure à 10 ou non affiche le message ou "bonjour".
Exercice :
Ecrire un algorithme qui lit **3** valeurs entières puis affiche la plus grande des trois.

III.3.2- Les instructions de répétition (ou boucles) :
Pour l'algorithme trois instructions de répétition sont fournit : tant que, pour, répéter alors en Pascal : **WHILE**, **REPEAT** et **FOR**.

III.3.2.1- L'instruction POUR (FOR) :
Est utilisée lorsqu'on connait à l'avance le nombre de fois qu'on répète un traitement, comme :
- Calculer les moyennes de 300 étudiants,
- Editer les fiches de paie de 1500 ouvriers,
- …

Syntaxe :
POUR i **de** val_init à val_fin **FAIRE**
Traitement
FINPOUR

Ou en Pascal :
FOR i := val_init **TO** (***DOWN TO***) val_fin **DO**
Traitement ;
Où :
- vl: est la variable de contrôle de la boucle
- val_init : est la valeur initiale (ou valeur de départ)
- val_fin : est la valeur finale de i

Le traitement est répété pour chacune des valeurs de **i** comprises dans l'intervalle [val_init… val_fin].

Remarque :
- l'incrémentation de la val_init se faite automatiquement par la valeur **1** .
-dans le cas implicite, le pas <> 1 est motionné.
- DOWN TO : pour la décrémentation de la val_init jusqu'à la val_fin.

III.3.2.2- L'instruction TANT QUE (WHILE) :

Est la plus générale et peut être utilisée dans tous les cas, voici sa syntaxe :

TANT QUE condition **FAIRE**
Traitement
FINTANT QUE
Ou en Pascal :
WHILE *condition* **DO**
Traitement ;
Le traitement est répété tant que la condition est satisfaite ("**VRAI**").

III.3.2.3- L'instruction REPETER (REPEAT) :

Est utilisée lorsque le traitement à répéter s'exécute au moins une fois, voici sa Syntaxe :

REPETER
Traitement
JUSQU'A condition

Ou en Pascal :
REPEAT
Traitement
UNTIL condition ;

Exercice:
Ecrire un algorithme qui lit un nombre *n* (*n >= 1*) et qui calcule la somme des *n* premiers entiers naturels. Si *n = 6*, l'algorithme calculera : 0 + 1 + 2 + 3 + 4 + 5.

Exercice:
Pour relancer la consommation des ménages et favoriser la croissance, le gouvernement déclare accorder une réduction d'impôt de **10%** aux ménages déclarant un revenu annuel *supérieur ou égal* à **60 000** Dinars.
1- Écrire un algorithme ou un programme pascal utilisant le test **Si** Revenu >= 60 000 **Alors**, et qui affiche ensuite soit "***Vous avez le droit à une réduction d'impôt***", soit "***Vous n'avez pas le droit à une réduction d'impôt***".
2- Reprendre l'exercice précédent, mais à partir du test : **Si Non** (Revenu < 60 000) **Alors**

Exercice:
Ecrire un algorithme ou un programme pascal qui demande un nombre ***n*** (qui insiste qu'il soit supérieur ou égal à 5), calcule la somme des entiers à partir de 3 jusqu'à ce nombre ***n*** et qui affiche le résultat de la manière suivante :
3 + 4 + ... + n = résultat

Chapitre IV
Les tableaux

IV.1- Introduction :

Jusqu'ici, nous avons employé des **variables** pour stocker une seule valeur de types primitifs : une variable de type *integer* pour stocker un entier, une variable de type *real* pour stocker un réel, une variable de type *char* pour stocker un caractère, etc.
Chaque fois que nous avons à traiter plusieurs données (ou valeurs) décrivant un même sujet comme :
- des notes d'étudiants,
- des relevés de température (ou pluviométrie) de différentes régions,
- des populations de différents pays,
- …

La meilleure solution et d'organiser ces données sous forme d'un ***tableau***.

IV.2- Notions de tableaux :

Un tableau est un ensemble (ou collection) de **N** variables simples de ***même*** type (ou nature) appelées : éléments (ou composantes) du tableau.
L'ensemble des N variables porte un seul et même nom.
On peut traiter un tableau comme un tout, ou composante par composante. Traité comme un tout, par exemple le stocker dans une variable ou le passer en paramètre. Chaque composante, désignée par son indice (qui correspond à sa position dans le tableau) peut être traitée comme une seule variable. [3]

Exemples :
Chaque élément du tableau est repéré par un numéro (position) compris entre 1 et N.
L'intervalle d'entiers [1, N] est appelé : ensemble des ***indices*** (ou indexes).

IV.3- Accès aux éléments :

L'accès à un élément particulier d'un tableau se fait grâce à l'opération d'indexage notée :
Nom du tableau [ind]
Ainsi, $\forall\ i \in [1, N]$:
- Nom du tableau [i] nous donne la valeur de l'élément qui se trouve à la position i.
- Nom du tableau [i] n'est pas définie lorsque $i \notin [1, N]$

Exemple :
Si T est le tableau d'entiers suivant :

7	0	3	4	5	6	9	8

T

alors :
T[2] a pour résultat 0
T[6] a pour résultat 6
T[10] n'est pas définie.

Remarques :
a- T[i] est l'équivalent d'une variable
- b ← T[i] : permet de récupérer la valeur de l'élément indice i dans la variable b
- T[i] ← val : permet de charger la valeur val dans le $i^{ième}$ élément du tableau T.

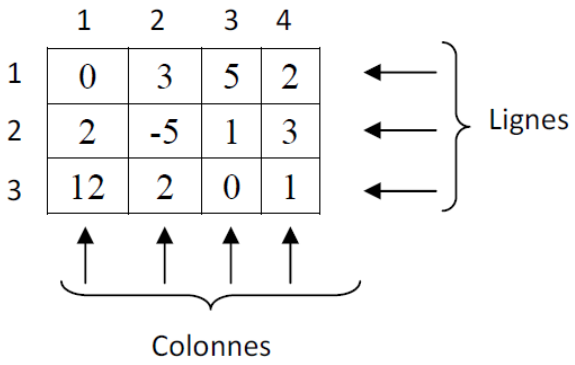
b- Les éléments d'un tableau peuvent être :
- simples : entiers, réels, caractères, …
- structurés : enregistrements, tableaux.

c- Un tableau T est un tableau à une dimension (ou vecteur) si on accède à ses éléments à l'aide d'un seul indice.

