**Djamel Eddine Z E G O U R**

# Apprendre et enseigner l'algorithmique

**Tome 1 : Cours et annexes.**

*Institut National d'Informatique*

# P r é f a c e

*Introduction*

Ce livre est le fruit d'une vingtaine d'années d'expérience dans le domaine de l'algorithmique et de la programmation. C'est le cours tel qu'il est assuré à l'Institut National d'Informatique d'Alger (Ex C.E.R.I) pour les étudiants de première année du cycle "ingénieur d'état en informatique".

Il constitue un support de cours pour des étudiants n'ayant aucune connaissance en programmation. Il est aussi destiné à des étudiants ayant déjà une première expérience en programmation et qui veulent connaître davantage sur l'art de la programmation.

*Objectif du cours*

Le cours couvre quatre grands aspects très liés : algorithmique, donnée, méthodologie et programmation. Dans une première étape, on s'intéresse essentiellement au formalisme algorithmique, ensemble de conventions permettant d'exprimer des solutions. On opère alors sur des objets simples et on développe des algorithmes sur la machine de Turing. Quant à la seconde étape, elle est totalement consacrée à l'étude des structures de données élémentaires à savoir les vecteurs, les listes linéaires et les fichiers . Un langage de programmation pédagogique est utilisé afin de s'initier à la programmation (PASCAL). Une méthode de recherche d'algorithmes, l'analyse descendante, est abordée avec tous les concepts qui s'y rattachent.

**Contenu**

La première partie renferme le cours d'algorithmique. Les concepts de base sont donnés en montrant la construction de programmes simples depuis leur expression en langage naturel avec ses inconvénients à leur expression entièrement dans un langage algorithmique. Une multitude d'algorithmes sont développés sur la machine de Turing afin de se familiariser avec le langage algorithmique. Une introduction aux fichiers et particulièrement aux structures de fichiers est donnée avec de nombreux programmes. On y trouvera essentiellement, les programmes de création, de maintenance et de tri de fichiers. Une méthode de conception d'algorithmes qu'est l'analyse descendante est exposée et illustrée en PASCAL en présentant tous les concepts qui s'y rattachent tels que la notion de portée, de communication entre modules, paramètres formels et réels, objet locaux et globaux, etc.

La seconde partie fournit un recueil de sujets d'examens. Pour chaque sujet, il est spécifié l'ensemble des cours à connaître.

La partie 3 fournit des corrigés types des sujets présentés dans la partie 2.

La partie 4 présente un ensemble d'exercices de programmation corrigés. Pour chaque programme, nous avons présenté les données et les résultats parfois détaillés dans le but de montrer leur conformité.

La partie 5 présente une série d'annexes très utiles pour un environnement de programmation. L'annexe 1 résume le langage algorithmique utilisé. Les annexes 2 et 3 donnent des

compléments d'informations sur quelques notions élémentaires et sur les disques. L'annexe 4 résume les principales fonctions DOS utiles pour toute utilisation de micro-ordinateur. Les annexes 5 et 6 fournissent des moyens, d'une part, pour représenter schématiquement des algorithmes (organigrammes) et, d'autre part, pour les traduire dans des langages de bas niveau (langage d'assemblage par exemple). Dans l'annexe 7, on trouvera une façon de rédiger un dossier de programmation.  Enfin, nous avons terminé par l'annexe 8 par un ensemble de conseils pratiques sous forme de proverbes utiles pour tout programmeur.

## Remerciements

Nous exprimons nos remerciements les plus chaleureux à notre collègue W.K Hidouci  pour ses conseils et surtout son efficacité dans la lecture approfondie de certaines parties de ce manuscrit.

Professeur Djamel Eddine  ZEGOUR

# Organisation du livre

***Tome1***

***Partie 1***
***. Cours d'algorithmique***

***Partie 2 : Annexes***
***1. Langage algorithmique***
***2. Notions élémentaires***
***3. Les Disques***
***4. Système d'exploitation MS-DOS : aide mémoire***
***5. Organigrammes***
***6. Traduction des algorithmes vers les langages de bas niveau***
***7. Rapport de programmation***
***8. Proverbes de programmation***

***Tome2***

***Partie 1***
***. Enoncés de sujets d'examens***

***Partie 2***
***. Corrigés de sujets d'examens***

***Partie 3***
***. Exercices programmés en PASCAL***

***Partie 4 : Annexes***
***1. Langage algorithmique***
***2. Rappel PASCAL***
***3. Rapport de programmation***
***4. Proverbes de programmation***

# S O M M A I R E

## *I. Cours d'algorithmique*

# *II. Annexes*

**Partie 1.**

# Concepts de base de l'algorithmique

## COURS 1. Une introduction à l'algorithmique

***Objectifs** : situer le mot " algorithme" dans les différentes étapes de la mise en oeuvre d'une application, puis en donner quelques définitions avant d'énoncer les principales propriétés que doit satisfaire un algorithme*

### 1.1 Introduction

L'utilisation d'un ordinateur pour la résolution d'une application (travail que l'on soumet à un ordinateur) nécessite tout un travail de préparation. N'ayant aucune capacité d'invention, l'ordinateur ne peut en effet qu'exécuter les ordres qui lui sont fournis par l'intermédiaire d'un programme. Ce dernier doit donc être établi de manière à envisager toutes les éventualités d'un traitement.

### 1.2 Mise en oeuvre d'une application

Plusieurs étapes sont nécessaires pour mettre en oeuvre une application :

**Etape 1 : Définition du problème**

Il s'agit de déterminer toutes les informations disponibles et la forme des résultats désirés.

