Objectif général :

Maîtriser l’algorithmique et les structures algorithmiques et savoir coder un algorithme en un programme Java.

Pré requis : Aucun

Organisation : 30h de cours intégrés et 30 h de travaux pratiques.

Préface

Ce document présente un support de cours et d'exercices pour l'enseignement du module Algorithmique et programmation Java.
Ce support s'adresse principalement aux stagiaires qui vont suivre le programme de pré-qualification pour la préparation aux tests d'accès aux cursus de formation du programme national de certification des compétences en TIC dans son tronc commun.

Le document est structuré en 9 chapitres. Le premier chapitre présente quelques définitions et généralités, les 3 chapitres suivant décrivent les structures de programmation allant des actions simples aux structures itératives et passant par les structures conditionnelles.

Le chapitre 5 traite les tableaux et le chapitre 6 définit la notion de tri des tableaux et présente quelques algorithmes de tri. Au cours du chapitre 7, une étude de la programmation procédurale est présentée. Les notions de l'orientée objet sont décrites au cours du chapitre 8. Le dernier chapitre introduit quelques notions générales sur le langage Java.

Le document est dans sa première version, donc on vous serez reconnaissant si vous nous faites part de vos remarques et suggestions.

Tout en s'excusant de notre part si jamais il y'a eu quelques omissions ou erreurs, on vous souhaite une bonne lecture et n'hésitez pas à nous faire part de vos remarques et suggestions dans l'objectif d'amélioration de ce support.

Sommaire

CHAPITRE I. GENERALITES

Objectif : Connaître le but du cours d'algorithmique

Éléments de contenu :

- Qu'est ce qu'une application informatique ?
- Comment arriver d'un problème réel à un programme pouvant être exécuté par ordinateur
- Liens entre ALGORITHMIQUE et STRUCTURES DE DONNEES

I.1.Intérêt de l'algorithmique

Informatiser une application, facturation de la consommation d'eau, par exemple, c'est faire réaliser par ordinateur, une tâche qui était réalisée par l'Homme.
Pour faire exécuter une tâche par ordinateur, il faut tout d'abord, détailler suffisamment les étapes de résolution du problème, pour qu'elle soit exécutable par l'homme. Ensuite, transférer la résolution en ***une suite d'étapes si élémentaire et simple à exécuter***, pouvant être codée en ***un programme*** dans un langage compréhensible par ordinateur.
Toute ***suite d'étapes si élémentaire et simple à exécuter*** s'appelle un ALGORITHME.

Un ***programme*** c'est un algorithme codé dans un langage compréhensible par ordinateur à l'aide d'un compilateur (traducteur).

I.2.Définitions

L'algorithme *est le résultat d'une démarche logique de résolution d'un problème pour la mise en œuvre pratique sur ordinateur et afin d'obtenir des résultats concrets il faut passer par l'intermédiaire d'un langage de propagation.*

Un algorithme *décrit une succession d'opérations qui, si elles sont fidèlement exécutées, produiront le résultat désiré.*

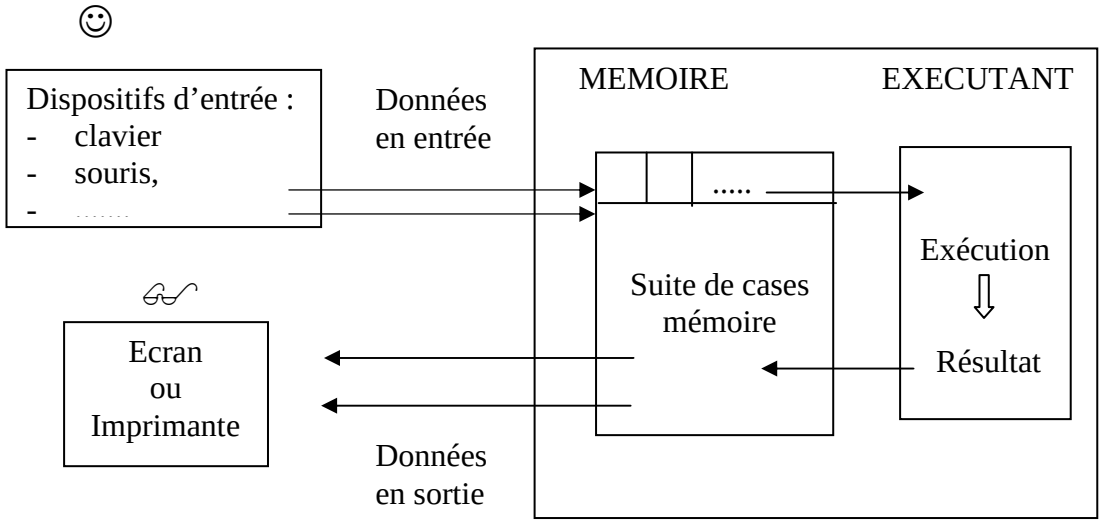
Un algorithme *est une suite d'actions que devra effectuer un automate pour arriver en un temps fini, à un résultat déterminé à partir d'une situation donnée. La suite d'opérations sera composée d'actions élémentaires appelées* ***instructions****.*

Qu'est ce que l'Algorithmique ?
C'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 terme d'actions élémentaires qu'il peut assurer et la logique d'exécution des instructions.

I.3.Les étapes de résolution d'un problème

1. Comprendre l'énoncé du problème
2. Décomposer le problème en sous-problèmes plus simple 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. Elaboration d'un algorithme.

Illustration du fonctionnement d'un ordinateur



On peut dire que la partie EXECUTANT est le problème de l'algorithmique, et la partie MEMOIRE (stockage de donnée) concerne la matière " Structures de données ".

I.4.Structure d'un algorithme

```
ALGORITHME nom_de_l'algorithme
CONSTANTES {Définition des constantes}
TYPES {Définition de types}
VARIABLES   {Déclaration de variables}
DEBUT
     {Suite d'instructions}
FIN
```

Notions de :
- Constante,
- Type,
- Variable.

Exemple 1

ALGORITHME afficher
DEBUT
    Ecrire("La valeur de 3*5 est ", 3*5)
FIN

Cet algorithme permet d'afficher sur l'écran la phrase suivante :
La valeur de 3*5 est 15

Exemple 2

On veut écrire l'algorithme qui permet de saisir 3 notes d'un étudiant dans trois matières, étant donnés les coefficients respectifs 2, 3 et 1.

Résolution

A partir de l'énoncé du problème, nous recherchons la solution par une démarche en 2 phases.

- On doit comprendre comment le résoudre manuellement,
- Définir ce qu'on a besoin comme ***données***, quelles est la démarche à suivre (***formules de calcul)*** pour arriver aux ***résultats.***

Pour notre problème, nous connaissons les coefficients et la formule de calcul ($\sum N_i * C_i / \sum C_i$), nous avons besoins des notes de chaque matière ***dans l'ordre,*** et enfin nous pouvons communiquer le résultat à l'utilisateur.

```
ALGORITHME MOYENNE
CONSTANTES C1=2
       C2=3
       C3=1
VARIABLES
        N1, N2, N3 : REEL
        MOY : REEL
DEBUT
{Affichage message : Invitation de l'utilisateur à introduire des données}
        ECRIRE (" Donner trois valeurs réelles ")
{Saisie des notes}
        LIRE (N1, N2, N3)
{Calcul de la moyenne}
        MOY← (N1*C1+N2*C2+N3*C3) / (C1+C2+C3)
{Affichage du résultat sur l'écran}
        ECRIRE (" La moyenne est = ", MOY)
FIN
```

Remarque : Le texte entre les accolades est purement explicatif, il sert à rendre l'algorithme plus lisible.

CHAPITRE II. LES ACTIONS ALGORITHMIQUES SIMPLES

Objectif : Comprendre les actions algorithmiques simples et connaître leurs syntaxes

Éléments de contenu :

- Concepts de base
- La saisie de données
- L'affichage
- L'affectation
- L'évaluation d'une expression arithmétique

II.1. Concepts de base

Dans tout ce qui suit, pour présenter les syntaxes, on suit les règles suivantes :

- Ce qui est entre les crochets est optionnel.
- La suite des points de suspensions "…" veut dire que ce qui précède peut se répéter plusieurs fois.
- Le symbole " | " veut dire : " ou bien ".
- Les mots en majuscule sont des mots réservés.
- Ce qui est entre accolades est un commentaire, pour la lisibilité des algorithmes.

II.2. L'affichage : ECRIRE

Cette action permet de communiquer un résultat ou un message sur écran ou sur imprimante pour l'utilisateur.

Syntaxe

ECRIRE(paramètre1 [[,paramètre2]…])
Paramètre = variable | expression | constante
Constante = nombre | message

Exemples

ECRIRE(" La valeur de 3*2 est égale à ", 3*2)
↑ ↑
message expression

ECRIRE(" La moyenne est = ", MOY)
↑
Variable

II.3.2. La saisie des données : LIRE

L'ordre LIRE permet à l'ordinateur d'acquérir des données à partir de l'utilisateur, dans des cases mémoire bien définies (qui sont les variables déclarées).

Rappel

Les variables sont des cases mémoire, supposées contenir un type de données, nommées par le nom de variable.

Syntaxe

LIRE(variable1 [[, variable2] …])

Remarques :

1. La saisie se fait uniquement dans des variables. Ce sont les cases (cellules) qui pourront accueillir les données correspondantes.
2. La donnée à introduire doit être de même type que la variable réceptrice.

II.4.3. Les expressions arithmétiques

Parmi les opérateurs, on distingue les fonctions et les opérateurs.

Les fonctions

- La fonction **DIV** permet de donner le résultat de la division entière d'un nombre par un autre. 7 DIV 2 → 3
- La fonction **MOD** (se lit Modulo), permet de donner le reste de la division entière d'un entier par un autre. 7 MOD 2 → 1
- La fonction ****** ou ^ permet d'élever un nombre à la puissance d'un autre. 2**3 →8

Les opérateurs

- Sont le "+", "-", "/", "*"

Ordre de priorité

Les opérateurs suivants sont ordonnés du plus prioritaire au moins prioritaire dans l'évaluation d'une expression arithmétique.

1- Les parenthèses
2- Les fonctions
3- Les opérateurs de multiplication " * " et de division " / "
4- Les opérateurs d'addition " + " et de soustraction " - "

Remarque

Si l'ordre entre les opérateurs dans une expression est le même, on évalue l'expression de gauche à droite.

Exemples

3**2+4 = 9+4=13
3**(2+4)=3**6 car les parenthèses sont plus prioritaires
17 MOD 10 DIV 3=(17MOD10)DIV3=7DIV3=2

II.5.4. L'affectation

C'est l'action de charger une valeur dans une variable. Cette valeur peut elle-même être une variable, le résultat d'une expression arithmétique ou logique ou une constante.

Syntaxe

Variable1 ← variable2 | expression | constante

A ← B **se lit** " A reçoit B "

Le résultat de cette action est de mettre le contenu de la variable B dans la variable A. Si B était une expression, elle aurait été évaluée, ensuite sa valeur est transférée dans la variable réceptrice (à notre gauche).

Remarque

L'affectation ne vide pas la variable émettrice (à notre droite) de sa valeur. Par contre, le contenu de la variable réceptrice est écrasé et remplacé par la nouvelle valeur.

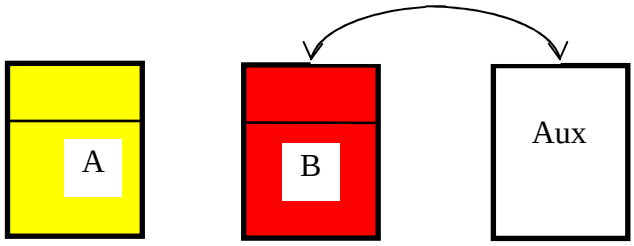
Illustration de l'affectation

Supposons qu'on ait deux récipients A et B où A contient un liquide coloré en jaune et B contient un liquide rouge.
Peut-on échanger les contenus de A et de B (c.-à-d. mettre le liquide rouge dans A et le liquide jaune dans B).

Résultat

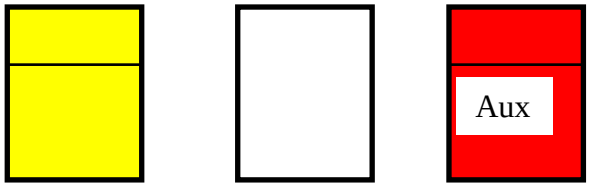
Cette opération n'est possible que si on utilise un troisième récipient qu'on appelle récipient auxiliaire.

Etat 0



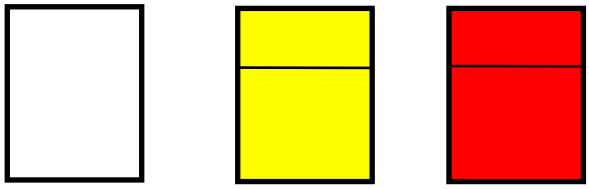
A = Jaune
B = Rouge
Aux = Vide

Etat 1



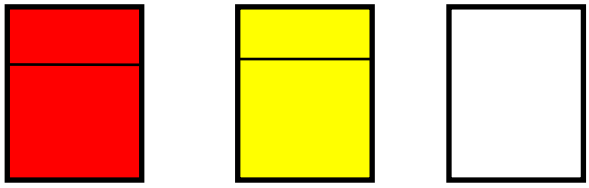
Aux←B
A = Jaune
B = vide
Aux = Rouge

Etat 2



B←A
A = Vide
B = Jaune
Aux = Rouge

Etat Final



A←Aux
A = Rouge
B = Jaune
Aux = Vide

Avec des variables réelles, cette opération d'échange de contenu se fait entre cases mémoires qui représentent les conteneurs (récipients).
Problème : Echanger les valeurs de 2 variables numériques.
Principe : pour éviter de perdre l'une des 2 valeurs initiales (A et B), on utilise une 3^{ième} variable pour préserver la valeur initiale de la première variable modifiée.