IV.4- Les tableaux à deux dimensions (matrices) :

On peut déclarer des tableaux dont lesquels les valeurs ne sont pas repérées par un seul indice, mais par deux indices, le premier indice sert à représenter les lignes et le deuxième indice pour les colonnes.
Autrement dit, une matrice ou un tableau à deux dimensions est constitué de lignes et de colonnes comme le montre l'exemple suivant :

Exemple :
M est une matrice de 3 lignes et 4 colonnes à valeurs entières :



Pour accéder à un élément de la matrice, nous écrivons M[i, j] où :
- i est l'indice (numéro) de la ligne,
- j est l'indice (numéro) de la colonne.

Ainsi :
M [1 , 3] a pour résultat 5
M [3 , 1] a pour résultat 12
M [2 , 4] a pour résultat 3
M [4 , 2] n'est pas définie
M [1 , 5] n'est pas définie

Remarques :
- Les cases d'un tableau (éléments) sont numérotées à partir de 1 ;
- Lors de la déclaration du tableau, on doit préciser la plus grande valeur de l'indice qui est le nombre maximal d'éléments du tableau ;
- Tous les éléments d'un tableau ont le même type.

IV.5- Algorithmes sur les tableaux :

IV.5.1- Déclaration d'un tableau :

La façon la plus simple de déclarer un tableau dans un algorithme (ou programme) est de le considérer comme une variable :

VAR nom_tableau : **TABLEAU** [1...n] **DE** type_elt

Ou en Pascal :

VAR nom_tableau : **ARRAY** [1...n] **OF** type_elt ;

Où :
nom_tableau : est le nom (identificateur) du tableau,
n : est une constante précisant le nombre d'éléments du tableau,
type_elt : est le type des éléments du tableau (entier, réel, …).

Exemple :

VAR notes_inf : **TABLEAU** [1..500] **DE** réel

Dans le cas d'une ***matrice***, on doit préciser le nombre de lignes et le nombre de colonnes de celle-ci.

Exemple :

VAR notes : **TABLEAU** [1..500, 1..8] **DE** réel

Remarque :
Les exemples précédents deviennent en Pascal :

VAR notes_inf : **ARRAY** [1..500] **OF** real ;
VAR notes : **ARRAY** [1..500, 1..8] **OF** real ;

IV.5.2- Lecture / Ecriture d'un tableau :

La déclaration d'un tableau ne fait que réserver de la place dans la mémoire de l'ordinateur.
Donc, avant tout traitement il faut lire (remplir) le tableau en affectant des valeurs à ses éléments. Cette opération de lecture est réalisée par :
- une boucle dans le cas d'un tableau à une dimension

Exemple :

POUR i allant de 1 **A** 350 **FAIRE**
LIRE (notes_inf [i])

Ou en Pascal :

FOR i := 1 **TO 3**50 **DO**
READ (notes_inf [i]) ;

- deux boucles imbriquées dans le cas d'une matrice

Exemple :

POUR i allant DE 1 **A** 500 **FAIRE**
POUR j allant DE 1 **A** 8 **FAIRE**
LIRE (notes [i, j])

Ou en Pascal :

FOR i := 1 **TO** 350 **DO**
FOR j := 1 **TO** 8 **DO**
READ (notes [i, j]) ;

Remarque:

Pour écrire (afficher) un tableau, on procède de la même façon. Il suffit de remplacer l'instruction de lecture par l'instruction d'écriture.

Exercice :

Ecrire l'algorithme qui permet de lire (où remplir) un tableau de N éléments puis d'afficher son contenu.

Solution :

```
Var T:tableau [1..100] des entiers;
    I,n :entier ;
Début
 Ecrire (' donnez le nombre d''element du tableau') ;
 Lire (n) ;
Ecrire (' introduisez', n ,'valeurs entières') ;
Pour i allant de 1 a n faire
    Lire (T[i] ) ;
Ecrire (' le contenu du tableau T est :') ;
Pour i allant de 1 a n faire
    Ecrire (T[i] ) ;
Fin.
```

La solution en Pascal :

```
Program Lecture_Ecriture ;
Uses wincrt ;
Var t : array [1.. 100] of Integer ;
    i, n : integer;
Begin
Writeln ('Nombre d''éléments de t? ') ;
Read (n) ;
Writeln ('Introduisez ', n, ' valeurs entières') ;
For i := 1 to n Do
    Read (t[i]) ;
Writeln ('Voici le contenu de t :') ;
```

```
For i := 1 to n Do
        Writeln (t[i] : 6) ;
End.
```

IV.5.3- Algorithmes de parcours d'un tableau :

Ils consistent à rechercher une ***valeur donnée*** ou une valeur particulière (minimum, maximum, …) dans un tableau.

Exercice :
Ecrire l'algorithme puis le programme qui recherche le maximum d'un tableau d'entiers de n éléments.
Solution :

```
var t: tableau[1..100] entier;
i,n,max:entier;

Début
Ecrire(' introduisdez',n'  valeurs entiéres');
Lire(n);
Pour i allant de 1 a 100 faire
        lire(t[i]);
max  ← t[1];
Pour i allant de 2 a 100 faire
        si t[i]>max alors   max ← t[i];
Ecrire('le maximum est :',  max);
Fin.
```

La solution en Pascal :

```
Program Rech_Max ;
Uses wincrt ;
Var tab : array [1.. 100] of Integer ;
i, n, max : integer;

Begin
Writeln ('Valeur de n (n <= 100) ? ') ;
Read (n) ;
Writeln ('Introduisez ', n, ' valeurs entières') ;
For i := 1 to n Do
Read (tab[i]) ;
max := tab[1];
For i := 2 to n Do
If tab[i] > max Then
max := tab [i] ;
Writeln ('Le maximum est : ', max);
End.
```

Exercice :
Ecrire le programme qui recherche une valeur donnée val dans un tableau d'entiers de n éléments.

Solution :

```
Var t : tableau [1.. 100] d'entiers;
        I,n,val:entier;
Début
Ecrire (' 'Introduisez ', n, ' valeurs entières');
Lire(n);
Pour i allant de n a 100 faire
        Lire (t[i]);
Ecrire (' donnez la valeur à rechercher ');
        Lire (val);

i ← 1 ;
Tant que (i<=100 et t[i]<> val ) faire
i ← i + 1;
Si t[i] = val alors écrire (val, à la position de ';i)
        Sinon écrire ('val,' ne figure pas dans le tableau');
Fin.
```

La solution en Pascal :

```
Program Rech_Val ;
Uses wincrt ;
Var tab : array [1.. 100] of Integer ;
i, n, val : integer;
Begin
Writeln ('Introduisez ', n, ' valeurs entières') ;
For i := 1 to n Do
Read (tab[i]) ;
Writeln ('donnez la Valeur à rechercher ? ') ;
Read (val) ;
i := 1 ;
While (i <= n) and tab[ i ] <> val Do
i := i + 1;
If tab[ i ] = val Then
Writeln (val, ' a pour position ', i)
ElseWriteln (val, ' ne figure pas dans le tableau tab') ;
End.
```

IV.5.4- Algorithme de tri d'un tableau :

Le tri d'un tableau permet de faciliter les recherches de valeurs dans celui-ci. Il existe plusieurs algorithmes dont le tri par bulles qui est sans doute le plus simple et le plus compact.