**Etape 2 : Analyse du problème**

Elle consiste à trouver le moyen de passer des données aux résultats. Dans certains cas, on peut être amené à faire une étude théorique. Le résultat de l'étape d'analyse est un algorithme. Une première définition d'un algorithme peut être la suivante :

" On appelle algorithme une suite finie d'instructions indiquant de façon unique l'ordre dans lequel doit être effectué un ensemble d'opérations pour résoudre tous les problèmes d'un type donné"

Sachez aussi qu'il existe des problèmes pour lesquels on ne peut trouver une solution et par conséquent il est impossible de donner l'algorithme correspondant.

**Etape 3 : Ecriture d'un algorithme avec un langage de description algorithmique**

Une fois qu'on trouve le moyen de passer des données aux résultats, il faut être capable de rédiger une solution claire et non ambiguë. Comme il est impossible de le faire en langage naturel pour les raisons que nous donnerons par la suite, l'existence d'un langage algorithmique s'impose.

**Etape 4 : Traduction de l'algorithme dans un langage de programmation**

Les étapes 1, 2 et 3 se font sans le recours de la machine. Si on veut rendre l'algorithme pratique, il faudrait le traduire dans un langage de programmation. Nous dirons alors qu'un programme est un algorithme exprimé dans un langage de programmation.

**Etape 5 : Mise au point du programme**

Quand on soumet le programme à la machine, on peut être confronté à deux types de problèmes :

- la machine corrige l'orthographe, c'est ce qu'on appellera syntaxe dans le jargon de la programmation.
- la machine traduit le sens exprimé par le programme.

Si les résultats obtenus sont ceux attendus, la mise au point du programme se termine.

Si nous n'obtenons pas de résultats, on dira que qu'il y a existence des erreurs de logique. Le programme peut nous répondre comme suit :
- aucun résultat
- des résultats inattendus
- une partie des résultats

Dans ce cas, il faut revoir en priorité si l'algorithme a été bien traduit, ou encore est-ce qu'il y a eu une bonne analyse.

Ce dernier cas est considéré comme grave car il faut tout refaire, c'est à dire la révision de l'étape d'analyse pour la réécriture de l'algorithme ainsi que sa traduction.

## 1. 3 Exemples d'algorithmes

**Exemple 1**

L'algorithme de résolution de l'équation ax2 + bx + c = 0 dans l'ensemble des réels est le suivant:

*1. Calcul du discriminant, soit Delta*
*2. Si Delta > 0 alors il y a deux solutions données par les formules*
*x1 = -b + racine carrée (Delta) / 4 a . c*
*x2 = -b - racine carrée (Delta) / 4 a. c*
*3. Si Delta = 0, alors il y a une racine double donnée par la formule*
*x = - b / 2 a*
*4. Si Delta < 0, alors il n y a pas de solution.*

**Exemple 2**

Le principe de détermination de la racine cubique d'un nombre positif A est le suivant :

*1. Donner une valeur arbitraire à X0.*
*2. Calculer les valeurs de X1, X2, .....Xn par la formule de récurrence*
*Xn+1 = ( 2 Xn + A/ Xn2 ) / 3*
*3. S'arrêter dés que la différence entre deux termes consécutifs devient inférieure à une précision donnée.*

**Autres exemples en bref**

- Recette de cuisine pour faire un gâteau
- Traçage de fonctions
- Ouverture d'un coffre

### 1. 4 Quelques définitions du mot 'algorithme'

On peut définir un algorithme comme :

- le résultat dune démarche logique de résolution d'un problème,

- un procédé de calcul, qui permet à partir de données numériques et en suivant un plan de calcul très précis, d'obtenir le résultat d'un calcul,

- le résultat de l'étape d'analyse.

### 1. 5 Propriétés

On peut énoncer les cinq propriétés suivantes que doit satisfaire un algorithme :

Généralité : un algorithme doit toujours être conçu de manière à envisager toutes les éventualités d'un traitement.

Finitude : un algorithme doit s'arrêter au bout d'un temps fini.

Définitude : toutes les  opérations d'un algorithme doivent être définies sans ambiguïté.

Répétitivité : généralement, un algorithme contient plusieurs itérations, c'est à dire des actions qui se répètent plusieurs fois.

Efficacité : idéalement, un algorithme doit être conçu de telle sorte qu'il se déroule en un temps minimal et qu'il consomme un minimum de ressources.

# COURS 2. Inconvénients du langage naturel

***Objectifs :*** *sur deux exemples, montrer les inconvénients des langages naturels, pour s'en convaincre de la nécessité d'un formalisme algorithmique pour la rédaction des solutions.*

## 2.1 Exemple 1

Si nous demanderons à n élèves d'exprimer en **langage naturel** la résolution de l'équation du second degré $ax^2+bx+c= 0$, il est clair que nous aurons n solutions différentes. Chaque élève utilise son propre vocabulaire, ses propres nominations, ses propres symboles.

Une solution possible serait :

1. *A, B, C <---- les données*
2. *Calculer le discriminant*
3. *Cas où il est > 0 : x1, x2 = -B +- Rac(Delta) / 4 A.C*
4. *Cas ou il est nul : x1 = x2 = -B / 2A*

Seul le rédacteur connaît avec exactitude l'interprétation des symboles et même de certaines tournures qu'il utilise.

Essayons de recenser quelques problèmes de la solution envisagée .