Remarques Importantes

- Toute variable utilisée dans un algorithme doit être déclarée au début de l'algorithme, une fois et une seule.

- L'affectation de valeur à une variable peut être effectuée autant de fois que l'on veut au cours d'un algorithme. La valeur de la variable sera alors modifiée à chaque affectation.
- Lorsqu'une variable apparaît en partie droite d'une action d'affectation, c'est que l'on suppose qu'elle contient obligatoirement une valeur. Cette valeur devra lui avoir été affectée auparavant (par initialisation ou saisie), sinon l'on dira que la valeur est indéfinie, et le résultat de l'affectation ne sera pas défini.
- La variable réceptrice d'une affectation doit être de même type que de la valeur à affecter ou de type compatible. Le type est dit compatible s'il est inclus dans le type de la variable réceptrice. Exemple : REEL ← ENTIER est possible mais pas l'inverse.

Exemple

Ecrire l'algorithme qui permet de calculer le discriminant Δ (delta) d'une équation du second degré.

TD 1 : Les actions simples

Exercice 1

Soit l'algorithme suivant :

```
ALGORITHME EQUATION2D
VAR a,b,c : REEL
        delta : REEL
DEBUT
  Ecrire("Donnez la valeur du premier paramètre")
  Lire(a)
  Ecrire("Donnez la valeur du second paramètre")
  Lire(b)
  Ecrire("Donnez la valeur du troisième paramètre")
  Lire(c)
  delta ←b^2 b * b - 4a * c
  Ecrire(" le discriminant est = Δ ")
Fin
```

1 - Décrire cet algorithme en détail (ligne par ligne), en donnant les éventuelles erreurs.
2 - Quelles sont les valeurs de delta dans les cas suivants :

a=2 b=-3 c=1
a=1 b=2 c=2

Exercice 2

Ecrire l'algorithme permettant de saisir l'abscisse d'un point A et de calculer son ordonné $f(x)= 2x^3 - 3x^2 + 4$
Evaluer le résultat en expliquant les ordres de priorité pour x=-2.

Exercice 3

Ecrire l'algorithme qui permet de permuter les valeurs de A et B sans utiliser de variable auxiliaire.

Exercice 4

Faire l'algorithme qui lit les coordonnées de deux vecteurs u et v, et de calculer leur norme et leur produit scalaire.

Exercice 5

Ecrire l'algorithme qui permet de saisir les paramètres d'une équation du second degré et de calculer son discriminantΔ.

Exercice 6

Ecrire l'algorithme permettant de calculer et d'afficher le salaire net d'un employé. Sachant que :

- Le salaire net = Salaire brut – Valeur de l'impôt – Valeur de CNSS
- Salaire brut = (Salaire de base + Prime de technicité + Prime de transport + Prime des enfants) * Taux de travail
- Taux de travail = Nombre de jours travaillés / 26
- Prime des enfants = Prime d'un enfant * Nombre d'enfants
- Valeur de l'Impôt = Taux de l'Impôt * Salaire Brut

- Valeur de CNSS = Taux de CNSS * Salaire Brut
- Taux CNSS = 26,5%
- Taux Impôt = 2%

Indication :
Décrire l'environnement de travail : toutes les variables en entrée, en sortie et de calcul.

CHAPITRE III. LES STRUCTURES CONDITIONNELLES

III.1. Introduction

Souvent les problèmes nécessitent l'étude de plusieurs situations qui ne peuvent pas être traitées par les séquences d'actions simples. Puisqu'on a plusieurs situations, et qu'avant l'exécution, on ne sait pas à quel cas de figure on aura à exécuter, dans l'algorithme on doit prévoir tous les cas possibles.
Ce sont les ***structures conditionnelles*** qui le permettent, en se basant sur ce qu'on appelle ***prédicat*** ou ***condition***.

III.2. Notion de PREDICAT

Un prédicat est un énoncé ou proposition qui peut être vrai ou faux selon ce qu'on est entrain de parler.
En mathématiques, c'est une expression contenant une ou plusieurs variables et qui est susceptible de devenir une proposition vraie ou fausse selon les valeurs attribuées à ces variables.
Exemple :
(10 < 15) est un prédicat vrai
(10 < 3) est un prédicat faux

III.3. Evaluation d'une expression logique

Une condition est une expression de type logique. Ils lui correspondent deux valeurs possibles VRAI et FAUX qu'on note par V ou F.
Considérons deux variables logiques A et B. Voyons quels sont les opérateurs logiques et leurs ordres de priorités.

1-a) Notons que

- ✓ (A = faux) ⇔ non A
- ✓ (A = vrai) ⇔ A

Les opérateurs logiques sont :

- ✓ La négation : "non"
- ✓ L'intersection : "et"
- ✓ L'union : "ou"

1-b) Notation et Ordre de priorité des opérateurs logiques

1. non : ¬
2. et : ∧
3. ou : ∨

1-c)

1-d) Tableaux d'évaluations

La négation d'une condition

A	Non A
Vrai	Faux
Faux	Vrai

L'intersection de deux conditions

A et B	Vrai	Faux
Vrai	Vrai	Faux
Faux	Faux	Faux

L'union de deux conditions

A ou B	Vrai	Faux
Vrai	Vrai	Vrai
Faux	Vrai	Faux

Théorème de DE MORGAN

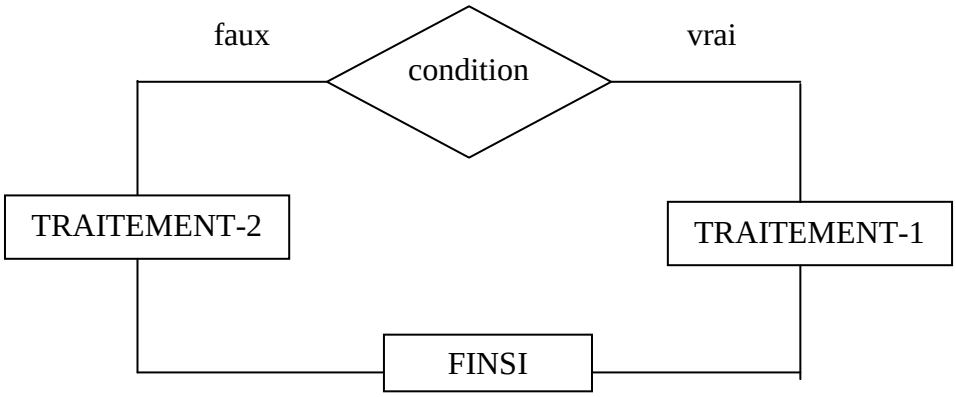
- ✓ $\neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B$
- ✓ $\neg(A \vee B) \Leftrightarrow \neg A \wedge \neg B$

III.4. La structure conditionnelle SI

Syntaxe

SI <Condition> ALORS
    <suite d'action(s)-1>
[SINON
    <suite d'actions(s)-2>]
FINSI

Format Organigramme



- La <condition> est un prédicat, qui peut être vrai ou faux, selon les valeurs des paramètres la constituant.
- Si la condition est vérifiée (sa valeur est vrai), c'est la <suite d'actions-1> qui sera exécutée. Ensuite, le système passe à l'exécution juste après le FINSI.
- Dans le cas contraire, lorsque la condition n'est pas vérifiée (valeur de la condition est faux), c'est la <suite d'actions-2> qui s'exécute, en cas où celle ci existe (facultative). Si elle n'existe pas, le système passe directement à l'instruction qui suit le FINSI.
- Les suites d'actions 1 et 2, peuvent être des actions simples ou même des structures conditionnelles.

Exemple 1

Lire un nombre réel, et dire s'il est positif ou strictement négatif.

```
ALGORITHME POS-NEG
VARIABLES A : réel
DEBUT
     ECRIRE("Donner un nombre ")
     LIRE(A)
     SI (A < 0) ALORS
          ECRIRE(A, " est négatif ")
     SINON
          ECRIRE(A, " est positif ")
     FINSI
FIN
```

Autrement :

```
ALGORITHME POS-NEG-1
VARIABLES A : réel
     B : logique
DEBUT
     ECRIRE("Donner un nombre ")
     LIRE(A)
     B ← (A < 0)
     SI (B) ALORS
          ECRIRE(A, " est négatif ")
     SINON
          ECRIRE(A, " est positif ")
     FINSI
FIN
```

Dans cet exemple, on a déterminé uniquement les cas de positivité ou de négativité, et on n'a pas déterminé le cas où A est nulle.

```
ALGORITHME POS-NEG-NUL
VARIABLES A : réel
DEBUT
     ECRIRE("Donner un nombre ")
     LIRE(A)
     SI (A < 0) ALORS
          ECRIRE(A, " est négatif ")
     SINON {A >= 0}
       SI (A > 0)ALORS
          ECRIRE(A, " est positif ")
       SINON {A = 0}
          ECRIRE (A, "est nulle")
       FINSI
     FINSI
FIN
```

Exemples

1) Ecrire l'algorithme qui permet de déterminer si un entier lu est pair ou impair.
2) Ecrire l'algorithme qui permet de saisir deux nombres A et B et de déterminer si la valeur de A est supérieure, inférieure ou égale à B.

III.5. La structure conditionnelle SELON

Cette structure conditionnelle est appelée aussi ***à choix multiple*** ou ***sélective*** car elle sélectionne entre plusieurs choix à la fois, et non entre deux choix alternatifs (le cas de la structure SI).

Syntaxe

SELON (sélecteur) FAIRE
Cas <liste de valeurs-1> : <suite d'action (s)-1>
[Cas <liste de valeur-2> : <suite d'action (s)-2>
…….]
[SINON : <suite d'action (s)-n>]
FINSELON

Le ***sélecteur*** peut être une variable de type scalaire ou une expression arithmétique ou logique.
La structure SELON évalue le "sélecteur", passe à comparer celui ci respectivement avec les valeurs dans les listes. En cas d'égalité avec une valeur, les actions correspondantes, qui sont devant cette valeur seront exécutées.
Devant "Cas", il peut y avoir une seule valeur, une suite de valeurs séparées par des virgules et/ou un intervalle de valeurs.
Après avoir traiter la suite d'actions correspondante, l'exécution se poursuit après le FINSELON.

Remarque

1. Le sélecteur doit avoir le même type que les valeurs devant les cas.
2. Le type de ces valeurs ne doit être, ni réel ni chaîne de caractères.

Exemple

Ecrire l'algorithme qui permet de saisir un numéro de couleur de l'arc-en-ciel et d'afficher la couleur correspondante : 1: rouge, 2 : orangé, 3 : jaune, 4 : vert, 5 : bleu, 6 : indigo et 7 : violet.

TD 2 : Les Structures Conditionnelles

Exercice 1

Evaluer les expressions logiques suivantes, avec (a, b, c, d) = (2, 3,5, 10) et (X, Y) = (V, F).

1) $(a < b) \wedge (a < c)$	2) $\neg ((a < b) \wedge (a < c))$	3) $\neg (a < b) \wedge (a < c)$
4) $(a < c) \wedge (c = d/2)$	5) $(d / a = c) = Y$	6) $(d / c = b) = Y$
7) $(d / c = b) = X$	8) $(a < b) \wedge (d < c)$	9) $(a < b) \wedge (d < c) = X$

Exercice 2

Réécrire l'exercice 6 de la série N°1 en supposant que le taux de l'impôt n'est pas fixe mais il varie selon la valeur du salaire de base. En effet :

- ✓ Taux de l'impôt = 0 si le salaire de base < 150
- ✓ Taux de l'impôt = 2% si le salaire de base $\in$[150,250[
- ✓ Taux de l'impôt = 5% si le salaire de base $\in$[250,500[
- ✓ Taux de l'impôt = 12% si le salaire de base >= 500.

En plus, la prime des enfants est définit comme suit :

- ✓ 7DT pour le premier enfant,
- ✓ 5DT pour le deuxième enfant,
- ✓ 4DT pour le troisième enfant.
- ✓ Pas de prime pour le reste.

Exercice 3

Ecrire l'algorithme qui permet de saisir un nombre puis déterminer s'il appartient à un intervalle donné, sachant que les extrémités de l'intervalle sont fixées par l'utilisateur.

Exercice 4

Ecrire l'algorithme qui permet de calculer le montant des heures supplémentaires d'un employé, sachant le prix unitaire d'une heure selon le barème suivant :

- Les 39 premières heures sans supplément,
- De la $40^{ième}$ à la $44^{ième}$ heure sont majorées de 50%,
- De la $45^{ième}$ à la $49^{ième}$ heure sont majorées de 75%,
- De la $50^{ième}$ heure ou plus, sont majorées de 100%.