1. Principe :

- Par une série de comparaisons et permutations, ramener le maximum (ou le minimum) à la dernière position du tableau t.
- Répéter cette opération sur le sous tableau t[1.. n-1], c'est-à-dire ne pas considérer le dernier élément qui est déjà trié.

- Refaites l'opération sur le sous tableau t[1.. n-2] (le dernier et l'avant dernier éléments sont triés),
- Répéter la même opération sur les sous-tableaux t[1.. n-3], t[1.. n-4], …, t[1.. 2].

2. Algorithme:

```
Var tab : tableau [1.. 100] d'entiers ;
          i, j, n, z : entier;
Début
{Lecture de n et t}
Lire(n);
 Pour i allant de 1 a n faire
        Lire(t[i]);

Pour i allant de 1 a  n-1 faire
Pour j allant de 1 a  n-i faire
si t[j] > t[j+1] alors
        Début
        z ← t[j] ;
        t[j] ←t[j+1] ;
        t[j+1] ←  z ;
        End ;
{Ecriture du tableau}
Pour i allant de 1 a n faire
        Ecrire (t[i]);

End.
```

3. Programme

```
Program Tri_Bulles ;
Uses wincrt ;
Var tab : array [1.. 100] of Integer ;
i, j, n, z : integer;
Begin
{Lecture de n et t}
For i := 1 to n-1 Do
For j := 1 to n-i Do
If t[j] > t[j+1] Then
Begin
z := t[j] ;
t[j] := t[j+1] ;
t[j+1] := z ;
End ;
{Ecriture du tableau}
End.
```

Chapitre V

Les sous-programmes (Procédures et fonctions)

V.1- Introductions

Soit à écrire le programme qui calcule le nombre de combinaisons de n objets pris p à p :

$$C_n^p = \frac{n!}{p!\,(n-p)!}$$

On peut l'écrire ainsi :

```
Program cnp ;
Uses wincrt ;
Var n, p, r1, r2, r3, i : Integer ;
Begin
Writeln ('Valeurs de n et p ? ') ;
Read(n, p) ;
r1 := 1 ;
For i := 2 to n do
r1 := r1 * i ;
r2 := 1 ;
For i := 2 to p do
r2 := r2 * i ;
r3 := 1 ;
For i := 2 to n-p do
r3 := r3 * i ;
Writeln ('Résultat = ', r1 DIV (r2 * r3)) ;
End.
```

Remarque:

Le traitement "Calcul de la factorielle d'un nombre est répété 3 fois. L'idée est d'écrire un "sous-programme" spécialisé dans le calcul de la factorielle d'un nombre puis de l'appeler chaque fois que c'est nécessaire dans le "programme principal" (programme calculant C_n^p).

Le sous-programme qui doit être paramétrable pour qu'il puisse calculer la factorielle de n'importe quel nombre sera appelé en Pascal : une procédure.

V.2- Les procédures

V.2.1- Structure d'une procédure :

Une procédure a la même structure qu'un programme :

Procédure nom (paramètres) ;

} Déclaration des variables locales de la procédure

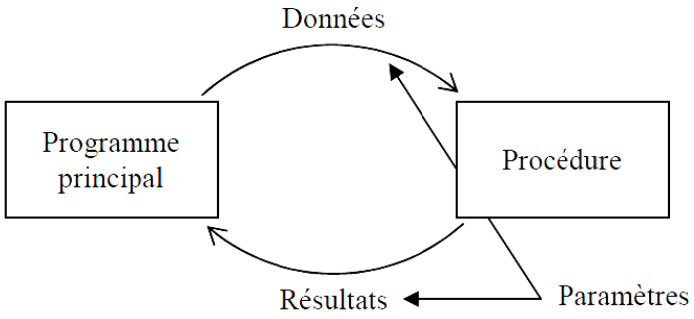
Begin

} Traitements

End ;

Les paramètres sont :

- La (les) donnée(s) transmise(s) par le programme principal,
- Le(s) renvoyé(s) par la procédure.



V.2.2- Les paramètres formels :

La procédure calculant la factorielle d'un nombre peut être définie comme suit :

Procedure fact (x: entier; var y: entier);
Var i: entier ;
Début
y ⟵ 1;
pour i ⟵ 2 jusqu'à x faire
y ⟵ y*i;
fin;
On pascal :
Procedure fact (x : Integer ; Var y : Integer) ;
Var i: Integer;
Begin
y := 1;

```
For i := 2 to x Do
y := y * i ;
End;
```

Remarques :
- La procédure fact calcule : y := x ! (x est la donnée, y est le résultat) ;
- Les paramètres x et y utilisés pour définir la procédure sont appelés :
paramètres formels et sont suivis de leur type (Integer dans notre cas) ;
- Les paramètres résultats (y dans notre cas) sont précédés du mot-clé Var.

V.2.3- Les paramètres réels :

Une fois la procédure *fact* définie, le programme principal peut l'appeler en lui donnant des valeurs qui vont remplacer les paramètres formels x et y.
Ces valeurs sont appelées : paramètres réels (ou paramètre effectifs).

Exemple :
Pour calculer R1 := n !, l'appel à la procédure fact s'écrit : fact (n, R1) ;
Le programme cnp précédant devient :

```
Program cnp ;
Uses wincrt ;
Var n, p, r1, r2, r3 : Integer ;
Procedure fact (x : Integer ; Var y : Integer) ;
Var i: Integer;
Begin
y := 1;
For i := 2 to x Do
y := y * i ;
End;
Begin
Writeln ('Valeurs de n et p ? ') ;
Read(n, p) ;
fact (n, r1) ;
fact (P, r2) ;
fact (n-p, r3) ;
Writeln ('Résultat = ', r1 DIV (r2 * r3) ;
End.
```

Remarque :
La procédure est définie (déclarée) dans la partie "déclarations" du programme au même titre que les variables.