Le signe <---- veut dire implicitement pour le rédacteur introduire les données dans A, B et C. Pour le rédacteur, le calcul du discriminant est évident. Par contre, c'est un mot étrange, qui ne veut rien dire, pour quelqu'un qui ne sait comment résoudre une solution du second degré.

On peut faire plusieurs remarques sur la ligne 3 :
Pour le rédacteur le mot "il" se rapporte au discriminant.
Pour le rédacteur l'écriture "x1, x2 = = -B +- Rac(Delta) / 4 A.C" veut dire qu'il existe deux solutions x1 et x2 et que l'on doit prendre alternativement le signe + ensuite - par son écriture +-. D'autres part, le rédacteur utilise la fonction Rac qui veut dire racine carrée. De plus, pour le rédacteur le signe "." désigne la multiplication.

Comment voulez-vous que l'on comprenne tout ça quand on ne connaît pas la technique de résolution d'une équation du second degré ?

En résumé, nous constatons les faits suivants :

- pour une solution donnée, il peut exister un ensemble presque illimité d'expressions différentes. Si n élèves, on a n solutions distinctes,

- dans chaque expression, on trouve des notations propres au rédacteur, ce qui entraîne des ambiguïtés et donc des interprétations différentes.

## 2.2 Exemple 2

Supposons que nous voulions exprimer en langage naturel la résolution de la racine cubique. Comme à priori, on ne connaît pas la démarche à suivre, nous utiliserons le résultat mathématique suivant :

On démontre que la suite suivante converge vers la racine cubique d'un nombre positif A:

*Donner une valeur arbitraire à X0*
*Xn+1 = ( 2 Xn + A/ Xn2 ) / 3*
*Test d'arrêt : valeur absolue ( Xn+1 - Xn) < μ*
*μ étant la précision "*

Commençons dans un premier temps par dérouler le principe du calcul à la main ( ou avec une calculatrice ne possédant pas la racine cubique ) avec les valeurs suivantes :

A = 30 ; x0 = 3 ; μ = 0.00005

Les détails de calcul sont présentés dans la table suivante :

| n | Xn | $Xn^2$ | $A/Xn^2$ | 2Xn | $2Xn + A/Xn^2$ | Xn+1 | \|Xn+1 - Xn \| |
|---|---|---|---|---|---|---|---|
| 0 | 3 | 9 | 3,33 | 6 | 9,33 | 3,11 | 0,11 |
| 1 | 3,11 | 9,6721 | 3,1017 | 6,22 | 9,3217 | 3,10723 | 0,00277 |
| 2 | 3,10723 | 9,65487 | 3,10723 | 6,2144 | 9,32169 | 3,10723 | 0,00000 |
| 3 | 3,10723 | | | | | | |

Un algorithme possible exprimé en langage naturel serait :

1. *Introduction des données*
   *A = 30, x0 = 3 , μ = 0.00005, n = 1*

2. *Calcul de Xn+1 = ( 2 Xn + A/Xn2 ) / 3*

3. *Comparaison de la valeur absolue de ( Xn+1 - Xn) et de μ*

4. *Si Valeur absolue( Xn+1 - Xn) > μ reprendre le calcul en 2 sinon continuer*

5. *Imprimer le résultat*

En plus des inconvénients cités plus haut, on constate dans cet exemple la difficulté d'automatiser le principe de répétition

### 2.3 Synthèse

Quelque soit l'algorithme, solution d'un problème donné, il doit être communiqué à l'ensemble. Ceci ne peut se faire en langage naturel pour les principales raisons suivantes:
- difficulté d'exprimer certaines idées, telles que la répétitive.
- ambiguïté

- plusieurs façons d'exprimer la même solution avec risque d'interprétations différentes.

La solution est donc d'utiliser un formalisme, ensemble de conventions, pour décrire un algorithme : le langage algorithmique.

# COURS 3. Structures de contrôle, introduction à la notion de variable

***Objectifs :*** *Introduire les structures de contrôle du langage algorithmique et écrire des algorithmes en résonnant sur des "ardoises".*

## 3.1 Structures de contrôle

Tous les algorithmes peuvent être exprimés par les "tournures" suivantes qu'on appelle structures de contrôle du fait qu'elles permettent de contrôler le déroulement des algorithmes :

**Le séquencement**

Exprime qu'un ensemble d'actions est exécuté dans un ordre bien précis.

**La répétitive**

Elle exprime qu'un ensemble d'actions se déroule plusieurs fois tant qu'une condition reste vérifiée.

*TANTQUE condition :*
*Action 1*
*Action 2*
*....*
*Action n*
*FINTANTQUE*

La condition constitue le critère d'arrêt.

**La conditionnelle**

Elle exprime qu'un ensemble d'actions est exécuté que si une condition est vérifiée.

*SI Condition :*
*Action 1*
*Action 2*
*....*
*Action n*
*FSI*

**L'alternative**

Elle exprime un choix entre deux ensembles d'actions à exécuter selon la valeur d'une condition.

*SI condition :*
*Action 1*
*Action 2*
*....*

*Action n*
*SINON*
*Action n+1*
*Action n+2*
*....*
*Action n + p*
*FSI*

**Remarque :** Chaque action peut être à son tour une répétitive, une conditionnelle, une alternative.

### 3.2 Notion d'ardoise

La propriété importante d'une ardoise est la possibilité d'écrire et d'effacer des informations continuellement. Ainsi une ardoise peut contenir une valeur à un moment donné et une autre valeur plus tard dans le temps.

A tout ardoise on peut associer un nom, par exemple le nom de son propriétaire (contenant)

On peut également lui associer un contenu, c'est à dire ce qui est écrit à un moment donné.