Exercice 5

Ecrire l'algorithme qui permet de saisir la moyenne générale d'un étudiant et de déterminer son résultat et sa mention. (les conditions de rachat sont appliquées à partir de 9,75.

Exercice 6

Ecrire l'algorithme qui permet de saisir les trois paramètres d'une équation du second degré, et de discuter les solutions selon les valeurs de a, b et c, lorsqu'elles sont nulles ou pas.

Exercice 7

Ecrire l'algorithme qui permet de saisir le jour, le mois et l'année d'une date (Mois : numéro du mois), et de déterminer si elle est correcte ou non, et où est l'erreur.

Exercice 8

Ecrire l'algorithme qui permet de saisir deux nombres, et un opérateur et d'évaluer l'expression arithmétique correspondante.

Exercice 9

Ecrire l'algorithme CONTRAT qui permet d'aider une compagnie d'assurance à prendre une décision concernant les demandes d'affiliation en se basant sur les critères suivants :

CRITERE / DECISION	AGE	Bonne santé	Accident
Contrat A	<=30	OUI	NON
Contrat B	>30	OUI	OUI
Contrat refusé	-	NON	OUI
Expertise demandée	-	OUI	OUI

Exercice 10

Ecrire un algorithme qui permet de saisir un numéro de mois et un jour (le contrôle n'est pas demandé) et d'afficher la période correspondante selon le tableau suivant :

Période	**DU**	**AU**
Vacances d'été	1/7	15/9
Premier trimestre	16/9	19/12
Vacances d'hiver	20/12	3/1
Deuxième trimestre	4/1	19/3
Vacances de printemps	20/3	3 / 4
Troisième trimestre	4/4	30/6

Exercice 11

Ecrire l'algorithme permettant de lire la valeur de la variable DEVINETTE et d'afficher parmi les messages suivants celui qui correspond à la valeur trouvée :
ROUGE si la couleur vaut R ou r
VERT si la couleur vaut V ou v
BLEU si la couleur vaut B ou b
NOIR pour tout autre caractère.

Exercice 12

Ecrire l'algorithme permettant de lire la valeur de la température de l'eau et d'afficher son état :
GLACE Si la température inférieure à 0,
EAU Si la température est strictement supérieure à 0 et inférieure à 100,
VAPEUR Si la température supérieure à 100.

Exercice 13

Ecrire l'algorithme qui lit un entier positif inférieur à 999 (composé de trois chiffres au maximum) et d'afficher le nombre de centaines, de dizaines et d'unités.

CHAPITRE IV. LES STRUCTURES REPETITIVES

IV.1. Introduction

Dans les problèmes quotidiens, on ne traite pas uniquement des séquences d'actions, sous ou sans conditions, mais il peut être fréquent d'être obligé d'exécuter un traitement (séquence d'actions), plusieurs fois. En effet, pour saisir les N notes d'un étudiant et calculer sa moyenne, on est amené à saisir N variables, puis faire la somme et ensuite diviser la somme par N. Cette solution nécessite la réservation de l'espace par la déclaration des variables, et une série de séquences d'écriture/lecture. Ce problème est résolu à l'aide des structures répétitives. Celles ci permettent de donner un ordre de répétition d'une action ou d'une séquence d'actions une ou plusieurs fois.

IV.2. La boucle POUR

Cette structure exprime la répétition d'un traitement un nombre de fois.

Syntaxe

POUR Vc DE Vi A Vf [PAS Vp] FAIRE
    <Traitement>
FINFAIRE

Où Vc est une variable entière, qui compte le nombre de répétition du <Traitement>,
Vi la valeur initiale à laquelle Vc est initialisé,
Vf la valeur finale à laquelle se termine Vc,
Vp la valeur du pas, c'est la valeur qu'on rajoute à Vc à chaque fin de traitement.

Remarque

1. La boucle POUR est utilisée lorsqu'on connaît le nombre de répétition du <Traitement> d'avance.
2. La valeur du pas peut être positive ou négative et par conséquent, il faut; au départ de la boucle; que Vi <= Vf ou Vi >= Vf selon la positivité ou la négativité de cette valeur.
3. La valeur du pas est égale à 1 par défaut.

Les étapes d'exécution de la boucle POUR

1) Initialisation de Vc par la valeur de Vi (comme si on avait Vc←Vi)
2) Test si Vi dépasse (±) Vf (du côté supérieur ou inférieur, selon la positivité ou la négativité du pas).
   Si oui, alors la boucle s'arrête et l'exécution se poursuit après le FINFAIRE
   Sinon,
   - Exécution du <Traitement>,
   - Incrémentation ou décrémentation de Vc par la valeur du pas,
   - Retour à l'étape 2.

Application

Ecrire l'algorithme qui permet de saisir les moyennes des N étudiants de la classe Informatique et de calculer la moyenne générale de la classe.

Résolution

Sans les boucles, on est obligé de déclarer N variables, et d'écrire N actions LIRE.

N fois

LIRE(note)
S← S + MOY
LIRE(MOY)
S← S + MOY

…..
LIRE(MOY)
S← S + MOY

La boucle POUR donne l'ordre à la machine d'itérer les deux actions LIRE(MOY) / S←S+ MOY N fois.
Donc le compteur varie de 1 jusqu'à N avec un pas de 1.

```
ALGORITHME MOYENNE
VARIABLES i, N : entier
     MOY, MC : réel
DEBUT
ECRIRE("Donner le nombre d'étudiants")
LIRE(N)
SI (N > 0) ALORS
     S ← 0 {Initialisation de S}
     POUR i DE 1 A N FAIRE {Le pas égale 1 par défaut}
          ECRIRE("Donner la moyenne de l'étudiant n°", i)
          LIRE(MOY)
          S ← S + MOY {on rajoute la moyenne du i ième étudiant à la somme}
     FIN FAIRE
     MC ← S / N
     ECRIRE("La moyenne de la classe est : ", MC)
SINON
     ECRIRE("Erreur dans le nombre d'étudiants")
FINSI
FIN
```

Remarque Juste Avant le FIN FAIRE, le changement de la valeur de i se fait automatiquement.

Application 1

Ecrire l'algorithme qui permet d'afficher tous les nombres pairs qui existent entre 1 et 10.

1ière solution

```
POUR i de 2 à 10 pas 2 Faire
     ECRIRE(i)
FINFAIRE
```

2ième solution

```
POUR i de 2 à 10 Faire
SI (i mod 2 = 0) ALORS
          ECRIRE(i)
FINSI
FINFAIRE
```

3ième solution

```
POUR i de 1 à 5 Faire
     ECRIRE(2*i)
FINFAIRE
```

Application 2

Ecrire l'algorithme qui permet d'afficher tous les nombres impairs entre 50 et 100 dans l'ordre décroissant.

```
POUR i de 99 à 50 PAS (-2) FAIRE
        ECRIRE(i)
FIN FAIRE
```

La valeur finale peut être 50 ou 51 car le test de sortie est i < Vf (49 < 50 ou à 51)

IV.3. La boucle Répéter … Jusqu'à

Syntaxe

```
Répéter
        <Traitement>
Jusqu'à (condition d'arrêt)
```

Cet ordre d'itération permet de répéter le <Traitement> une ou plusieurs fois et de s'arrêter sur une condition. En effet, lorsque la condition est vérifiée, la boucle s'arrête, si non elle ré-exécute le <Traitement>.

Remarques

1. Dans cette boucle, le traitement est exécuté au moins une fois avant l'évaluation de la condition d'arrêt.
2. Il doit y avoir une action dans le <Traitement> qui modifie la valeur de la condition.

Les étapes d'exécution de la boucle Répéter

1) Exécution du <Traitement>
2) Test de la valeur de la <condition d'arrêt>
   Si elle est vérifiée Alors la boucle s'arrête
   Sinon Retour à l'étape 1.

Application

Ecrire un algorithme qui saisit un nombre pair et qui détermine combien de fois il est divisible par 2. Exemple 8 est divisible 3 fois par 2 (2*2*2).

```
ALGORITHME PAIR-NBDIV2
VARIABLES N, N2 : entier
DEBUT
{Saisie d'un entier qui doit être pair}
Répéter
     ECRIRE("Donner un entier pair")
     LIRE(N)
Jusqu'à (N MOD 2 = 0) {condition pour que N soit pair}
{Détermination du nombre de division par 2}
N2 ← 0
NB ← N
Répéter
     NB ← NB div 2
     N2 ← N2 +1
Jusqu'à (NB MOD 2 <> 0) {On s'arrête lorsque NB n'est plus divisible par 2}

ECRIRE(N, "est divisible par 2", N2,"fois")
FIN
```

IV.4. La boucle TANT QUE …

Syntaxe

TANT QUE (condition d'exécution)
FAIRE
    <Traitement>
FIN FAIRE

Cet ordre d'itération permet de répéter le <Traitement> **zéro** ou plusieurs fois et de s'arrêter lorsque la condition d'exécution n'est plus vérifiée. En effet, lorsque la condition d'exécution est vérifiée, le <Traitement> est exécuté, si non elle s'arrête.

Les étapes d'exécution de la boucle Répéter

1) Test de la valeur de la <condition d'exécution>
2) Si elle est vérifiée Alors
       Exécution du <Traitement>
       Retour à l'étape 1.
   Sinon Arrêt de la boucle.

Remarques

1. Dans cette boucle, le traitement peut ne pas être exécuté aucune fois, c'est lorsque la condition d'exécution est à faux dés le départ.
2. Les paramètres de la condition doivent être initialisés par lecture ou par affectation avant la boucle.
3. Il doit y avoir une action dans le <Traitement> qui modifie la valeur de la condition.

Application

Ecrire un algorithme qui saisit un nombre pair et qui détermine combien de fois il est divisible par 2. Exemple 8 est divisible 3 fois par 2 (2*2*2).

```
ALGORITHME PAIR-NBDIV2
VAR N, N2 : entier
DEBUT
{Saisie d'un entier qui doit être pair}
Répéter
     ECRIRE("Donner un entier pair")
     LIRE(N)
Jusqu'à (N MOD 2 = 0) {condition pour que N soit pair}
{Détermination du nombre de division par 2}
N2 ← 0
NB ← N
TANT QUE (NB MOD 2 = 0)
FAIRE
     NB ← NB div 2
     N2 ← N2 +1
FIN FAIRE {On s'arrête lorsque NB n'est plus divisible par 2}
ECRIRE(N, "est divisible par 2", N2,"fois")
FIN
```

- La condition d'arrêt avec la boucle Répéter est l'inverse de la condition d'exécution de la boucle TANTQUE.

Remarque
Le Traitement d'une boucle peut contenir lui aussi une autre boucle. On l'appelle dans ce cas des boucles imbriquées.

TD 3 : Les structures répétitives

Exercice 1
Ecrire l'algorithme qui permet d'afficher les N premiers entiers impairs dans l'ordre décroissant.

Exercice 2
Ecrire l'algorithme qui permet d'afficher les diviseurs d'un entiers N.

Exercice 3
Ecrire l'algorithme qui détermine si une entier N est parfait ou non. Un entier est dit parfait s'il est égal à la somme de ses diviseurs. Exemple 6 = 3 + 2 +1

Exercice 4
Ecrire l'algorithme qui permet de calculer le produit de deux entiers en utilisant des additions successives.

Exercice 5
Ecrire l'algorithme qui permet de calculer la division de deux entiers en utilisant des soustractions successives

Exercice 6
Ecrire l'algorithme qui permet de saisir un entier N et d'afficher s'il est premier ou non. Un nombre est dit premier s'il est divisible uniquement par 1 et par lui-même.

Exercice 7
Ecrire l'algorithme qui détermine le $20^{ième}$ terme d'une suite définie par :
$S_0 = 2$, $S_1 = 3$ et $S_n = S_{n-2} + (-1)^n * S_{n-1}$

Exercice 8
Ecrire l'algorithme qui détermine le $N^{ième}$ terme d'une suite définie par :
$S_0 = 2$, $S_1 = 3$, $S_2 = -2$ et $S_n = S_{n-3} + (-1)^n * S_{n-1}$

Exercice 9
On démontre en mathématique que le cosinus d'un angle exprimé en radian est donné par la somme infinie suivante :
$COS(x) = 1 – X^2 / 2! + X^4/4! – X^6/6! +…$
On décide d'arrêter la somme à un certain rang n (n>3) donné.
Ecrire l'algorithme qui permet d'évaluer le cosinus d'une valeur x donnée.

Exercice 10
Ecrire l'algorithme qui permet de saisir autant de nombres que l'utilisateur le veuille, et de déterminer le nombre de réels strictement positifs et celui des négatifs. On s'arrête lorsque la valeur est 999.

Exercice 11
Ecrire l'algorithme qui permet de saisir autant de nombres que l'utilisateur le veuille, pourvu qu'ils soient dans l'ordre croissant. On s'arrête lorsque la valeur est 999.

Exercice 12
Ecrire l'algorithme qui permet de saisir un entier positif en décimal et de le transformer en binaire.
Exemple $(7)_{10} = (111)_2$

Exercice 13
Ecrire un algorithme qui permet de saisir un entier et une base inférieure ou égale à 10 et de vérifier si ce nombre appartient à la base ou non.