V.3- Les fonctions :

Une procédure permet de retourner plusieurs résultat, pas forcément de même type. Toute procédure calculant un seul résultat peut être définie (écrite) comme : une fonction (qui ne peut retourner qu'un seul résultat.

Une fonction est un ensemble d'instructions qui forment un sous programme, les fonctions en algorithmique (programmation) ressemblent à celles de mathématique, Chaque fois qu'on l'appelle elle renvoie au programme appelant une valeur qui est le résultat du traitement effectués par les instructions de la fonction.

Structure d'une fonction :

Une fonction a la même structure qu'une procédure :
Function nom (paramètres) : type du résultat ;
Déclaration des variables locales de la fonction
Begin
Traitements
End ;

Remarque :
Les paramètres de la fonction sont tous des données. Le résultat est renvoyé dans le nom de la fonction, qui est lui-même une variable de retour.

Exemples :
a- La fonction calculant la factorielle d'un nombre entier.

```
Function facto (x : Integer) : Integer ;
Var i, r : Integer ;
Begin
r := 1 ;
For i := 2 to x Do
        r := r * i ;
        facto := r ;
End ;
```

b- Fonction nous disant si un nombre entier donné est de 5 ou non.

```
Function mult5 (x : Integer) : Boolean ;
Begin
r := 1 ;
If x Mod 5 = 0 Then
        mult5 := True
Else
        mult5 := False ;
End ;
```

Le programme cnp peut prendre la forme :

```
Program cnp ;
Uses wincrt ;
Var n, p, res : Integer ;
Function facto (x : Integer) : Integer ;
Var i, r : Integer ;
Begin
r := 1 ;
For i := 2 to x Do
        r := r * i ;
        facto := r ;
```

```
    End ;
Begin
Writeln ('Valeurs de n et p ? ') ;
Read(n, p) ;
res := fact(n) Div (facto(p) * facto(n-p))
Writeln ('Résultat = ', res) ;
End.
```

V.4- Les paramètre "valeur" et "variable" :

Un paramètre d'une procédure sera transmis comme :
- une valeur lorsqu'on veut interdire à la procédure de modifier sa valeur,
- une variable lorsqu'on veut autoriser la procédure à modifier sa valeur.

Ainsi, dans la procédure fact dont l'en-tête est :

Procedure fact (x : integer ; **Var** y : Integer) ;
x est un paramètre "valeur" alors que y est un paramètre "variable"

Remarque :
En général, les données sont transmises comme des valeurs alors que les résultats sont transmis comme des variables.

Exercice :
Un nombre entier est parfait s'il est égal à la somme de ses diviseurs propres (diviseurs autres que le nombre lui-même).

Exemples :
- 6 = 1 + 2 + 3
- 28 = 1 + 2 + 4 + 7 + 14
- …

Ecrire le programme qui nous dit si un entier n est parfait ou non, en utilisant une procédure (ou fonction) calculant la somme des diviseurs propres d'un nombre.

Exercice :
Deux entiers A et B sont amis si la somme des diviseurs propres de A est égale à B et a somme des diviseurs propres de B est égale à A.

Exemple : 220 et 284
Ecrire le programme qui nous dit si 2 entiers A et B sont amis ou non, en utilisant la procédure (ou fonction) définie dans l'exercice V.1.

Chapitre VI

Les enregistrements et les fichiers

VI.I- Les enregistrements :

VI.I.1- Définition 1 :

Un enregistrement est une structure (complexe) de données permettant de représenter un ensemble de données (dites champs) hétérogènes, c'est à dire de types différents.

On peut considérer un enregistrement comme étant un tableau à une dimension, où chaque case (composante) du tableau représente un champ dans l'enregistrement. La différence réside dans :

– Une case de tableau est accessible à travers un indice entier, par contre un champ est repéré par un nom (identificateur) ;

– Les cases du tableau sont toutes du même type (structure homogène), par contre, les champs de l'enregistrement ne sont pas obligatoirement du même type (structure hétérogène).

Définition 2 :

Un enregistrement est un ensemble quantique d'éléments n'ayant pas nécessairement le même type contrairement aux tableaux où tous les éléments doivent être de même type. On utilise la notion d'enregistrement lorsqu'on désire stocker plusieurs informations relatives à un même objet.

Exemple 1:

Pour représenter les informations liées à un produit quelconque dans une même structure, on peut utiliser un enregistrement constitué de champs suivants :

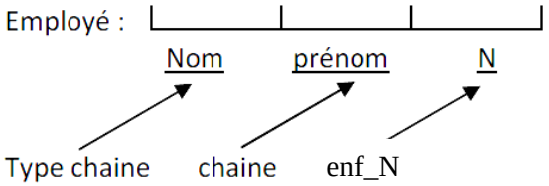
- La désignation du produit
- La référence du produit
- Sa quantité en stock
- Son prix unitaire
- Etc…

Les informations précédentes (les champs de l'enregistrement) sont de types différents, elles ne peuvent être décrites par un tableau (dont les cases sont du même type). Les types de ces informations sont respectivement :

- Chaîne de caractères
- Chaîne de caractères
- Entier
- Réel

Supposant que l'on veuille stocker les informations suivantes : le nom, le prénom, le nombre d'enfants relatives à un employé. L'algorithme et programme en pascal se traduit comme suit :

Algorithme :
Type enf_N=1..10;
Personne = enregistrement ;
Nom: chaine de caractère;
Prénom: chaine de caractère;
N: enf_N;
Fin;
Var Employé : Personne ;
Programme en pascal :
Type enf_N = 1..10 ;
Personne = **Record**
Nom : string ;
Prénom : string ;
N : enf_N;
End;
Var Employé : Personne ;



Remarque :

La déclaration *Personne = Record…* sert à définir un type nommé Personne composé de trois éléments : le nom de type chaine, le prénom de type chaine, et le nombre d'enfants de type enf_N. Cette définition de type n'implique aucunement une réservation de cellule en mémoire centrale de l'ordinateur.
La déclaration *var Employé : Personne* sert à déclarer une variable nommée *Employé* de type *Personne*, cette instruction implique une réservation en mémoire centrale d'un espace mémoire nommé *Employé* est organisé de la manière représentée sur la figure ci-dessus. Les éléments : nom, prénom et N du type enregistrement *Personne* sont appelés des champs.