Si A désigne le nom attribué à l'ardoise, on désignera son contenu à un instant donné par (A).

### 3.3 Exemples

Reprenons nos deux algorithmes ( équation du second degré et racine cubique ) et donnons les algorithmes correspondants en utilisant :
- les structures de contrôle présentées,
- les ardoises

**Equation du second degré**

Nous utilisons un ardoise nommée Ardoise pour mettre la valeur du discriminant. Nous désignerons son contenu par l'écriture (Ardoise).

*Introduire les données A, B et C*
*Ardoise <--- B2 - 4 A.C*
*SI (Ardoise ) > 0 :*
*Ecrire( " la première racine est ", - B + Racine(Ardoise) / 4 A.C )*
*Ecrire( " la deuxième racine est ", - B - Racine(Ardoise) / 4 A.C )*
*SINON*
*SI (Ardoise) = 0*
*Ecrire( " Une racine double ", - B / 2A )*
*SINON*
*Ecrire( " Pas de racine réelle " )*
*FSI*
*FSI*

L'action Ecrire permet de restituer les résultats.
**Racine cubique**

Nous utiliserons 3 ardoises. Les ardoises 1 et 2 contiennent à tout moment deux éléments consécutifs de la suite et la troisième leur différence en valeur absolue.

*Introduire les valeurs A, X0 et μ*
*Ardoise1 <--- X0*
*Ardoise2 <--- ( 2 (Ardoise1) + A/ (Ardoise1)2 ) / 3*
*Ardoise 3 <--- Absolu ( (Ardoise2) - (Ardoise1) )*
*TANTQUE ( Ardoise3 ) < μ :*
*Ardoise1 <--- (Ardoise2)*
*Ardoise2 <--- ( 2 (Ardoise1) + A/ (Ardoise1)2 ) / 3*
*Ardoise 3 <--- Absolu ( (Ardoise2) - (Ardoise1) )*
*FINTANTQUE*
*Restituer le résultat qui est dans (Ardoise2)*

# COURS 4. Objets, notion d'affectation et structure d'un algorithme

***Objectifs :*** *définir les objets élémentaires manipulés par les algorithmes, introduire la notion d'affectation et écrire des algorithmes complets.*

## 4.1 Variables et constantes

Un algorithme opère sur des objets. A tout objet est associé un **nom** qui permet de l'identifier de façon unique. C'est généralement une suite de caractères alphanumériques dont le premier est alphabétique.

On distingue deux types d'objets :

- des objets qui peuvent varier durant le déroulement d'un algorithme **Variables** (ardoises).

- des objets qui ne peuvent pas variés par le déroulement d'un algorithme **Constantes**.

On peut répartir l'ensemble des objets en sous ensembles appelés **classes** ou types. Il existe 4 types standards :
- ENTIER : l'ensemble des entiers relatifs
- REEL : l'ensemble des réels
- BOOLEEN : les valeurs VRAI et FAUX
- CAR : l'ensemble des chaînes de caractères

En langage algorithmique, on définit les objets comme suit :

*CONST Type Nom = valeur*
*VAR Nom1, Nom2, .... : Type*

Exemples:

CONST REEL PI = 3.14
VAR A, B, C : ENTIER
VAR X, Y : CAR

## 4.2 Expressions sur les objets

Une expression est une suite d'opérandes reliés par des opérateurs.

Une expression peut être :
- arithmétique. On utilisera les opérateurs suivants +,-,/, *.
Exemple A + B/C, A/B/C

- logique. Les opérateurs les plus utilisés sont : ET, OU et NON.
Exemple X et Y, NON X OU Z

- relationnelle. Les opérateurs utilisés sont : <, <=, >, >=, =, <>.
Exemple : A < B, A+5 = B/C

## 4.3 Autres actions du langage algorithmique

### Affectation

C'est l'opération de base. Elle permet d'attribuer la valeur d'une expression calculable à une variable. On utilisera le symbole :=.

Exemple : A := (B + C) / D

### Lecture

Elle permet d'introduire des données dans des variables. Nous utiliserons l'opération

LIRE( X, Y, Z, .....)

Ce qui est équivalent à

X := première donnée
Y := deuxième donnée
Z := troisième donnée

### Ecriture

Elle permet de restituer des résultats. Nous utiliserons l'opération

ECRIRE(Expression1, Expression2,.....)

## 4.4 Structure d'un algorithme

Un algorithme est composé de deux parties :

- définition des données : en-tête
- corps de l'algorithme

l'en-tête précise

- le nom de l'algorithme
- définition des objets manipulés

Le corps est un ensemble d'actions ordonnées composé généralement de 3 parties :

- initialisations ( lectures, .. )
- itérations ( parcours d'un ensemble de données)
- opérations finales ( écritures, .. )

La description algorithmique est la suivante :

*ALGORITHME Nom*
*En-tête: définition des objets*
*DEBUT*
*Corps : définition du corps*
*FIN*

### 4.5 Exemples

Remarquer dans les algorithmes qui suivent qu'on n'utilise plus la notion d'ardoise. C'est désormais une variable. Noter aussi qu'on n'utilise non plus les parenthèses pour désigner le contenu d'une variable. Une variable désigne **contenant** quand elle est placée à gauche du signe d'affectation et **contenu** quand elle figure dans une expression.

Avec ces considérations, on peut maintenant écrire des algorithmes complets.