Exercice 14
Ecrire un algorithme qui permet de saisir deux entiers et de vérifier si les chiffres du premier appartiennent à ceux du second nombre ou non.

Exercice 15
Ecrire un algorithme qui permet de saisir deux entiers positifs et de déterminer leur plus grand commun diviseur (PGCD).
Le PGCD(A,B) = PGCD(A-B, B) si A est le plus grand et
à PGCD(A,B) = PGCD(A, B-A) si B est le plus grand. Si A=B le PGCD(A,B) est A ou B.

Exercice 16
Ecrire un algorithme qui permet de calculer la factorielle d'un entier N donné.

Exercice 17
Ecrire un algorithme qui permet de saisir des entiers alternatifs (si l'un est positif sont suivant doit être négatif et vice versa).

Exercice 18
Ecrire l'algorithme qui permet de saisir deux entiers et de déterminer leur plus petit commun multiple (PPCM).

CHAPITRE V. TRAITEMENT DES TABLEAUX

Pourquoi les tableaux ?

1) Calculer la moyenne de 30 élèves
2) Effectuer leur classement

↳ ***Réponse***

pour i de 1 à 30
faire
    Ecrire (" Donner la moyenne de l'étudiant N°",i)
    Lire (moyenne)
Fin faire

↳ ***Conclusion*** : On ne peut pas effectuer le classement
Pourquoi ? Parce qu'on ne garde pas les moyennes précédentes et la variable moyenne contient uniquement la dernière valeur.

V.1. Utilisation des tableaux

Intérêt Gain de temps, rétrécissement du volume de l'algorithme et possibilité de réutilisation de toutes les valeurs ultérieurement dans l'algorithme.
Il est plus convenable, alors, de définir un espace mémoire qu'on appelle MOY qui sera divisé en 30 parties équitables, indicées de 1 à 30.

MOY

Contenu →	15	12	5	10	4	50	….						
Indice →	1	2	3	4	5	6	7	8	9	10	11	12	13

On définit un tableau de 30 cases à une seule dimension qu'on appelle VECTEUR.

```
ALGORITHME MOYENNE
CONSTANTES
      Bi=1
      Bs=30
VARIABLES
      T : Tableau [bi..bs] de réel
     i : entier
```

V.2. Les vecteurs

Un vecteur est une partie de mémoire contenant n zones variables référencées par le même nom de variable pour accéder à un élément particulier de ce vecteur.
On indice le nom de variable. L'indice peut être une constante, une variable ou une expression arithmétique.

```
MOY[i]
```

↑ ↖ indice d'un élément du vecteur
variable qui indique le nom du vecteur

MOY[i] : représente l'élément du vecteur MOY occupant le rang " i ".
L'indice peut être :

- Une constante →MOY[5]
- Une variable →MOY[i]
- Une expression →MOY[i*2]

ATTENTION
Avant d'utiliser un tableau, il faut déclarer sa taille pour que le système réserve la place en mémoire, nécessaire pour stocker tous les éléments de ce tableau.
Les éléments d'un même tableau doivent être de même type.

V.2.1 Rappel de Déclaration d'un vecteur

Dans la partie CONST, on peut définir la taille du tableau. Ensuite, on peut déclarer le nombre d'éléments à saisir dans le tableau.
Remarque : Le nombre d'éléments à saisir ne doit pas dépasser la taille du tableau pour ne pas déborder sa capacité.
On appelle dimension d'un vecteur le nombre d'éléments qui constituent ce vecteur.

V.2.2 Chargement d'un Vecteur

Le chargement d'un vecteur consiste à saisir les données des éléments du vecteur. (Remplir des cases successives du tableau). On doit utiliser une boucle qui permet de saisir à chaque entrée dans la boucle la $i^{ième}$ case.

```
ALGORITHME          Vecteur
CONSTANTES          N = 30
VARIABLES
        MOY : Tableau[1..N] de réels
Début
{Chargement du tableau}
Pour i de 1 à N
Faire
        Ecrire (" donner la moyenne de l'étudiant N° " , i )
        Lire ( MOY [i])
Fin Faire
{ fin chargement }

{Calcul de la somme des moyennes}
SMOY ← 0
Pour i de 1 à N
Faire
        SMOY←SMOY+MOY[i]
Fin Faire

SMOY ← SMOY / 30
Ecrire (" la moyenne du groupe est ", SMOY )
{ calcul de la différence entre la moyenne de groupe et celle  de l'étudiant }
Pour i de 1 à N
```

Faire
    Ecrire (" la différence de la moyenne du groupe et celle de l'étudiant ",i , " est= ", SMOY-MOY[i])
Fin Faire
Fin

✍ On peut écrire les deux premières boucles en une seule. Simplifier alors cet algorithme.

Remarque
La taille d'un tableau est fixe et ne peut être donc changée dans un programme : il en résulte deux défauts :
- Si on limite trop la taille d'un tableau on risque le dépassement de capacité.
- La place mémoire réservée est insuffisante pour recevoir toutes les données.

Application
1) Charger un vecteur de 10 éléments par les 10 premiers entiers naturels positifs.
2) Charger un vecteur de 10 éléments par les 10 premiers multiples de 7.

V.2.3 Recherche dans un vecteur

Recherche séquentielle
On peut chercher le nombre d'apparition d'un élément dans un vecteur, sa ou bien ses positions. Pour cela, on doit parcourir tout le vecteur élément par élément et le comparer avec la valeur de l'élément à chercher.

Applications
1. Chercher la position de la première occurrence d'un élément e dans un vecteur V contenant N éléments. (On suppose que le vecteur est définit)
2. Chercher le nombre d'apparition d'un élément e dans un vecteur V contenant N éléments, ainsi que les positions des occurrences de cet élément.

Réponse 1
```
i ← 1
Trouv ← vrai
Tant que ((i <= N) et (Trouv = vrai))
     Faire
          Si V[i] = e Alors
               Trouv ← Faux
          Sinon
               i ← i +1
          Fin Si
     Fin Faire
Si (Trouv = vrai) Alors
     Ecrire(e, "se trouve à la position" , i)
Sinon
     Ecrire(e, "ne se trouve pas dans V")
Fin Si
```

Recherche dichotomique
Ce type de recherche s'effectue dans un tableau ordonné.
Principe
1. On divise le tableau en deux parties sensiblement égales,

2. On compare la valeur à chercher avec l'élément du milieu,
3. Si elles ne sont pas égales, on s'intéresse uniquement la partie contenant les éléments voulus et on délaisse l'autre partie.
4. On recommence ces 3 étapes jusqu'à avoir un seul élément à comparer.

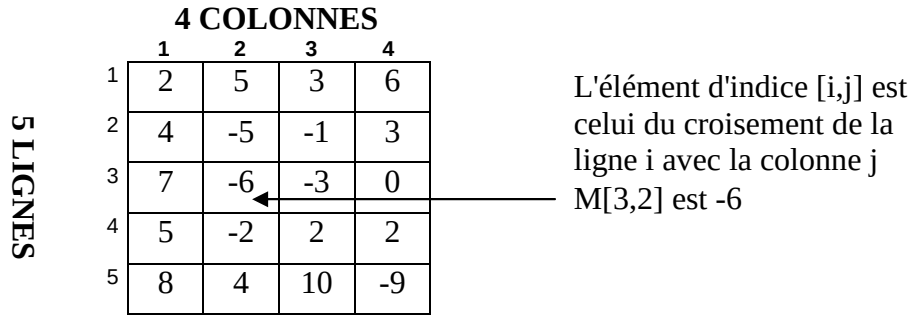
Application
On suppose qu'on dispose d'un vecteur V de N éléments. On veut chercher la valeur Val.

```
ALGORITHME DICHOTHOMIE
...
Inf ← 1
Sup ← N
Trouv  ← vrai
Tant que ((Inf <= Sup) et (Trouv = vrai))
Faire
     Mil← (Inf+Sup)DIV 2
     Si (V[Mil] = Val) Alors
          Trouv← faux
     Sinon
          Si (V[Mil] < Val) Alors
               Inf ← Mil + 1
          Sinon
               Sup ← Mil -1
          Fin Si
     Fin Si
Fin Faire

Si (Trouv = faux) Alors
     Ecrire(Val, "existe à la position" , Mil)
Sinon
     Ecrire(Val, "n'existe pas dans V)
Fin Si
```

V.3. 2. Les matrices

Les matrices sont les tableaux à deux dimensions.



TD 4 : Les tableaux

Exercice 1
Ecrire un algorithme qui lit 4 notes et les met dans un tableau, puis affiche la moyenne.
Exercice 2
Ecrire un algorithme qui lit une chaîne de caractères et affiche le nombre de voyelle dans cette chaîne.
Exercice 3
Ecrire un algorithme qui lit 20 valeurs réelles et qui détermine la moyenne des valeurs strictement positives et la moyenne des valeurs strictement négatives.
Exercice 4
Ecrire un algorithme qui lit 35 notes réelles et affiche la note maximale. On n'utilise pas de tableau.
Exercice 5
Ecrire un algorithme qui lit 35 notes réelles et affiche le nombre de notes qui sont au-dessous de la moyenne (<10)
Exercice 6
Ecrire un algorithme qui lit une chaîne de caractères et qui affiche son inverse.
Par exemple :
STOP devient POTS
FRUIT devient TIURF
En utilisant une seule chaîne de caractères
Exercice 7
Ecrire un algorithme qui lit deux chaînes de caractères et qui affiche les caractères en commun.
Exercice 8
Ecrire un algorithme qui met dans un tableau à 2 dimensions la table de multiplication de 1 à 9.
Exercice 9
Ecrire un algorithme qui lit un tableau de 10 entiers et met les entiers négatifs à droite et les entiers positifs à gauche.
Exercice 10
Ecrire un algorithme qui affiche le résultat de la conversion d'un nombre entier strictement positive dans une base quelconque (comprise entre 2 et 10). Le nombre à convertir ainsi que la valeur de la base sont fournis par clavier.
Exercice 11
Ecrire un algorithme qui lit une chaîne de caractères contenant des parenthèses ouvrantes et fermantes et doit affiché si le nombre de parenthèses est cohérent ou non.
Exercice 12
Ecrire un algorithme qui permet de saisir des chaînes de caractères et de les afficher et ne s'arrête que si on saisit la chaîne "fin".
Exercice 13
Ecrire un algorithme qui permet de lire une chaîne de caractères et d'afficher si elle est palindrome ou non.
Une chaîne est palindrome si elle se lit de gauche à droite comme elle se lit de droite à gauche. Exemple : aziza, radar, elle.
Exercice 14
Ecrire un algorithme qui lit une chaîne de caractères et affiche le pourcentage des lettres voyelles.
Exercice 15

La crible d'Eratosthène permet de déterminer les nombres premiers inférieurs à une valeur n. Le programme C à écrire consiste pour chaque entier n à rechercher parmi les suivants qui en sont des multiples et à les éliminer. Seul les nombres premiers vont rester.
Ecrire un algorithme qui lit un entier positif et affiche les nombres premiers qui lui sont inférieurs pour n <30 et utiliser un tableau.

Exercice 16

Ecrire un algorithme qui lit une matrice de réels de dimension 5 et 6.

Exercice 17

Ecrire un algorithme qui met dans une matrice la table d'addition de 1 à 9.

Exercice 18

Ecrire un algorithme qui lit une matrice de réel et calcule la somme de ses éléments. La matrice est de dimension 7 et 12.

Exercice 19

Ecrire un algorithme qui lit une matrice de dimension 50 et 20 et affiche le nombre de réels strictement positifs

Exercice 20

Ecrire un algorithme qui lit une matrice de réels positifs de dimension 50 et 20 et affiche le plus grand élément.

Exercice 21

Ecrire un algorithme qui lit une matrice de dimension 12 et 10 et affiche la position du plus petit élément.

Exercice 22

Ecrire un algorithme qui lit une matrice de dimensions 10 et 8 et qui génère une matrice transposée.

Exercice 23

Ecrire un algorithme qui lit un réel et une matrice de dimension 3 et 4 et multiplie cette dernière par ce réel.

Exercice 24

Ecrire un algorithme qui lit deux matrices de dimension 5 et 6 et effectue leur addition dans une nouvelle matrice.

Exercice 25

Ecrire un algorithme qui lit une matrice M de dimension 10 et 10 et un réel x et affiche le nombre d'occurrence de x dans M.

CHAPITRE VI. LES ALGORITHMES DE TRI

Dans ce chapitre on présente quelques algorithmes utiles, qui permettent d'ordonner les éléments d'un tableau dans un ordre croissant ou décroissant. L'ordre est par défaut croissant.

Un vecteur est dit trié si V[i] <= V[i+1], $\forall$ i $\in$ [1..n-1]

VI.1 Tri par sélection

Principe

Utiliser un vecteur VT (vecteur trié) comme vecteur résultat. Celui ci contiendra les éléments du vecteur initial dans l'ordre croissant.

Le principe est de :

0- Chercher le plus grand élément dans le vecteur initial V
1- Sélectionner le plus petit élément dans V
2- Le mettre dans son ordre dans le vecteur VT
3- Le remplacer par le plus grand élément dans le vecteur initial (pour qu'il ne sera plus le minimum)
4- Si le nombre d'éléments dans le vecteur résultat n'est pas identique à celui dans le vecteur initial Retourner à l'étape 1
   Sinon on s'arrête.