VI.I.2- Référencer un champ d'une variable enregistrement :

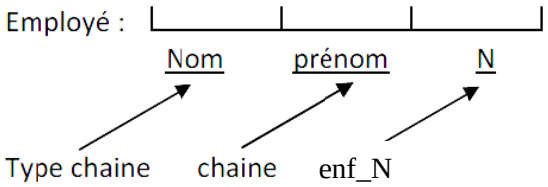
Pour référencer un champ particulier X d'une variable enregistrement V, on utilise la relation : V.X

Exemples :
1- Supposant que l'on veuille stocker dans la variable *Employé* les informations suivantes :
nnn ⟶ Nom
ppp ⟶ Prénom
3 ⟶ enf_N

Voici comment faire :
Begin
.
.
Employé :='nnn' ;
Employé :='nnn' ;
Employé :='nnn' ;
.
End.
2- L'instruction *writeln (Employé.Nom)* permet d'afficher le contenu du champ *Nom* de la variable *Emplyé*.

VI.I.3- Syntaxe d'un enregistrement :

Type nom *du nouveau type* = **Record**
Liste 1 de champs ;
Liste 2 de champs ;
.
.
Liste *n* de champs ;
End ;



VI.I.4- Enregistrement dans un enregistrement :

Il est possible en Pascal de définir un type enregistrement à l'intérieur d'un enregistrement.
Exemple :
Supposant que l'on veuille stocker la date d'embauche et la date de naissance dans la variable *Employé*, voici le résultat :
Type chaine = packed array [1..10] of char;
Enf_N = 1..10 ;
Date = **Record**
Jour : 1..31 ;
Mois : 1..12 ;
Année : 2000.. 2100 ;
End;
Personne = **Record**
Nom, Prénom : chaine ;
N : Enf_N ;
Dnais, Drecr : Date ;
End;

VI.I.5- L'instruction "WITH" :

L'instruction ***WITH*** est une instruction de factorisation. Considérant la variable *'Employé'* de type *'Personne'* où *Personne* est définie comme suit :
Type chaine = packed array [1..10] of char;
Enf_N = 1..10 ;

Personne = **Record**
Nom, Prénom : chaine ;
N : Enf_N ;
End;
Var Employé : Personne ;
Considérons la séquence d'instruction suivante qui consiste à afficher le contenu nde chaque champ de la variable '*Employé'*.
Writeln (Employé.Nom) ;
Writeln (Employé.Prénom) ;
Writeln (Employé.N) ;
Nous remarquons que dans chaque instruction d'affichage le nom 'Employé' figure.
Il existe une écriture plus simplifiée mettant en facteur le nom de la variable '*Employé*', on utilise l'instruction *WITH*.

```
Writeln (Employé.Nom) ;
Writeln (Employé.Prénom) ;   ⟺   WITH Employé DO
Writeln (Employé.N) ;                Begin
                                         Writeln (Nom) ;
                                         Writeln (Prénom) ;
                                         Writeln (N) ;
                                     End;
```

VI.I.5.1- Syntaxe de l'instruction WITH :

WITH Nom_variable **DO**
Instruction / **Begin**
Séquence d'instructions
End ;

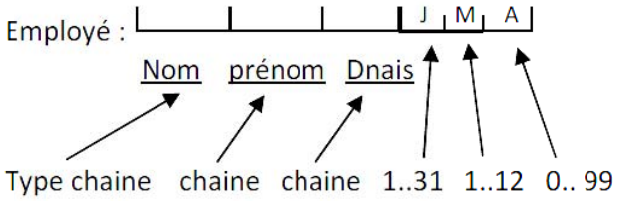
La partie située après le mot clé *DO* s'appelle le corps de l'instruction *WITH*. Lorsque le corps de l'instruction *WITH* est une séquence d'instructions, on doit délimiter cette dernière par les deux mots clés *Begin* et *End*.
Pour faire référence à un champ particulier de nom var dans le corps *WITH*, on a qu'à spécifier le nom du champ en question uniquement.

VI.I.5.2- L'instruction "WITH" dans une instruction "WITH" :

Supposant qu'on plus des informations à savoir le *Nom*, le *Prénom*, et le *nombre d'enfants*, on a besoin de la date de naissance de l'employé.

Exercice
Ecrire le programme qui permet d'afficher les informations suivantes :
'NNN'------ Nom
'PPP' ------ Prénom
5 ---------- Nombre d'enfants
05/11/90 -- Date de naissance
Program emp ;
Type chaine = packed array [1..10] of char;
Date = **Record**

J : 1..31 ;
M : 1..12 ;
A : 0.. 99 ;
End;
Personne = **Record**
Nom, Prénom : chaine ;
N : 0.. 10 ;
Dnais : Date ;
End;
Var Employé : Personne ;
WITH Employé **DO**
Begin
Writeln (Nom) ;
Writeln (Prénom) ;
Writeln (N) ;
End;



Begin

/* Remplissage de la variable Employé */

Employé.Nom := 'NNN' ;
Employé.Prénom := 'PPP' ;
Employé.N := 5 ;
Employé.Dnais.J := 5 ;
Employé.Dnais.M := 11 ;
Employé.Dnais.A := 90 ;

/* Affichage */

Writeln (Employé.Nom) ;
Writeln (Employé.Prénom) ;
Writeln (Employé.N) ;
Writeln (Employé.Dnais.J) ;
Writeln (Employé. Dnais.M) ;
Writeln (Employé. Dnais.A) ;
End.

```
/* Remplissage de la variable Employé */
WITH Employé DO
   Begin
     Nom := 'NNN' ;
     Prénom := 'PPP' ;
     N := 5 ;
     WITH Dnais DO
        Begin
          J := 5 ;
          M := 11 ;
          A := 90 ;
        End ;
   End;

/* Affichage */
WITH Employé DO
   Begin
     Writeln (Nom) ;
     Writeln (Prénom) ;
     Writeln (N) ;
     WITH Dnais DO
       Writeln (J, M, A) ;
   End;
```

VI.I.6- Les variantes dans les enregistrements :

Certains champs d'un enregistrement peuvent être différents suivant la valeur d'un des champs en utilisant la structure *Case... Of* dans la déclaration de l'enregistrement.

Exemple :

```
Type Statut = (Célibataire, Marié, Divorcé, Veuf);
     Personne = Record
                  Nom : string[20];          } ← Partie figée
                  Case situation : Statut Of
                     Célibataire : ();
                     Marié : (enfants : 0..10);
                     Divorcé, Veuf : (enfants : 0..10; remarié : boolean)   } Partie non figée
                  End ;
                End ;
```

Dans cette exemple, suivant la situation de la personne, si c'est marié alors l'enregistrement *Personne* aura le champ *enfants*, si c'est divorcé ou veuf alors en plus du champ *enfants* on aura un autre champ booléen qui est *remarié*.