**Equation du second degré**

```
ALGORITHME Equation
VAR A, B, C, Delta : REEL
DEBUT
 LIRE(A B, C)
 Delta := B2 - 4 A.C
 SI  Delta  > 0 :
  ECRIRE( "la première racine est ",  - B+Racine(Ardoise)/4A.C)
  ECRIRE( "la deuxième racine est ",  -B- Racine(Ardoise)/4 A.C)
 SINON
  SI Delta = 0
   ECRIRE( " Une racine double ",  - B / 2A )
  SINON
   ECRIRE( " Pas de racine réelle " )
  FSI
 FSI
FIN
```

**Racine cubique**

```
Algorithme Racine_cubique
VAR  A, X0, Mu, Ancien, Nouveau, Difference : REEL
DEBUT
 LIRE(A, X0, Mu)
 Ancien := X0
 Nouveau := ( 2  Ancien +  A/  Ancien2 ) / 3
 Difference := Absolu (  Nouveau - Ancien  )
 TANTQUE  Difference <  Mu :
  Ancien := Nouveau
  Nouveau :=   ( 2 Ancien +  A/ Ancien2 ) / 3
  Difference := Absolu ( Nouveau - Ancien )
 FINTANTQUE
 Ecrire (" Racine carrée de ", A , " est ", Nouveau )
FIN
```

## TRAVAUX  DIRIGES

**1.** Exprimer en langage naturel, les solutions des problèmes suivants :
- la somme des N premiers entiers naturels.
- la somme des N premiers nombres impairs.

**2.** Proposer des conventions pour exprimer la répétitive et l'alternative.

**3.** Ecrire les corps des algorithmes suivants :

- Calcul de la somme S = 1 + 2 + . . . + N pour N donné.

- Déterminer les N premiers termes de la suite définie par :
U0 = U1 = 1
Un = Un-1 + Un-2 pour n > 1

- Calcul de la somme des N premiers termes de la suite xi/ i pour x donné.

- Calcul de la somme $x - x^3/3 + x^5/5 - ..$ en prenant N termes

N.B. : Le raisonnement se fait sur les ardoises.

**4.** Ecrire l'algorithme permettant de déterminer
- la somme des N premiers entiers naturels.
- la somme des N premiers nombres impairs.

**5.** Ecrire l'algorithme qui reconnaît si un nombre est premier ou non.

**6.** Ecrire un algorithme qui calcule la somme des N premiers nombres premiers.

**7.** Ecrire les algorithmes qui réalisent les fonctions suivantes :
- la somme $2^2 + 4^2 + 6^2 + .....$ en prenant N termes.
- le PGCD de 2 entiers positifs A et B donnés.
- la factorielle d'un nombre A donné.

**8.** On dispose d'une machine qui ne sait qu'additionner. Ecrire l'algorithme réalisant le produit de 2 nombres A et B donnés.

**Partie 2.**

# Programmation

## COURS 5. Présentation générale du langage PASCAL

***Objectifs :*** *introduire un minimum de concepts PASCAL pour commencer à programmer.*

### 5.1 Vocabulaire

C'est l'ensemble
- des identificateurs ( nom des variables et constantes ),
- mots clés ( BEGIN, END, WHILE, .....),
- séparateurs ('.', ';', ',',... ),
- opérateurs ('+', '-', '/',...),
- chaînes de caractères.

### 5.2 Objets

Les objets sont répartis en deux classes : les constantes et les variables. Les objets peuvent être de type : entier (INTEGER) , réel (REAL), caractère (CHAR), booléen (BOOLEAN)

Les constantes sont définies par :
*CONST Identificateur = valeur*

Les variables sont déclarées par*:*
*VAR Identificateur1, Identificateur2,.... : Type*

### 5.3 Expressions

Les expressions peuvent être arithmétiques (+, -, *, /, Div, ...), logiques ( Not, And, OR,...) ou relationnelles ( <, <=, >, >=, <> ).

### 5.4 Instructions

Une instruction PASCAL peut être simple ou composée.

Dans ce qui suit, Instruction désigne une instruction simple ou composée, Variable une variable et Expression une expression.

**Instructions simples**

Donnons dans un premier temps les instructions simples suivantes :

Affectation : *Variable := Expression*
Boucle While : *WHILE Expression DO instruction*
Conditionnelle : *IF Cond THEN Instruction*
Alternative : *IF Cond THEN Instruction ELSE Instruction*

Lecture : *READ(Variable)*
Ecriture : *WRITE (Expression)*

**Instruction composée**

Une instruction composée a la forme suivante :

*BEGIN Instruction1; Instruction2; .....END*

**Exemples**

Affectation : *A := (A + B) / C*

Boucle While : *WHILE X > 5 DO X := X + 5*

```
WHILE X+Y  >= 1000 DO
 BEGIN
                          X := X + 10 ;
     Y := Y - 2
END
```

Conditionnelle : *IF A > B THEN READ(X, Y)*

Alternative : *IF X < Y THEN WRITE(X) ELSE WRITE(Y)*

Lecture : *READ(X, Y, Z)*
Ecriture : *WRITE ( ' Résultats : ', Res)*

## 5.5 Structure d'un programme

Un programme PASCAL a la forme suivante :

```
PROGRAM Nom;
 définition des constantes  ( Const .... )
 définition des variables  (Var .... )
BEGIN
 Corps du programme
END.
```

## 5.6 Exemples

Reprenons nos exemples et donnons les programmes PASCAL correspondants.