Exemple

Soit le vecteur V contenant 4 éléments.

		V				VT			
Au départ		10	15	-1	7				
	Phase 1	10	15	15	7	-1			
	Phase 2	10	15	15	15	-1	7		
	Phase 3	15	15	15	15	-1	7	10	
	Phase 4	15	15	15	15	-1	7	10	15

Schéma de l'algorithme

```
ALGORITHME TRI_SELECTION
CONSTANTES Bi = 1
      Bs = 10
VARIABLES  V, VT : Tableau[Bi..Bs] de réel
     N, i, j, indmin : entier
     MIN, MAX : réel
DEBUT
{Chargement du vecteur V}
…
{Recherche du maximum}
MAX←V[1]
Pour i de 2 à N
FAIRE
 Si MAX < V[i] Alors
     MAX←V[i]
 FINSI
FINFAIRE
```

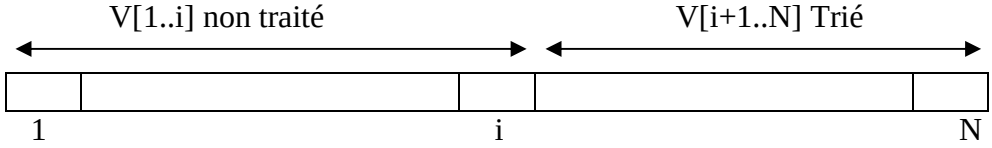
```
POUR i de 1 à N-1 FAIRE
  {Recherche du minimum}
     MIN ← V[1]
     indmin ← 1
     Pour j de 2 à N faire
          Si  MIN > V[j] ALORS
              MIN←V[j]
              Indmin ← j
          Fin si
     Fin Faire
  {Mettre le minimum trouvé à sa place dans le vecteur résultat}
     VT[i] ← MIN
     V[indmin] ← MAX
Fin Faire
VT[N] ← MAX
FIN
```

Peut-on améliorer cet algorithme ?

VI.2 Algorithme de tri par sélection et permutation

Il s'agit ici d'éviter la construction d'un second vecteur et d'utiliser un seul vecteur initial qui sera trié.

Supposons traités n-i (1 <= i < N) éléments du vecteur.



On peut considérer le vecteur V comme la concaténation de deux sous-vecteurs : le sous-vecteur V[1..i] dont les éléments n'ont pas encore été triés, et le sous vecteur V[i+1..N] dont les éléments sont triés. D'autre part tous les éléments du sous-vecteur V[1..i] sont inférieurs ou égaux à l'élément V[i+1].
On a donc :

V[1..i] non traité, V[1..i] <= V[i+1], V[i+1..N] Trié

On a deux cas :

- I = 1
  (V[1] non traité, V[1]<= V[2], V[2..N] trié) donc V[1..N] trié
  L'algorithme est terminé.
- I > 1
  Pour augmenter le sous-vecteur V[i+1..n] d'un élément, il suffit de chercher le plus grand élément contenu dans le sous-vecteur V[1..i] et de placer cet élément en position i.

Schéma de l'algorithme

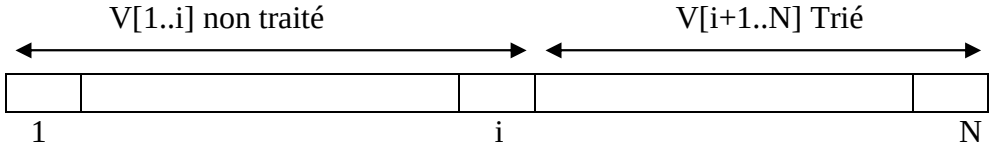
```
ALGORITHME SLECTION_PERMUTATION
```

```
CONST Bi = 1
      Bs = 10
VAR  V : Tableau[Bi..Bs] d'entier
     N, i, j : entier
DEBUT
{Chargement du vecteur V}
…
Pour i de N à 2 Faire
{Recherche de l'indice du maximum dans V[1..i]}
 indmax ← 1
 Pour j de 2 à i
 FAIRE
  Si V[indmax] < V[j] Alors
     indmax ← i
  FIN SI
FIN FAIRE
{Mettre le maximum relatif trouvé à sa place}
Si indmax <> i Alors
     Aux ← V[indmax]
     V[indmax] ← V[i]
     V[i] ← Aux
Fin Si
Fin Faire
```

VI.1 Tri par la méthode des bulles

Même principe que le précédent.

Après avoir traité n-i (1 <= i < N) éléments du vecteur.



On peut donc considérer le vecteur V comme la concaténation de deux sous-vecteurs : le sous-vecteur V[1..i] dont les éléments n'ont pas encore été triés, et le sous vecteur V[i+1..N] dont les éléments sont triés. D'autre part tous les éléments du sous-vecteur V[1..i] sont inférieurs ou égaux à l'élément V[i+1].

On a donc :

V[1..i] non traité, V[1..i] <= V[i+1], V[i+1..N] Trié

On a deux cas :

- I = 1

  (V[1] non traité, V[1]<= V[2], V[2..N] trié) donc V[1..N] trié

  L'algorithme est terminé.

- I > 1

Pour augmenter le sous-vecteur V[i+1..n] d'un élément, il suffit de chercher le plus grand élément contenu dans le sous-vecteur V[1..i] et de placer cet élément en position i.

On parcourt le sous-vecteur V[1..i] de gauche à droite et, chaque fois qu'il y a deux éléments consécutifs qui ne sont pas dans l'ordre, on les permute. Cette opération permet d'obtenir en fin du $i^{ième}$ parcours le plus grand élément placé en position i, et les éléments après cette position sont ordonnés.

Schéma de l'algorithme

```
ALGORITHME TRI_BULLE1
CONST N= 10
VAR  V : tableau[1..N] de réel
     N, i, j : entier
     AUX : réel
DEBUT
{Chargement du vecteur}
…
POUR i de N à 2 pas –1 FAIRE
     POUR j de 1 à i FAIRE
          SI V[j]>V[j+1] ALORS
            AUX ← V[j]
            V[j] ← V[j+1]
            V[j+1] ← AUX
          FIN SI
     FIN FAIRE
FIN FAIRE
FIN
```

Application

Exécuter à la main cet algorithme avec les vecteurs suivants :

2	2	-1	3	0	1

1	-1	2	5	13	15

Que remarquez-vous ?

3. Schéma de l'algorithme à bulle optimisé

```
ALGORITHME TRI_BULLE1
CONST N= 10
VAR  V : tableau[1..N] de réel
     N, i, j : entier
     AUX : réel
DEBUT
{Chargement du vecteur}
…
i ← N
atonpermuté ← vrai
TANT QUE (atonpermuté) FAIRE
     j←1
     atonpermuté ← faux
     TANT QUE (j < i) FAIRE
          DEBUT
           SI (V[J+1] < V[j]) ALORS
```

```
                AUX←V[J+1]
                V[J+1] ←V[J]
                V[J] ← AUX
              FIN SI
                atonpermuté←vrai
          FIN
          j←j+1
     FIN
     i←i-1
     FIN
FIN

POUR i de N à 2 pas –1 FAIRE
     POUR j de 1 à i FAIRE
          SI V[j]>V[j+1] ALORS
            AUX ← V[j]
            V[j] ← V[j+1]
            V[j+1] ← AUX
          FIN SI
     FIN FAIRE
FIN FAIRE
FIN
```

CHAPITRE VII. LA PROGRAMMATION PROCEDURALE

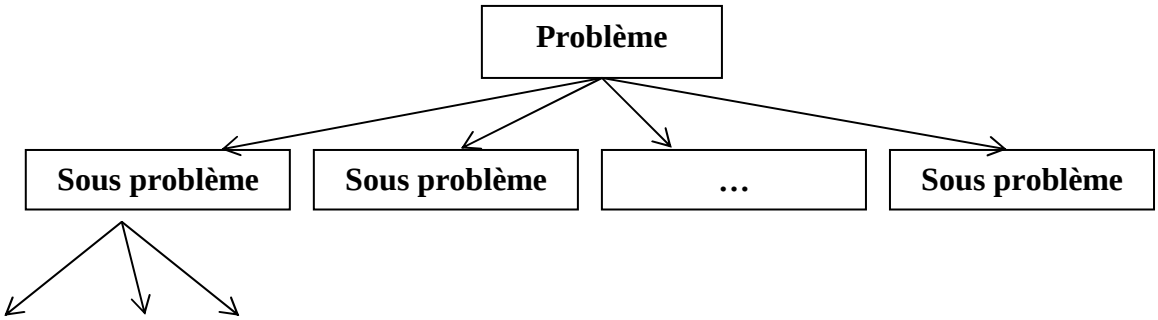
VII.1.Introduction

Informatiser une application, facturation de la consommation d'eau, par exemple, c'est faire réaliser par ordinateur, une tâche qui était réalisée par l'homme.
Les étapes de résolution d'un problème:

1. comprendre l'énoncé du problème.
2. décomposer le problème en sous problèmes plus simple à résoudre.
3. associer à chaque sous problème, une spécification (une description).
    - Les données nécessaires
    - Les données résultantes
    - La démarche à suivre pour arriver aux résultats en partant d'un ensemble de données.

VII.2.L'analyse modulaire

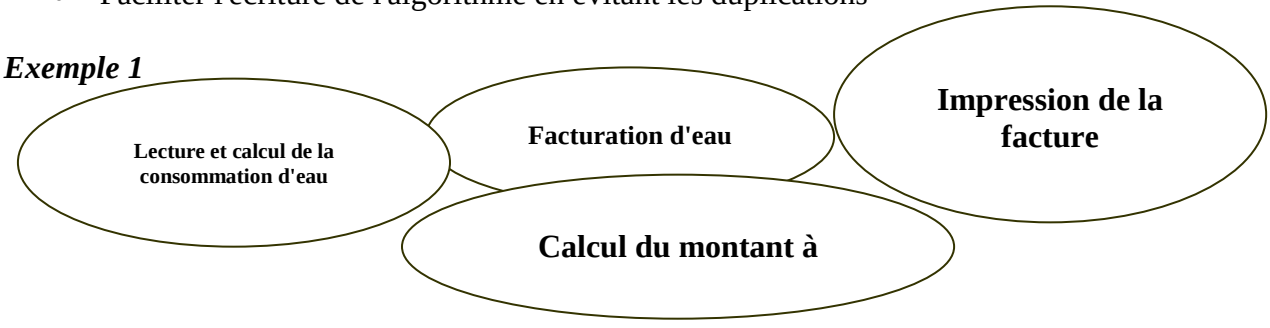
"Diviser les difficultés en autant de parcelles qu'il se peut afin de les mieux résoudre" Descartes
La conception d'un algorithme procède en général par des affinements successifs: on décompose le problème à résoudre en sous problèmes, puis ces derniers à leur tour, jusqu'à obtenir des problèmes "facile à résoudre", pour chaque sous problème on écrira un sous programme.
Ainsi la résolution du problème sera composée d'un algorithme principal et d'un certain nombre de sous programmes. L'algorithme principal a pour but d'organiser l'enchaînement des sous programmes.



L'intérêt de l'analyse modulaire est

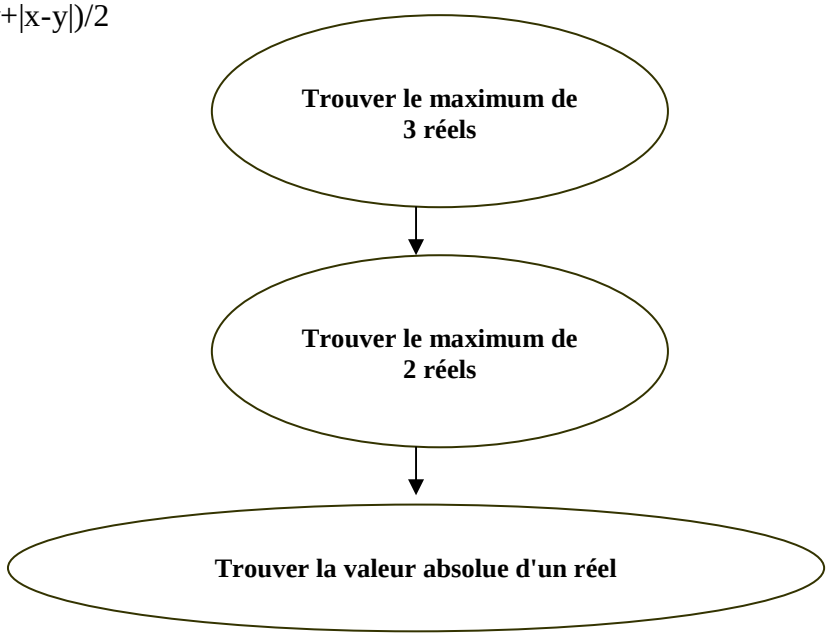
- Répartir les difficultés
- Faciliter la résolution d'un problème complexe
- Pouvoir poursuivre l'analyse comme si les sous problèmes étaient résolus
- Faciliter l'écriture de l'algorithme en évitant les duplications

Exemple 1



Exemple 2

Ecrire un algorithme qui détermine le maximum de 3 valeurs numériques en utilisant les formules suivantes:
Max (x,y,z) = max(max(x,y),z)
Max(x,y)=(x+y+|x-y|)/2



VII.3.Les sous programmes

C'est une unité fonctionnelle formée d'un bloc d'instructions, nommée et éventuellement paramétrée. Que l'on déclare afin de pouvoir l'appeler par son nom en affectant s'il y a lieu des valeurs à ses paramètres. Les données fournies au sous programme et les résultats produits par ce dernier sont appelés des arguments ou des paramètres.