Les champs situés dans la *partie figée* feront partie de la structure de l'enregistrement à tout moment. Les champs situés dans la *partie non figée* sont soumis à une condition, ils feront partie de la structure de l'enregistrement selon cette condition.

VI.II- Les fichiers

VI.II.1- Définition

Des fois, il est nécessaire de conserver certaines données après la fin du programme, ceci pour une utilisation future, les fichiers ont été conçus pour ces fins.
Un **fichier** est une structure de données, toutes de même type. L'accès à un élément (une donnée) du fichier peut se faire :

- De manière séquentielle : c'est-à-dire en parcourant le fichier élément par élément depuis le début jusqu'à l'élément choisi ;
- De manière direct : en donnant la position de l'élément ;

Les fichiers sont conservés en **mémoire secondaire** (disques, flash disk, ...), les données qui les constituent reste toujours tant que cette mémoire secondaire n'est pas formatée ou endommagée. Chaque fichier est désigné par un nom et possède des attributs tels que date de création, taille, icône...

Il existe deux types de fichiers :

1. **Les fichiers binaires :** Contenant du code binaire représentant chaque élément. Ces fichiers ne peuvent être manipulés que par des programmes !
2. **Les fichiers texte :** appelés aussi imprimables, contenant des caractères et susceptibles d'être lus, éditées, imprimés....

VI.II.2- Operations sur les fichiers

VI.II.2.1- Création :

Pour créer un fichier il faut d'abord définir sa structure : son nom logique et physique..

Exemple :

Type info_etud = **structure**
Mat : string[10] ;
Nom : string [15] ;
Prenom : string[15] ;
Note: réel ;
Fin ;

Le fichier sera reconnu par l'algorithme (programme) par un nom dit logique. Le nom logique est attribué au fichier dans la partie déclaration au moyen d'une variable de type fichier.

Exemple :

Var étudiant = **fichier** d'info_etud;

Un fichier n'est reconnu par le SGF (système de gestion des fichiers) que par un nom physique, et par conséquent, un nom physique est associé au nom logique du fichier. L'affectation (ou l'association) du nom logique au nom physique est réalisée par la procédure prédéfinie **ASSIGN** dont la syntaxe est :

ASSIGN (nom logique, nom physique) ;

Exemple :

ASSIGN (étudiant, 'c:\etud.txt') ;

Cela veut dire que *étudiant* est le fichier etud.txt qui est stocké dans la racine de la partition C :

VI.II.2.2- Ouverture d'un fichier

Pour pouvoir agir sur le contenu d'un fichier existant il faut l'ouvrir, il y a deux modes d'ouvertures de fichiers :

a) Ouverture en mode écriture : Cette opération revient à créer un nouveau fichier et à le préparer pour l'écriture, cela est effectué par la procédure prédéfinie **Rewrite**

Exemple :

Rewrite (étudiant) ;

Remarque :

Si cette procédure est appliquée sur un fichier déjà existant, son contenu sera perdu.

b) Ouverture en mode lecture : C'est une opération qui permet d'ouvrir un fichier stocké sur disque pour la lecture et la modification grâce à la procédure **RESET**.

Exemple :

Reset (étudiant) ;

VI.II.2.3- Fermeture d'un fichier

Une fois l'utilisation d'un fichier est terminée, il faut le fermer par la procédure **CLOSE**.

Exemple :
Close (étudiant) ;

VI.II.2.4- Lecture et écriture d'un enregistrement dans un fichier

La lecture d'un enregistrement à partir d'un fichier existant s'effectue au moyen de la procédure **READ**.[5]
Read (étudiant, enregistrement) ;
L'écriture d'un enregistrement dans un fichier s'effectue au moyen de la procédure **Write**.
Write (étudiant, enregistrement) ;

VI.II.2.5- Autres opérations

a) suppression d'un fichier : pour supprimer un fichier, on utilise la procédure **ERASE**
Erase (fichier1) ;
b) Changement du nom physique : pour changer le nom physique d'un fichier on utilise la procédure **RENAME.**
Rename (fichier1, nom physique2) ;
c) Consultation d'un fichier : Les enregistrements (contenu) d'un fichier peuvent être lus de deux manières :

- **Accès séquentiel :** parcourir le fichier du premier élément au dernier (fin du fichier). La fin du fichier est détectée par la fonction booléenne **EOF** (End Of File) **,** qui retourne la valeur vrai si c'est la fin du fichier ou faux sinon.

Exemple : si EOF (fichier1) **alors**

- **Accès direct :** il est possible d'accéder directement à un élément du fichier en précisant son numéro d'ordre. Ceci est réalisé par la procédure SEEK.[5]

Exemple : SEEK (nom logique, position) ;
Exercice VI.1 :
Ecrire l'algorithme qui permet de créer un fichier étudiant contenant : le matricule, le nom, le prénom, niveau d'étude, l'âge et la moyenne.
Solution

```
program exo1 ;
type info_etud = record
                    mat : string[8] ;
                    nom : string[15] ;
                    prenom : string[15] ;
                    niv_etud : integer ;
                    age : integer ;
                    moyenne : real ;
                 end ;
var etudiant : file of info_etud ;
   enreg : info_etud ;
begin
    assign (etudiant, 'c:\etudiant.txt') ;
    rewrite (etudiant) ;
    close (etudiant) ;
end.
```

Exercice VI.2 :

Ecrire l'algorithme qui permet d'ouvrir le fichier précédent (étudiant) et le remplir.

Solution

```
program exo2 ;
type info_etud = record
mat : string[8];
nom : string[15];
prenom : string[15];
niv_etud : integer;
age : integer;
moyenne : real;
end ;
var etud : file of info_etud ;
enreg : info_etud ;
i : integer ;
car : char ;
begin
assign (etud,'c:\etudiant.txt') ;
reset (etud) ;
car:= 'o' ;
while car = 'o' do
begin
writeln ('saisir un enregistrement') ;
write ('?maticule=') ; readln (enreg.mat) ;
write ('?nom=') ; readln (enreg.nom);
write ('?prenom=') ; readln (enreg.prenom) ;
write ('?niveau=');readln (enreg.niv_etud) ;
```

```
write ('?moyenne=') ; readln (enreg.moyenne) ;
write (etud,enreg) ;
writeln ('Si vous voulez continuer tapez "O"…') ;
readln (car) ;
end ;
close (etud) ;
end.
```

Sujets d'examens

Sujet 1

Exercice 1 (6 points) : Retrouvez les valeurs des expressions ci-dessous en indiquant l'ordre et le résultat de chaque opération :

1) 5 + 2 * 3 – 4
2) 3 * 2 – 5 mod 2
3) 8 div (5 mod 2 + 1)
4) 5 < 2 + 2 * 3

Exemple : pour l'expression : 4 + 3 * 2, la réponse devrait être présentée comme suit :

a) 3 * 2 = 6
b) 4 + 6 = 10

Exercice 2 (6 points) : x, y et s étant des variables entières, on considère la séquence d'instructions suivante:

```
If x > 0
Then s := 1
Else If y > 0 Then s := 1
Else s := 0 ;
```

Questions :

1) Complétez la table de vérité ci-dessous en indiquant la valeur de s dans chaque cas.

x > 0	y > 0	S
Faux	Faux	
Faux	Vrai	
Vrai	Faux	
Vrai	Vrai	

2) Proposez une séquence d'instructions qui donnerait le même résultat mais dans laquelle on ne répèterait pas l'instruction : s := 1.