**Equation du second degré**

```
PROGRAM Equation;
VAR A, B, C, Delta : REAL;
BEGIN
```

```
READ(A B, C);
Delta := B              *B - 4 A*C;
IF  Delta  > 0
THEN
 BEGIN
 WRITE( 'la première racine est',  - B+RAC(Ardoise)/4A*C);
 WRITE( 'la deuxième racine est',  -B- RAC(Ardoise)/4 A*C)
 END
ELSE
 IF Delta = 0
  WRITE( ' Une racine double',  - B / 2*A )
 ELSE
  WRITE( ' Pas de racine réelle' )

END.
```

**Racine cubique**

```
PROGRAM Racine_cubique;
VAR  A, X0,  Mu, Ancien, Nouveau, Difference : REAL;
BEGIN
 READ(A, X0, Mu);
 Ancien := X0;
 Nouveau := ( 2* Ancien +  A/  (Ancien*Ancien) ) / 3 ;
 Difference := Abs (  Nouveau - Ancien  ) ;
 WHILE  Difference < Mu DO
  BEGIN
   Ancien := Nouveau;
   Nouveau :=   ( 2*Ancien +  A/ Ancien*Ancien ) / 3;
   Difference :=  Abs ( Nouveau - Ancien )
  END;
 WRITE (' Racine carrée de', A , " est ", Nouveau )
END.
```

# COURS 6. Entrées/Sorties PASCAL

***Objectifs**: fournir les outils PASCAL pour, d'une part, introduire les données de l'écran et restituer les résultats sur écran et d'autre part pour imprimer des tableaux ou tout autre dessin. Introduire également le concept de fichier 'TEXT'.*

## 6.1 Lecture

Toute introduction de données se fait par l'ordre :

*READ[LN]( V1, V2, ...Vn)*

[ ] désigne une partie facultative.

Vi est une variable de type INTEGER, REAL ou CHAR.

Cette instruction provoque la lecture de n données à partir de l'écran. Pour les nombres, les données sont séparées par des blancs. Pour les caractères, le lecture se fait toujours caractère/caractère.

Si l'option LN est présente, il y a positionnement à la ligne suivante après la lecture des données.

## 6.2 Ecriture

Un restitution de résultats se fait par l'ordre

*WRITE[LN] (P1, P2, ....., Pn)*

P1, P2, ...., Pn sont des expressions suivies éventuellement du mode d'écriture.

On peut avoir les 3 formes :
- E
- E : E1
- R : E1 : E2

avec E, E1, E2, R des expressions.

E : expression de type INTEGER, CHAR, REAL, BOOLEAN
R : expression de type REAL
E1, E2 de type INTEGER indique une largeur de champ ( E1 : nombre total de caractères de la valeur écrite, E2 : nombre de chiffres après le point décimal )

**Première forme : E**

Si E1 n'est pas spécifiée, une valeur ( pour E1 ) par défaut est assurée dépendant du type c'est à dire :
CHAR : 1
INTEGER : 11
BOOLEAN : 8
REAL : 20

Ces valeurs peuvent changer selon la version PASCAL considérée.

**Deuxième forme : E : E1**

E1 désigne le nombre total de caractères à écrire. Si E1 ne suffit pas, le nombre ne sera pas tronqué sinon cadré à gauche par des blancs. Si E est réelle, le nombre est écrit en virgule flottante.

**Troisième forme : R : E1 : E2**

La partie fractionnaire de R doit comporter E2 caractères. Le nombre est écrit en format virgule fixe sans exposant.

**Exemple**

```
PROGRAM  Exemple;
CONST Tab = 5;
VAR I, I1, I2  : INTEGER;
 R1  : REAL;
 B : BOOLEAN;
BEGIN
 B := TRUE ;
 I1 := -3 ;
 R1 := 4.5;
 I2 := 6 ;
 WRITELN(' Exemple de sortie');
 WRITELN; { saut de ligne }
 WRITELN(' ':Tab, 'I1=',I1:I2,'R1=',R1:13);
 WRITELN(I1:2, I1, 'B=', B : 3 );
 WRITELN( R1:8, R1:4:1)
END.
```

Ce programme donne en sortie :

```
 I1=       -3R1=  4.500000E+00
-3-3B=TRUE
 4.5E+00  4.5
```

## 6.3 Les fichiers TEXT

Au lieu de lire les données à partir de l'écran, ce qui peut être fastidieux lors de la mise au point des programmes, il est préférable et même très avantageux de lire les données à partir d'un fichier TEXT construit préalablement par un éditeur de texte.

La déclaration d'un tel fichier se fait comme suit

```
VAR  Fe : TEXT
```

où Fe désigne le nom logique du fichier.

Tout fichier déclaré de la sorte doit être lié à un fichier physique. Ce lien est établit grâce à l'instruction ASSIGN définie comme suit

*ASSIGN ( Nom logique, Nom physique)*

De plus, le fichier doit être ouvert par l'opération

*RESET(Nom logique)*

Les opérations de lectures sont faites par :

*READ[LN] ( Nom logique, V1, V2, .....Vn)*

De même, il est conseillé de récupérer les résultats sur un fichier TEXT, puis d'aller vers un éditeur de texte et d'exploiter calmement les résultats de votre programme. Il faudra donc déclarer le fichier et le lier avec le fichier physique comme précédemment.

Par contre le fichier doit être ouvert par

*REWRITE( Nom logique )*

et les opérations d'écriture se font par

*WRITE[LN] ( Nom logique, P1, P2, ....Pn)*

Il faut toujours rajouter l'instruction CLOSE( Nom logique ) afin de ne pas perdre les dernières écritures sur le fichier.