Un sous programme peut être une procédure ou une fonction.

1. Procédure: c'est un sous programme contenant un certain nombre d'instructions et admettant **zéro ou plusieurs résultats**.
2. Fonction: c'est un sous programme contenant un certain nombre d'instructions et qui retourne **un résultat unique affecté à son nom**.

VII.4.Les fonctions

Définition d'une fonction

```
Fonction nom_fact (parmètres):type_resultat
        Déclaration des variables
        Début
                Instructions
        Fin
```

Exemple 1

Fonction valeur_absolue (x : réel):réel

```
Variables
a  réel
Début
si x > 0 alors
        a ← x
sinon
        a ← -x
fin si
valeur_absolue ← a
Fin
```

Exemple 2

```
Fonction moyenne (x : réel, y: réel):réel
        Début
                moyenne ← (x + y ) / 2
        Fin
```

Appel de fonction

L'appel de fonction est une expression, dont la valeur est le résultat retourné par cette fonction, son appel s'effectue donc comme si on va évaluer une expression.

Var ← nom_fact(paramètre effectifs)

Ou

Ecrire (nom_fact(paramètre effectifs)

Exemple

```
a ← valeur_absolue(-31.)
d ← -7.5
c ← valeur_absolue(d) - d * a
```

Remarque

Une fonction peut ne pas avoir de paramètre, par exemple

```
Fonction pi(): réel
Début
pi ← 22/7
Fin
```

Appel de pi:

```
air ← pi() * r * r
```

VII.5.Les procédures

C'est un sous programme qui est appelé avec zéro ou plusieurs paramètres et peut donner zéro ou plusieurs résultats.

Définition d'une procédure

Procédure nom_proc (paramètres)

Remarque

A la différence de la fonction, la procédure ne retourne pas de résultat.

Exemple

```
Procédure Affiche_somme ( a: entier, b:entier)
        S entier
         Début
        S ← a + b
        Ecrire (S)
        Fin
```

Appel de procédure

nom_proc(paramètres effectifs)

Exemple

```
Algorithme affiche
                Variables
        x entier
        y entier
        Début
        écrire ("donnez un entier")
        lire (x)
        écrire ("donnez un autre entier")
        lire (y)
        Affiche_somme(x,y)
        Fin
```

Passage de paramètres

- Paramètre fourni en donnée (transmission par copie de valeur) : Ce paramètre est fournie par le programme ou le sous programme appelant, sa valeur reste inchangée après l'exécution du sous programme appelé.

Un paramètre "donnée" est précédé par DON lors de la déclaration de la procédure.
Procédure essai1 (DON p1: réel)

- Paramètre fourni en résultat (transmission par référence) : Ce paramètre est retourné par la procédure appelée, l'exécution de la procédure appelée calcule une valeur qui lui sera affectée en fin de traitement.

Un paramètre "résultat" est précédé par RES.
Procédure essai2 (RES p2 : réel)

- Paramètre fourni en donnée_résultat (transmission par référence) : Ce paramètre est fourni par le sous programme ou le programme appelant, est modifié par la procédure appelée.

Un paramètre donnée_résultat est précédé par DONRES.
Procédure essai3 (DONRES p3: réel)

Remarque

Par défaut un paramètre est fourni en donné.
Procédure essai (p: réel)
p est passé en donné

Exemple

Algorithme appelant
Variables
a réel
b réel
c réel
Début
a ← 45.
essai1 (33.)
essai1 (a)
essai1 (35. * a -2.)
~~***essai1 (b)***~~ C'est incorrecte parce que b ne contient aucune valeur alors que le paramètre est fourni en donnée à la procédure essai1
essai2 (a)
essai2 (b)
~~***essai2 (a*4.)***~~ C'est incorrecte parce que a*4 est une expression numérique (donc une valeur) alors que le paramètre est fourni en résultat à la procédure essai2
~~***essai2 (35.)***~~ C'est incorrecte parce que 35 est une valeur alors que le paramètre est fourni en résultat à la procédure essai2
essai3 (a)
~~***essai3 (c)***~~ C'est incorrecte parce que c ne contient pas de valeur alors que le paramètre est fourni en donnée résultat à la procédure essai3
~~***essai3 (45.)***~~ C'est incorrecte parce que 45 est une valeur alors que le paramètre est fourni en donnée résultat à la procédure essai3
~~***essai3 (a*2.)***~~ C'est incorrecte parce que a*2 est une expression numérique (donc une valeur) alors que le paramètre est fourni en donnée résultat à la procédure essai3
Fin

VII.6.Conclusion

La possibilité pour un programmeur de définir ses propres fonctions et procédures, permet de décomposer un programme important en un certain nombre d'éléments plus petits. Tout programme peut être simplifié grâce à un usage intelligent des sous programmes.
L'approche cartésienne dans le développement des programmes présente plusieurs avantages. Nombreux sont par exemple les programmes dans les quels certaines séquences d'instructions sont appelées plusieurs fois depuis divers points d'appel. Il est intéressant de les isoler dans un sous programme qui puisse être appelé à tout moment si nécessaire. En plus, utiliser un sous programme permet de lui associer à chaque fois un jeu de données différent et, par conséquent, d'éviter de programmer des instructions redondantes.
Un autre apport important de l'approche cartésienne, est la clarté qu'elle confère aux programmes lorsque ceux ci sont découpés en sous programmes. Ce type de programme est plus facile à concevoir, à mettre au point et à maintenir. Ceci est particulièrement vrai dans le cas de problèmes longs et complexes.

TD 5 : Les fonctions

Exercice 1
Créer une fonction qui calcule le carré d'un nombre réel.
Exercice 2
Créer une fonction min2 qui donne le minimum de deux réels en utilisant la formule suivante:
Min(x,y)=(x+y-|x-y|)/2
Utiliser la fonction valeur_absolue déjà définie.
Exercice 3
Ecrire une fonction min3 qui retourne le minimum de 3 réels en utilisant la formule suivante:
Min(x,y,z)=Min(Min(x,y),z)
Utiliser la fonction min2
Exercice 4
Ecrire un programme C qui lit 3 réels au clavier et qui affiche le minimum, utiliser la fonction min3.
Exercice 5
Créer une fonction qui retourne le cube d'un réel. Utiliser la fonction carrée sachant que $x^3 = x^2\ x$
Exercice 6
Ecrire un programme C qui lit une valeur réelle au clavier et affiche x^2, x^3, x^5 et x^6 .
Utiliser les fonctions carrée et cube sachant que:
$x^5 = x^2\ x^3$
$x^6 = x^3\ x^3 = (x^2)^3 = (x^3)^2$
Exercice 7
Créer une fonction moyenne3 qui calcul la moyenne de 3 réels.
Exercice 8
Créer une fonction factorielle.
0!=1
n!=nx(n-1)x….x1
Exercice 9
Créer une fonction qui calcul le cardinale n p

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

Exercice 10
Ecrire une fonction qui permet de retourner $f(x)=4x^3+2x+9$, x étant un réel.
Exercice 11
Ecrire une fonction qui permet de retourner $f(x,y)= xy+5x^2-2y$ avec x et y deux réels.
Exercice 12
Ecrire une fonction qui calcul le PGCD de 2 entiers strictement positifs en appliquant l'algorithme d'Euclide.
PGCD(a,b)=b si b divise a
PGCD(a,b)=PGCD(b,r) avec r le reste de la division de a par b
Exercice 13
Ecrire une fonction qui retourne le PGCD de 3 entiers strictement positifs, en utilisant la formule suivante:
PGCD(a,b,c)=PGCD(PGCD(a,b),c)
Exercice 14
Ecrire une fonction qui retourne le PPCM de deux entiers strictement positifs, sachant que :

$$PPCM(a,b) = \frac{ab}{PGCD(a,b)}$$

Exercice 15
Ecrire une fonction qui retourne le PPCM de 3 entiers strictement positifs, en utilisant la formule suivante:
PPCM(a,b,c)=PPCM(PPCM(a,b),c)
Exercice 16
Ecrire une fonction qui détermine si oui ou non deux entiers strictement positifs sont premiers entre eux.
x et y sont premiers entre eux si et seulement si PGCD(x,y)=1
Exercice 17
Ecrire un programme C qui lit 2 entiers strictement positifs au clavier et qui affiche:

1. Le PGCD
2. Le PPCM
3. Si les 2 entiers sont premiers entre eux ou non

Exercice 18
Ecrire un programme C qui affiche les entiers parfaits se trouvant entre 2 valeurs m et n entiers lus au clavier telle que 2 < m < n
Exercice 19
On se propose d'écrire une fonction qui calcule (exponentiel) à base des logarithmes népériens Telle que log(e) = 1. On pourra choisir n assez grande et approcher la valeur de e.

$$e = \sum_{k=0}^{n} \frac{1}{k!}$$

Exercice 20
Ecrire une fonction qui calcul e^x pour x un entier strictement positif.
Exercice 21
Ecrire une fonction qui retourne la position du minimum dans un tableau de réels de borne inf et de borne sup.
Exercice 22
Réécrire l'algorithme du tri par sélection en utilisant la fonction précédente.

```
Algorithme tri_sélection
Variables
min entier
ind entier
i entier
temp entier
Début
T tableau [1..14] de entier
Pour ind de 1 à 13 faire
        min ←ind
        pour i ind + 1 à 14 faire
                si T[i] < T[min] alors
                        min ← i
                fin si
        fin pour
        temp ← T[ind]
```

T[ind] ← T[min]
T[min] ← temp
Fin pour
Fin

Exercice 23

Ecrire une fonction qui vérifie si un tableau de réel de borne inf et sup est trié.

Exercice 24

Ecrire une fonction qui vérifie si une valeur réelle se trouve dans un tableau de réels de borne inf et sup.

Exercice 25

Ecrire une fonction qui retourne si oui ou non une chaîne de caractères est composé de caractères autres que les lettres.

Exercice 26

Ecrire une fonction qui retourne la dernière position d'un caractère dans une chaîne, il retourne 0 si ce caractère n'existe pas dans la chaîne.

Exercice 27

Ecrire un programme C qui lit une chaîne de caractères et affiche le nombre d'occurrence de chaque caractère dans cette chaîne.

Exercice 28

Ecrire une procédure qui calcule la somme et le produit de deux réels.

Exercice 29

Exécuter cet algorithme:

Algorithme p_essai
Variables
z réel
n réel
x réel
p réel
Début
z ← 0.
n ← 5.2
x ← -3.
som_prod(n,x,z,p)
écrire (n,x,z,p)
som_prod(p,z,n,x)
écrire (n,x,z,p)
Fin

Avec som_prod la procédure de l'exercice précédent

Exercice 30

Ecrire une procédure qui simplifie une équation dans 9^2 ax + by=c
Si PGCD(a,b) divise c

$$\frac{a}{PGCD(a,b)}x+\frac{b}{PGCD(a,b)}y=\frac{c}{PGCD(a,b)}$$

Exercice 31

Ecrire une procédure qui permute le contenu de deux variables réels.

Exercice 32

Ecrire un programme C qui lit deux valeurs réelles au clavier et les affiche puis permute ces deux réels et réaffiche les deux valeurs.

Exercice 33

Soit une procédure son qui génère un son de fréquence f:
Procédure son (DON f: entier)
Ecrire un programme C qui permet de simuler le fonctionnement d'un instrument de musique : l'appui sur les touches A, Z, E, R, T, Y, U, I devant produire un son correspondant à l'une des notes musicales Do, Ré, Mi, Fa, Sol, La, Si, Do les fréquences correspondants aux notes sont: 500, 561, 630, 667, 749, 841, 944 et 1000.
L'algorithme s'arrête si l'utilisateur appui sur une autre touche.

Exercice 34

Ecrire une procédure qui permet d'afficher les éléments d'un tableau de réels de borne inf et sup.

Exercice 35

Ecrire une fonction fréquence qui détermine le nombre d'apparition d'une valeur dans un tableau de réels de borne inf et sup.

Exercice 36

Ecrire une fonction qui renvoie la somme des éléments d'un tableau de réels de borne inf et sup.

Exercice 37

Ecrire une procédure qui copie les valeurs d'un tableau d'entiers de taille n dans un autre tableau.

Exercice 38

Ecrire une procédure qui éclate un tableau de n réels en deux tableaux de réels positifs et négatifs.

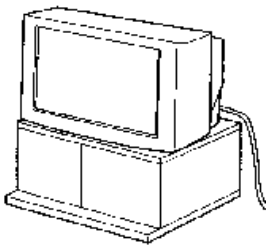
CHAPITRE VIII. LA PROGRAMMATION ORIENTE OBJET

VIII.1.Introduction

La POO correspond à une méthodologie qui permet d'aborder autrement les problèmes, cela dés la phase de l'analyse. Avec cette méthode, le problème n'est pas posé en termes de fonctionnalités abstraites devant être ensuite successivement raffinées. Ici, on part des entités manipulées dans la réalité. Ces entités, les **objets** ont des propriétés bien précises , lesquelles devront être définies aussi soigneusement que possible lors de l'analyse puis implémentées dans un langage de programmation orienté objet.