Exercice 3 (8 points) : Soit l'algorithme :
Var a , b , r : **Entier**
Début
Ecrire (' Valeurs de a et b ? ')
Lire(a , b)
TantQue (b ≠ 0) **Faire**
r ← a mod b
a ← b
b ← r
FinTantQue
Ecrire ('Resultat =' , a)
Fin
Questions :
1) Faire une trace complète de cet algorithme pour : a = 32 et b = 12.
2) Ecrire le programme PASCAL équivalent à cet algorithme.
3) Réécrire cet algorithme en utilisant la boucle **Répéter** à la place de la boucle **TantQue**.

Corrigé-sujet 1

Exercice1:

5 + 2 * 3 – 4	3 * 2 – 5 mod 2	8 div (5 mod 2 + 1)	5 < 2 + 2 * 3
a) 2 * 3 = 6 b) 5 + 6 = 11 c) 11 – 4 = 7	a) 3 * 2 = 6 b) 5 mod 2 = 1 c) 6 – 1 = 5	a) 5 mod 2 = 1 b) 1 + 1 = 2 c) 8 div 2 = 4	a) 2 * 3 = 6 b) 2 + 6 = 8 c) 5 < 8 = true

Exercice 2:
1)
If x > 0
Then s := 1
Else If y > 0 **Then** s := 1
Else s := 0 ;

x > 0	y > 0	s
Faux	Faux	0
Faux	Vrai	1
Vrai	Faux	1
Vrai	Vrai	1

2)

```
If (x>0) or (y>0)
Then  s := 1
Else  s := 0 ;
```

ou bien

```
If (x<=0) and (y<=0)
Then s := 0
Else s := 1 ;
```

Exercice 3:

1) trace de l'algorithme pour : a=32 ; b=12

Algorithme

```
Algorithme Emd1
Var a , b , r : Entier
Début
   Ecrire ( ' Valeurs de a et b ? ' )
   Lire(a , b)
   TantQue ( b ≠ 0 ) Faire
      r ← a mod b
      a ← b
      b ← r
   FinTantQue
   Ecrire ('Resultat =' , a )
Fin
```

trace ⟹

a	b	r	b ≠ 0
0	0	0	faux
32	12		vrai
		8	
12			
	8		
		4	
8			
	4		
		0	
4			
	0		Faux

2) Traduction de l'algorithme en un programme Pascal :

```
Program Emd1 ;
Uses WinCrt ;
Var a , b , r : Integer ;
Begin
Writeln ( ' Valeurs de a et b ? ' ) ;
Readln (a , b) ;
While ( b < > 0 ) Do
Begin
r := a mod b ;
a := b ;
b := r ;
End ;
Write ('Resultat =' , a ) ;
End.
```

3) Le même algorithme en utilisant la boucle Répéter :

Algorithme Emd1

Var a , b , r : **Entier**

Début
Ecrire (' Valeurs de a et b ? ')
Lire(a , b)
Répéter
r ← a mod b
a ← b
b ← r
Jusqu'à b=0
Ecrire ('Resultat =' , a)
Fin

Sujet 2

Exercice 1
Soit le programme suivant :

```
Programme exo1 ;
Var x, y, z, i : integer ;
Begin
    Write ('Introduire un entier SVP : ') ;
    Read (a) ;
    b := 0 ;
    For i := 2 to a do
        b := b * i ;
    Write (b) ;
End.
```

1- Déroulez ce programme pour **a = 5**.
2- Cherchez l'erreur (ou les erreurs) dans le programme et corrigez-les.
3- Déroulez ce programme, maintenant, pour **a = 4**.
4- Que fait ce programme ?
5- Réécrire ce programme en utilisant la boucle "*While … Do*" au lieu de la boucle "*For … Do*".

Exercice 2
Ecrire un algorithme (ou un programme) qui demande à l'utilisateur sa note de l'examen et sa note de TD pour un module donnée et qui lui affiche en sortie sa moyenne et une mention ***bien*** si sa moyenne est supérieure ou égale à **15/20**.
NB. moyenne = (note_EMD x 2 + note_TD)/3

Solution :

Programme exo1 ;
Var x, y, z, i : integer ;
Begin

Write ('Introduire un entier SVP : ') ;
Read (a) ;
b := 0 ;
For i := 2 to a **do**
b := b * i ;
Write (b) ;
End.

Corrigé-sujet 2

Exercice 1

Soit le programme suivant :

```
Programme exo1 ;
Var x, y, z, i : integer ;
Begin
    Write ('Introduire un entier SVP : ') ;
    Read (a) ;
    b := 0 ;
    For i := 2 to a do
        b := b * i ;
    Write (b) ;
End.
```

1- Pour **a = 5**.

i	b
	0
2	0 x 2 = 0
3	0 x 3 = 0
4	0 x 4 = 0
5	0 x 5 = **0**

La valeur de sortie pour b est égale à **0**

2- Cherchez l'erreur (ou les erreurs) dans le programme et corrigez-les.

Erreur	corrigé
- **Var** x, y, z, i : integer ; - b := 0 ;	- **Var** a, b, i : integer ; - b := 1 ;

3- Pour **a = 4**.

I	b
	1
2	1 x 2 = 2
3	2 x 3 = 6
4	6 x 4 = **24**

La valeur de sortie pour b est égale à **24**

```
Programme exo1 ;
Var x, y, z, i : integer ;
Begin
Write ('Introduire un entier SVP : ') ;
Read (a) ;
b := 0 ;
For i := 2 to a do
b := b * i ;
Write (b) ;
End.
```

4- Ce programme calcule le factoriel d'un nombre entier donné introduit au clavier. On introduit en entrée l'entier *a* et le programme calcule et affiche le *a* factoriel (*a!*).