**Exemple**

Le programme qui suit effectue la somme des données lues sur le fichier Entree.pas et écrit les sommes temporaires sur le fichier Sortie.pas.

```
PROGRAM SOMME;
VAR
 Fe, Fs : TEXT;
 I, S, Nombre, Val : INTEGER ;
BEGIN
   ASSIGN(Fe, 'Entree.pas');
ASSIGN(Fs, 'Sortie.pas');
RESET(Fe); REWRITE(Fs);
READLN(Fe, Nombre);
S := 0;
FOR I:= 1 TO Nombre DO
 BEGIN
  READLN(Fe, Val);
  S := S + Val;
  WRITELN(Fs, 'Somme temporaire = ', S);
 END;
WRITELN(Fs, '> Somme = ', S);
CLOSE(Fs)
 END.
```

**Contenu du fichier Entree.pas :**

```
12      Nombre d'éléments
34
65
87
34
23
64
93
88
12
54
34
33
```

**Contenu du fichier Sortie.pas :**

```
Somme temporaire = 34
Somme temporaire = 99
Somme temporaire = 186
Somme temporaire = 220
Somme temporaire = 243
Somme temporaire = 307
Somme temporaire = 400
Somme temporaire = 488
Somme temporaire = 500
Somme temporaire = 554
Somme temporaire = 588
Somme temporaire = 621
> Somme = 621
```

# TRAVAUX DIRIGES

**1.** Ecrire un algorithme de multiplication de deux entiers n'utilisant que des additions et des soustractions.

**2.** Ecrire un algorithme de division de deux entiers donnant le quotient entier et le reste.

**3.** Ecrire l'algorithme pour le calcul du PGCD (Plus grand diviseur commun) de deux entiers.

**4.** Ecrire l'algorithme pour le calcul du PPMC (Plus petit multiple commun) de deux entiers.

**5.** Ecrire l'algorithme, puis le programme (en PASCAL) pour calculer Cos (x) avec une précision µ donnée à l'aide de la série :
Cos( x) = 1 - x2/2! + x4/4! - ...

**6.** Ecrire un algorithme qui calcule à une valeur approchée µ donnée la racine carrée d'un nombre a > 0 sachant que la série :

X1 = a
Xn = (a/Xn-1 + Xn-1 ) / 2

converge vers la racine carrée de a.
Donner le programme PASCAL correspondant.

**7.** Imprimer 10 factures d'électricité, sachant qu'il y a deux tranches, la première facturée à 21.47 centimes le KWh et la deuxième à 14.25 centimes le KWh, que le montant de l'abonnement est de 89.09 DA et que la TVA est 19.5 %. Les données sont, pour chaque client, son nom et sa consommation dans chacune des deux tranches.

**8.** On veut imprimer une liste de bons du type suivant, dans lesquels, à la place de X, on fera figurer les noms lus en donnée :

```
**********************************************

Monsieur X a gagné un poste à transistor
Le retirer à la mairie, se munir du présent bon

**********************************************
```

Le nombre de gagnants est donné.

**9.** Construire la table des racines cubiques des 20 premiers nombres. Utiliser la suite convergente définie en cours. Le résultat sera de la forme :

```
    ----------------------------------------
I     nombre     I   Racine cubique    I
----------------------------------------
I       1        I        1.0000       I
----------------------------------------
I       2        I                     I
......
......
```

**Partie 3.**

# Expérimentation sur la machine de Turing

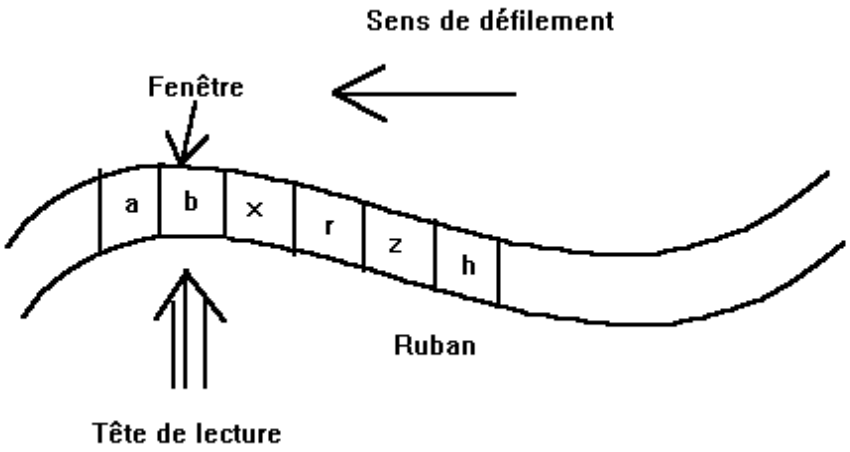
## COURS 7. Machine-caractères

***Objectif :*** *maîtriser le formalisme algorithmique en développant des algorithmes sur la machine-caractères.*

### 7.1 Présentation

La machine est une boite qui renferme un ruban. Au cours de l'utilisation de la machine, le ruban défile devant une fenêtre. Le défilement se fait toujours dans un seul sens. Le ruban est composé d'une suite finie de cases comportant chacune un caractère. Le ruban se termine par une case spéciale contenant le caractère '.'. A tout instant, le seul caractère du ruban auquel la machine a accès est celui qui est présent dans la fenêtre. Nous baptisons ce caractère le caractère courant. L'obtention du caractère suivant se fait par un ordre de lecture.

Elle peut être schématisée comme suit :




**Machine-Caractères**

### 7.2 Exemples

Sur cette machine, on peut développer de nombreux algorithmes tels que la détermination du nombre total de caractères, la recherche du plus long mot, la recherche des mots de longueur donnée, etc...
A titre d'exemples, développons quelques uns.