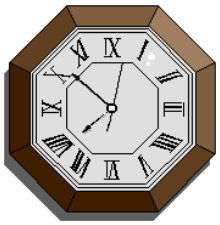
VIII.2.L'objet

Chaque objet possède sa propre identité. C'est à dire deux objets sont distincts même si tous les attributs ont des valeurs identiques.



Télévision Sony grise Trinitron Multi système

Clé jaune de la porte principale de la maison de Mr Ben Fedhil



Horloge marron

Un objet a une structure de données, **attributs** et un comportement, **opérations**

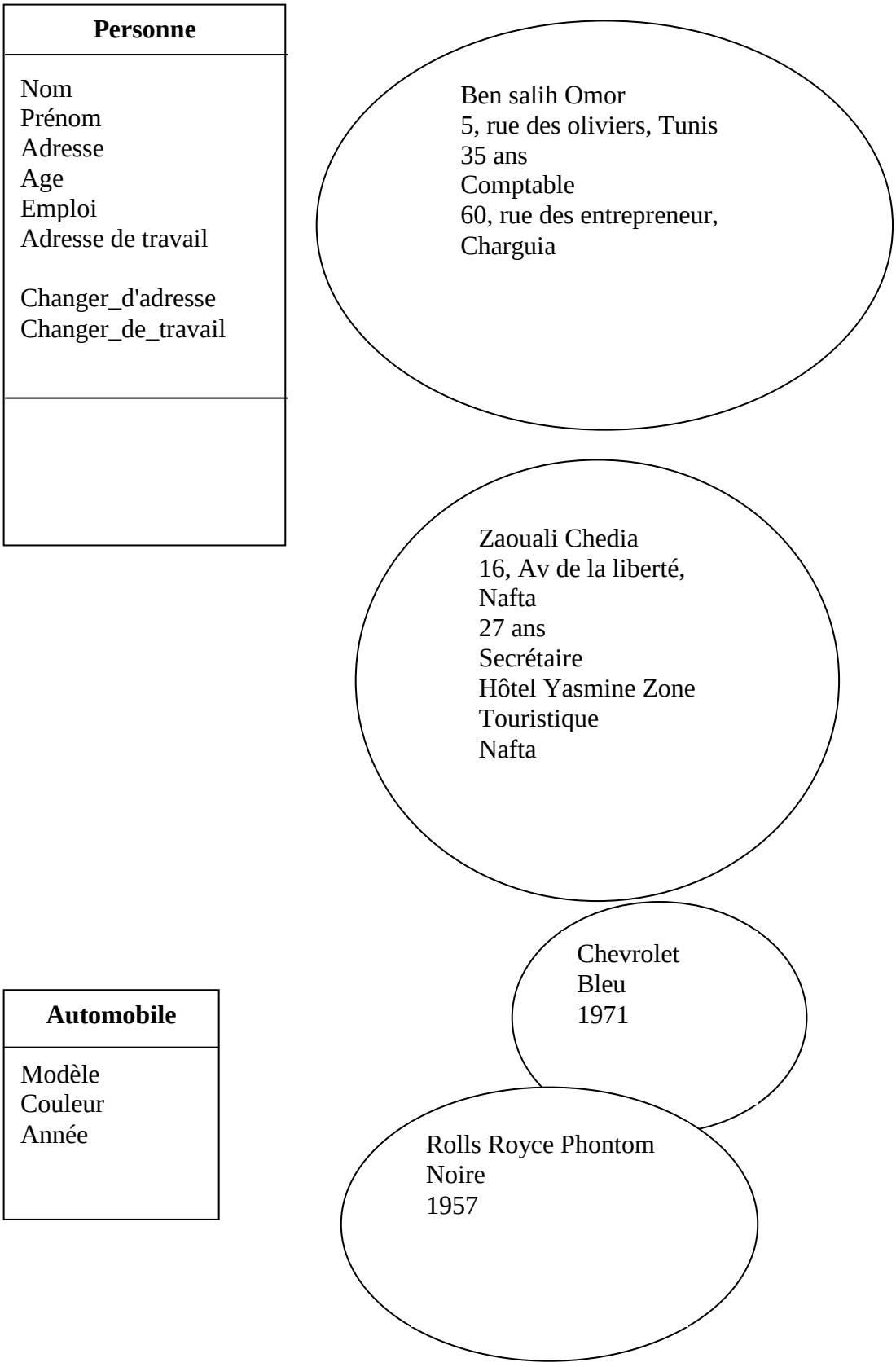
Exercice

Tous les objets possèdent une identité et se distinguent les uns des autres. Décrivez comment on pourrait distinguer les uns des autres les objets suivants:

Les objets	Critères de distinction
Tous les habitants du monde pour leur adresser du courrier	
Tous les habitants du monde pour les besoins d'une enquête criminelle	
Tous les clients ayant un coffre dans une banque donnée	
Tous les téléphones du monde pour passer des appels téléphoniques	
Toutes les adresses des courriers électroniques dans le monde	

VIII.3.Les classes

Les objets ayant même attributs et même opérations appartient à une classe. Chaque objet est dit comme étant une instance d'une classe. Une classe est donc, une définition, une moule qui permet de créer de nouveaux objets.


Personne
Nom
Prénom
Adresse
Age
Emploi
Adresse de travail
Changer_d'adresse
Changer_de_travail
Ben salih Omor
5, rue des oliviers, Tunis
35 ans
Comptable
60, rue des entrepreneur,
Charguia
Zaouali Chedia
16, Av de la liberté,
Nafta
27 ans
Secrétaire
Hôtel Yasmine Zone
Touristique
Nafta
Automobile
Modèle
Couleur
Année
Chevrolet
Bleu
1971
Rolls Royce Phontom
Noire
1957

VIII.4.L'encapsulation

Ou masquage d'information, elle consiste à séparer les aspects externes d'un objet accessibles pour les autres objets qu'on désigne par **publique**, des détails d'implémentation interne, rendues invisibles aux autres objets qu'on désigne par **privé**.
Ainsi l'implémentation d'un objet peut être modifiée sans affecter les applications qui emploient cet objet.

Exemple

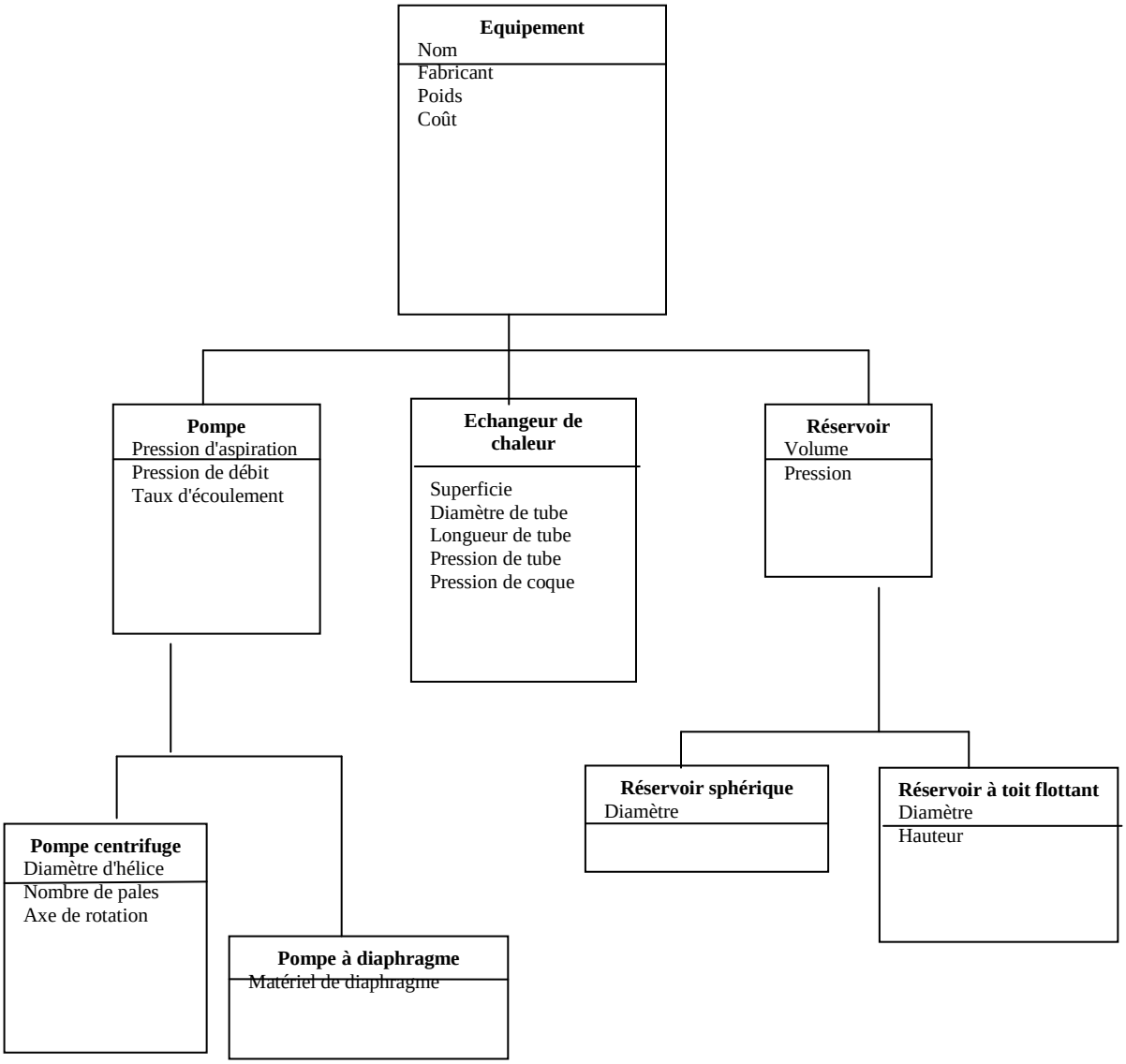
```
Classe Cpersonne
Publique Nom chaîne[40]
Publique Prénom chaîne[40]
Privé  Adresse chaîne[60]
Publique  Age entier
Publique procédure Changer_adresse (nouv_adresse: chaîne)
        Début
        Adresse ← nouv_adresse
        Fin
Fin Cpersonne
```

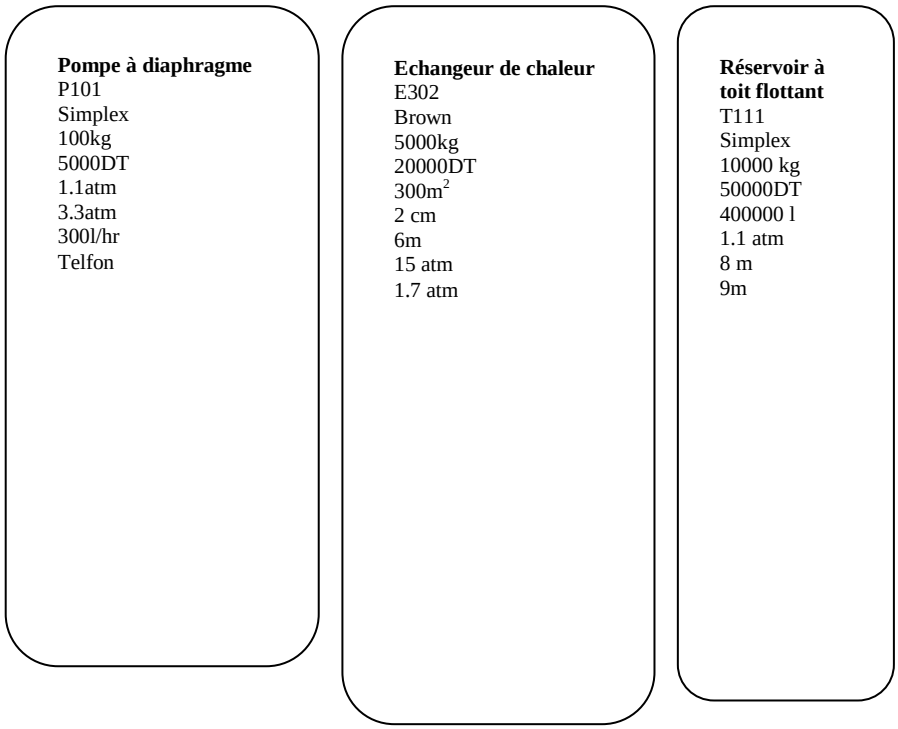
Nous supposons qu'un programme **P** utilise des objets de la classe Cpersonne il ne peut utiliser que les attributs: Nom, Prénom, Age et l'opération Changer_adresse soit:

```
Algorithme P
Début
        Cpersonne Per
        Per.Nom ←"Ben Abdallah"
        Per.Prénom←"Mohamed"
        Per.Age←25
        Per.Changer_Adresse("5, rue des fleures Tunis")
Fin
```

VIII.5.L'héritage

C'est le partage des attributs et des opérations entre des classes s'appuyant sur une relation hiérarchique. Une classe peut être définie à grands traits et ensuite raffinée dans des sous classes de plus en plus fins. Chaque sous classe hérite toutes les propriétés de sa super classe et y ajoute ses propres et uniques propriétés.