5- Réécriture de ce programme en utilisant la boucle "*While … Do*".

```
Programme exo1 ;
Var a, b, i : integer ;
Begin
    Write ('Introduire un entier SVP : ') ;
    Read (a) ;
    b := 1 ;
    i := 2;
    While i <= a do
      Begin
        b := b * i ;
        i := i + 1;
      end;
    Write (b) ;
End.
```

Exercice 2

Ecrire un algorithme (ou un programme) qui demande à l'utilisateur sa note de l'examen et sa note de TD pour un module donnée et qui lui affiche en sortie sa moyenne et une mention ***Bravo*** si sa moyenne est supérieure ou égale à **15/20**.

NB. moyenne = (note EMD x 2 + note TD)/3

```
Programme exo2 ;
Var emd, td, moy : real ;
Begin
    Write ('Introduire ta note-EMD STP : ') ;
    Read (emd) ;
    Write ('Introduire ta note-TD STP : ') ;
    Read (td) ;
    moy := (emd x 2 + td)/3;
    Write ('Ta moyenne = ', moy) ;
    If moy >= 15 then
        Write ('Bravo') ;
End.
```

Sujet 3

Soit l'algorithme suivant :

```
Var tab : tableaux [1..5] d'entiers ;
i, permut, a : entier ;
Début
Pour i allant de 1 jusqu'à 5 faire
Lire (tab[i]) ;
Répéter
permut ← 0 ;
Pour i allant de 1 jusqu'à 4 faire
Si tab [i] > tab [i+1] alors
Début
        a ← tab [i];
        tab [i] ← tab [i+1];
        tab [i+1] ← a;
        permut ← 1;
Fin;
Jusqu'à permut = 0;
Pour i allant de 1 jusqu'à 5 faire
Ecrire (tab[i]) ;
Fin.
```

1- Dérouler cet algorithme pour le tableau : tab = (10, 8, 5, 11, 30).
2- Que fait cet algorithme ?

Exercice 2 :
Ecrire l'algorithme (ou programme) qui permet de rechercher le minimum et le maximum dans un tableau de 10 éléments, de calculer et d'afficher la moyenne entre ce minimum et ce maximum.

Exercice 3 :
Soient **note** un tableau qui contient les notes de tous les modules d'un étudiant X, **coef** : le vecteur des coefficients qui correspond à ces notes ; Ecrire l'algorithme (ou programme) qui permet de calculer la moyenne de l'étudiant X.

Corrigé-sujet 3

Exercice 1

Soit l'algorithme suivant :

```
Algorithme EMD1EX01.
Var tab : tableaux [1..5] d'entiers :
    i, permut, a : entier :
Début
    Pour  i allant de 1 jusqu'à 5 faire
        Lire (tab[i]) ;
    Répéter
        permut ← 0 ;
        Pour i allant de 1 jusqu'à 4 faire
            Si tab [i] > tab [i+1] alors
                Début
                    a ← tab [i];
                    tab [i] ← tab [i+1];
                    tab [i+1] ← a;
                    permut ← 1;
                Fin;
    Jusqu'à permut = 0;
    Pour  i allant de 1 jusqu'à 5 faire
        Ecrire (tab[i]) ;
Fin.
```

Réponse :

1- Déroulement de l'algorithme pour :
tab = (10, 8, 5, 11, 30)

i	tab	permut
1 2 3 4	10, 8, 5, 11, 30 8, 10, 5, 11, 30 8, 5, 10, 11, 30 // //	0 1 1
1 2 3 4	5, 8, 10, 11, 30 // // //	0 1
1 2 3 4	**5, 8, 10, 11,** **30** // // //	0

2- Cet algorithme trie un vecteur de 5 éléments par ordre croissant.

Exercice 2

Ecrire l'algorithme (ou programme) qui permet de rechercher le minimum et le maximum dans un tableau de 10 éléments, de calculer et d'afficher la moyenne entre ce minimum et ce maximum.

Solution

Var t : tableau [1..10] d'entiers ;
i, min, max : entier ;
moy : réel ;
Début
Ecrire ('Introduire les éléments du tableau t : ') ;
Pour i allant de 1 jusqu'à 10 **faire**
Lire (t[i]) ;
Min ← t[1] ;
Max← t[1] ;
Pour i allant de 2 jusqu'à 10 **faire**
Début
Si t[i] < min **alors**
min ← t[i] ;
Si t[i]> max **alors** max ← t[i] ;
Fin ;
moy ← (min + max) / 2;
Ecrire ('La moyenne entre le minimum et le maximum du tableau t = ', moy);
Fin.

Exercice 3:
Soient **note** un tableau qui contient les notes de 7 modules d'un étudiant X, **coef** : le vecteur des coefficients qui correspond à ces notes ; Ecrire l'algorithme (ou programme) qui permet de calculer et d'afficher la moyenne de l'étudiant X.

Algorithme :

Var note : tableau [1..7] de réel ;
coef : tableau [1..7] d'entiers;
i, somcoef : entier ;
somprod, moy : réel ;
Début
Ecrire ('Introduire les notes de X : ') ;
Pour i allant de 1 jusqu'à 7 **faire**
Lire (note[i]) ;
Ecrire ('Introduire les coefficients correspondants : ') ;
Pour i allant de 1 jusqu'à 7 **faire**
Lire (coef[i]) ;
somprod ← 0 ;
Pour i allant de 1 jusqu'à 7 **faire**
somprod ← somprod + note[i] x coef[i];
somcoef ← 0 ;
Pour i allant de 1 jusqu'à 7 **faire**
somcoef ← somcoef + coef[i];
moy ← somprod / somcoef ;
Ecrire ('La moyenne de l''étudiant X = ', moy);
Fin.

Références bibliographiques

[1] Harold Abelson, Gerald Jay Sussman, and Julie Sussman. Structure et interprétation des programmes informatiques. InterEditions, 1989.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest et Clifford Stein. *Introduction to Algorithms*. MIT Press et McGraw-Hill edition. ISBN : 978-0-262-03293-3, 2001.

[3] Mc Belaid. *Algorithmique et structures de données*. Edition les pages bleus. ISBN : 978-9961-734-93-2, 2008.

[4] Mc Belaid. *Algorithme et Programmation en Pascal*. Edition les pages bleus. ISBN : 978-9961-734-68-8, 2006.

[5] Jacques Courtin. *Initiation à l'algorithmique et aux structures de données*. Edition DUNOD, 1998.

[6]Michel Divay. *Algorithmes et structures de données génériques*. Edition Dunod, 2004.