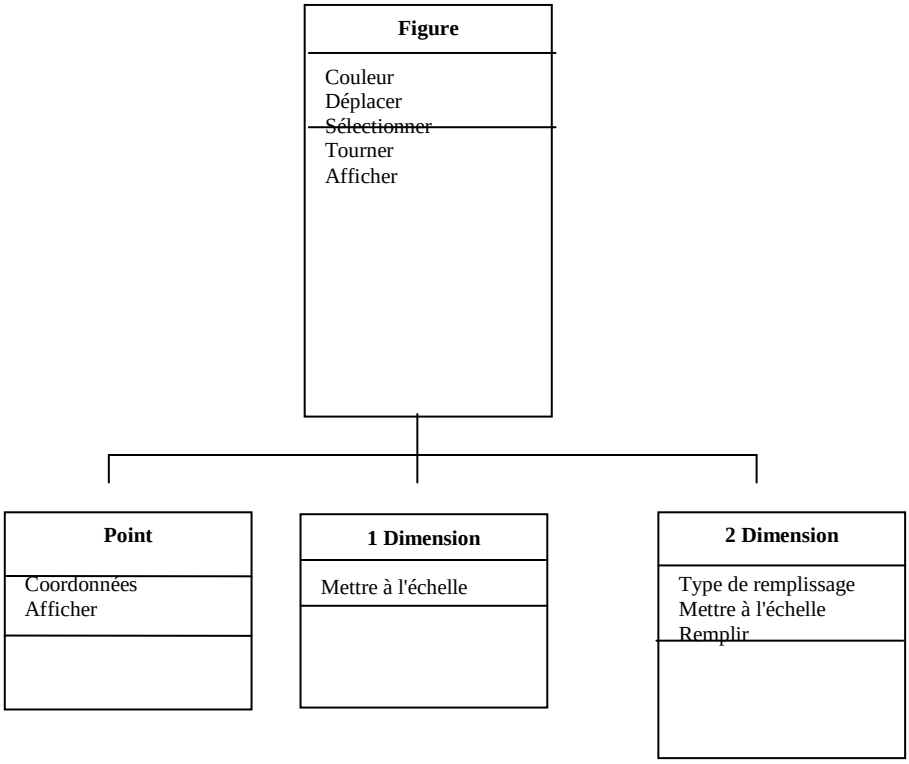

Equipement
Nom
Fabricant
Poids
Coût
Pompe
Pression d'aspiration
Pression de débit
Taux d'écoulement
Echangeur de chaleur
Superficie
Diamètre de tube
Longueur de tube
Pression de tube
Pression de coque
Réservoir
Volume
Pression
Pompe centrifuge
Diamètre d'hélice
Nombre de pales
Axe de rotation
Pompe à diaphragme
Matériel de diaphragme
Réservoir sphérique
Diamètre
Réservoir à toit flottant
Diamètre
Hauteur


Pompe à diaphragme
P101
Simplex
100kg
5000DT
1.1atm
3.3atm
300l/hr
Telfon
Echangeur de chaleur
E302
Brown
5000kg
20000DT
300m2
2 cm
6m
15 atm
1.7 atm
Réservoir à toit flottant
T111
Simplex
10000 kg
50000DT
400000 l
1.1 atm
8 m
9m

VIII.6.Le polymorphisme

Signifie que la même opération peut se comporter différemment sur différents classes. L'opération déplacer, par exemple, peut se comporter différemment sur les classes fenêtre et pièce de jeu d'échecs.

Dans le monde réel, chaque objet "sait comment" effectuer ses propres opérations. Cependant, dans un langage de programmation orienté objet, le langage sélectionne automatiquement l'opération correcte en se basant sur le nom de l'opération et sur la classe de l'objet sur lequel elle opère.



CHAPITRE IX. INITIATION AU LANGAGE JAVA

IX.1 Historique

Java est un langage objet ressemblant au langage C++. Il a été mis au point en 1991par la firme Sun Microsystem. Le but de Java à l'époque était de constituer un langage de programmation pouvant être intégré des appareils électroménagers, afin de pouvoir les contrôler, de les rendre interactifs, et surtout de permettre une communication entre les appareils. Ce programme développement se situait dans un projet appelé Green, visant à créer une télécommande universelle (Star 7) comprenant un OS capable de gérer l'ensemble des appareils électroménagers de la maison. Etant donné que le langage C++ comportait trop de difficulté, James Gosling, un des acteurs du projet (considéré comme le père du Java) décida de créer un langage orienté objet reprenant des caractéristiques principales du C++, en éliminant ses points difficiles, et en le rendant moins encombrant et plus portable (il devait pouvoir intégré n'importe quel appareil..) Ainsi ce langage fut baptisé dans un premier temps Oak (Oak signifiant chêne). Toutefois, puisque ce nom était déjà utilisé, il fut rebaptisé Java en l'honneur de la boisson préférée des développeurs, c.à.d. le café, dans la production provient de l'île de Java.

IX.2 Fonctionnement de JAVA

Un programme Java doit être :
Compilé: le résultat est un nouveau fichier au format "ByteCode"
Interprété: grâce à une JVM (Java Virtual Machine)
Remarques : Le fichier source d'un programme écrit en Java est un fichier simple texte dont l'extension est par convention ".java"
Ce fichier source doit être un fichier texte non formaté, c.à.d. un fichier texte dans sa plus simple expression, sans mise en forme particulière ou caractères spéciaux, c.à.d. qu'il contient uniquement les caractères ASCII ou EBCDIC de base.
Lorsque le programme est prêt à être essayé, il s'agit de "compiler" (le traduire en langage machine) à l'aide d'un compilateur). Toutefois, contrairement aux langages compilés traditionnels, pour lesquels le compilateur créer un fichier binaire directement exécutable par un processeur données), le langage Java est compilé en un langage intermédiaire (appelé bytecode) dans un fichier portant le même nom que le fichier source à l'exception de son extension ".class"
Cette caractéristique est majeure, car c'est elle qui fait qu'un programme écrit en Java est portable, c.à.d. qu'il ne dépend pas d'une plate-forme donnée. En réalité le code intermédiaire n'est exécutable sur aucune plate-forme sans la présence d'une machine virtuelle, un interpréteur (la machine virtuelle est d'ailleurs parfois appelée interpréteur Java) tournant sur une plate-forme donnée, et capable d'interpréter le code intermédiaire.
JVM : Pour peu qu'une plate-forme (Win, Unix, Linux,..) possède une JVM fonctionnant sur son système, celle-ci est capable d'exécuter n'importe quelle application Java!
JRE/JDK : Le JRE (Java Runtime Environment) est:

- Un fichier permettant d'interpréter des applications Java.
- Plusieurs centaines de classes (API)

IX.3 Syntaxe de définition de classe

Tout code java doit se trouver dans une classe. Hormis les définitions de classe, de méthode, et de boucle, toute instruction se finit par ";". Le corps d'une classe est délimité par un bloc (les accolades {})
Le fichier Java (.java):

- doit porter le nom (majuscule/minuscule) de l'unique classe (publique) qu'il contient.
- peut contenir d'autres classes non-publiques.

Remarque:

- Ce sont les accolades qui délimitent la portée de la classe.
- Il est possible de créer des classes publiques, et non publiques: la différence est importante, nous l'a verrons dans l'unité liée aux packages.
- Si on crée une classe dans un fichier, ce fichier doit porter exactement le même nom que la classe: le cas échéant, le compilateur refusera de compiler le fichier, en apportant bien sur un justificatif de son refus.
- Si on crée plusieurs classes dans un même fichier, c'est l'unique classe définie avec le mot réservé "public" qui donnera le nom du fichier.

IX.4 Déclaration, initialisation d'attributs, de variables locales, de constantes

Le bloc délimité par les accolades définit la portée des variables.
Remarque:

- Ce sont les accolades qui délimitent la portée de la classe.
- Un attribut sera "accessible" durant toute la durée de vie de l'objet dans lequel il se trouve.
- Un attribut PEUT être initialisé avant d'être utilisé.
- On indiquera un critère de visibilité pour les attributs: on sélectionnera en général "private"
- Une variable locale n'a d'existence que durant, l'exécution de la méthode/bloc dans laquelle elle se trouve.
- Une variable locale ne peut recevoir de critères de visibilité (private,public,...): elle est de toute façon accessible uniquement dans le bloc dans laquelle elle est définie.
- Une variable locale DOIT être initialisée avant d'être utilisée, un attribut reçoit une initialisation par défaut.

IX.5 Type de bases

Chaque variable DOIT être d'un type donné. Un type de donnée détermine l'intervalle de valeur susceptible d'être contenue. Dans les entiers, le type "int" est le type par défaut, pour les décimaux c'est le type "double".
ATTENTION!! Les types de base sont en minuscule. A ne pas confondre avec les "Wrappers" (classe ayant le même nom que les types de base): Ex: byte vs Byte
Entier:

byte: 8 bits signés
short: 16 bits signés
int: 32 bits signés
long: 64 bits signés

Decimal:

float: 32 bits signés
double: 64 bits signés

Autre:

char: 16 bits UNICODE2
boolean: 1 bit, 2 valeurs possibles true or false

IX.6 Impressions sur console

System.out.println("Bonjour\nVous")
System.out.print("Bonjour\nToi")

IX.7 Conversion/initialisation des variables numériques

- Les constantes
Pour définir une constante, il vous suffit de rajouter "final" à la déclaration de l'attribut.
On n'oubliera pas d'utiliser la convention de nommage!
public final int MAX = 10;
int variable = MAX; // ok
MAX = 5; //erreur
- Initialisation des attributs
Les attributs reçoivent une initialisation par défaut
Types numériques : 0
char: "\u0000"
boolean: false
Objet: null
Les variables locales nécessitent une initialisation explicite avant usage.

IX.8 Opérateurs d'assignements

op1 += op2 op1 = op1 + op2 Addition
op1 -= op2 op1 = op1 - op2 Soustraction
op1 *= op2 op1 = op1 * op2 Multiplication
op1 /= op2 op1 = op1 / op2 Division
op1 %= op2 op1 = op1 % op2 Modulo
Incrémentation/Décrémentation en préfixe/suffixe:
int i = 3, j=4;
++i;
i++; // égal à i = i + 1, soit 4
--i;
i--; // égal à i = i -1, soit 3
j=i++; // (i=4,j=3: on copie le contenu de i dans j, puis on incrémente i
j=++i; //(i=5,j=5: on incrémente i, puis on copie le contenu de i dans j.
Exemple : i = 3 ; j=4 ; int k = --j * i ++ <-----> i = 4 , j=3 , k=9

IX.9 Opérateurs conditionnels

> op1 > op2 op1 est strictement supérieur à op2
>= op1 >= op2 op1 est supérieur ou égal à op2
< op1 < op2 op1 est strictement inférieur à op2
<= op1 <= op2 op1 est inférieur ou égal à op2
== op1 == op2 op1 est égal à op2
!= op1 != op2 op1 est différent de op2
&& op1 && op2 op1 et op2 sont égal à true. Evaluation de op2 éventuel.
|| op1 || op2 op1 et op2 sont égal à true. Evaluation de op2 éventuel.

! !op1 op1 est false

IX.10 Structures alternatives

- if...else... ; l'élément conditionnel doit ramener true ou false

```
boolean b=false;
if (b=true) System.out.println("bonjour");
else System.out.println("au revoir");
```

- L'instruction switch

```
int i=3;
switch (i) // Uniquement int, char, byte, short
{
case 1:
System.out.println("un");
break;
case 2: case 3:
System.out.println("deux ou trois");
break;
case MAX:
System.out.println("maximum");
default:
System.out.println("les autres cas");
}
```

IX.11 Boucles

- while et do...while

La principale différence est que dans un do...while il passe les instructions une fois au moins avant la condition.

- For

```
for (int i=0 ; i<10 ; i++) // (index ; tant que la condition retourne true; incrémentation)
```

- continue et break

IX.12 Commentaires

Il y a 3 types de commentaires: simple ligne, bloc, Javadoc
Simple : // Simple ligne
Bloc : /* */
Javadoc: destiné à générer automatiquement une documentation sur votre code sous DOS : javadoc.exe Test.java
Commentaire javadoc: avant une méthode c'est un commentaire de méthode, avant un attribut c'est un commentaire d'attribut, avant une classe c'est un commentaire de classe.

Références

1. M. Baccouche, Cours Algorithmique, Iset de Radès, 2000.
2. S. Hriz, Cours Algorithmique, Iset de Sfax, 2000
3. J. Hubbard, Programmation Java, EdiScience, 2004
4. C. Ben Othman Zribi, Initiation à la programmation orientée objet en Java, CPU Tunis, 2003
5. P. Cégielski, Initiation au langage Java, Université Paris Est Créteil, Mai 2010
6. S.Tahé , Apprentissage du langage JAVA, - ISTIA - Université d'Angers, juin 2002
7. D. Lapoire, Initiation à l'algorithmique, Enseirb, France,octobre 2006
8. C. Cormen, Introduction à l'algorithmique, Dunod