**1. Nombre de 'LES'**

```
ALGORITHME LES
VAR
 LES : ENTIER
 C : CAR
DEBUT
 Les := 0
 LIRE(C)
 TANTQUE C # ' '.' :
  SI C = 'L' :
   Lire(C)
   SI C = 'E' :
    LIRE(C)
    SI C = 'S' :
     Les := Les + 1
     LIRE(C)
    FSI
   FSI
  SINON
   LIRE(C)
  FSI
 FINTANTQUE
 ECRIRE(Les)
FIN
```

**2. Nombre de mots se terminant par "S'**

```
ALGORITHME S
VAR
 Cs : ENTIER
 C : CAR
DEBUT
 Cs := 0  LIRE(C)
 TANTQUE C # '.' :
  SI C = 'S' :
   LIRE(C)
   SI C = ' ' OU C = '.'  :
    Cs := Cs + 1
   FSI
  SINON
   LIRE(C)
  FSI
 FINTANTQUE
FIN
```

**3. Nombre de mots**

```
ALGORITHME Nombre_de_mots
VAR
 Cmot : ENTIER
 C, Mot : CAR
DEBUT
 LIRE(C)
```

```
Cmot := 0
TANTQUE C # '.' :
 TANTQUE C = ' ' ET C # '.' :
  LIRE(C)
 FINTANTQUE
 Mot := ''
 TANTQUE C # ' ' ET C # '.' :
  Mot := Mot + C
  LIRE(C)
 FINTANTQUE
 SI MOT # '' :
  Cmot := Cmot + 1
 FSI
FINTANTQUE
ECRIRE( Cmot)
FIN
```

# COURS 8. Machine-nombres

***Objectifs:***
*- maîtriser le formalisme algorithmique en développant des algorithmes sur la machine-nombres.*
*- introduire à ce niveau la structure de contrôle "POUR".*

## 8.1 Présentation

Par analogie à la machine-caractères, on peut définir la machine-nombres où chaque ordre de lecture délivre le prochain nombre. On suppose que la première lecture délivre le nombre de nombres. On pourra alors développer les algorithmes tels que la recherche du maximum, la recherche d'un élément donné, la recherche du nombre de sous suites croissantes, etc.

## 8.2 Structure de contrôle " POUR"

On utilise cette structure quand on connaît à l'avance le nombre d'itérations d'une boucle. La formulation algorithmique est la suivante :

```
POUR  Variable := Expression1, Expression2, Expression3
 Action 1
 Action 2

 .....

 Action n
FINPOUR
```

Variable désigne une variable entière.
Expression1, Expression2 et Expression3 désignent des expressions entières.
Expression1 est la valeur initiale de Variable, Expression2 la valeur finale et expression3 l'incrément.

Toute boucle "POUR" peut se traduire en boucle "TANTQUE". Par contre, l'inverse n'est pas toujours possible.

Exemple 1

```
      POUR I:= 1, 20, 4
 ECRIRE(I)
FINPOUR
```

Les valeurs imprimées sont 1, 4, 8, 12, 16 et 20.

Exemple 2

```
S := 0
POUR I:= 40, 1, -1
 S := S + I
FINPOUR
```

C'est la somme des 40 premiers entiers naturels.

**Utilisation en PASCAL**

En PASCAL, l'incrément est toujours égal à +1 ou -1.

Dans le cas où l'incrément est +1 on utilise l'instruction "FOR" comme suit :

*FOR Variable = Expression1 TO Expression2 DO Instruction*

Dans le cas où l'incrément est -1 on utilise l'instruction "FOR" comme suit :

*FOR Variable= Expression1 DOWNTO Expression2 DO Instruction*

Instruction désigne une instruction simple ou composée.

Exemple :

L'exemple 2 se traduit en PASCAL comme suit :

```
S := 0;
FOR I:= 40 DOWNTO 1 DO
 S := S + I
```

## 8.3 Exemples

De même, présentons quelques algorithmes.

**1. Recherche du maximum**

On suppose qu'il y a K éléments sur la machine-nombres ( K > 0)

```
ALGORITHME Maximum
VAR
 MAX, NOMBRE : ENTIER
 I, K : ENTIER
DEBUT
 LIRE( Max)
 POUR k := 2, K :
  LIRE(Nombre)
  SI Nombre > Max :
   Max := Nombre
  FSI
 FINPOUR
FIN
```

**2. Nombre de sous suites croissantes**

On suppose qu'il y a K éléments sur la machine-nombres ( K > 0)

```
ALGORITHME Sous_suite
VAR
 N, Nombre, K, I: ENTIER
```

```
DEBUT
 N := 1
 LIRE ( Nombre)
 POUR I :=  2, K :
  Prec := Nombre
  LIRE(Nombre)
  SI Nombre < Prec :
   N := N +1
  FSI
 FINPOUR
 ECRIRE( N)
FIN
```

**3. Premier nombre dont le carré est égal à la somme des deux précédents**

On suppose qu'il  y a K éléments sur la machine-nombres ( K > 0)

```
ALGORITHME Premier_carré
VAR
 Prec1, Prec2, Nombre : ENTIER
 I, K : ENTIER
 Trouv : BOOLEEN
DEBUT
 LIRE(Prec1)
 LIRE(Prec2)
 Trouv := FAUX
 I := 3
 TANTQUE I <= K et NON Trouv :
  LIRE(Nombre)
  SI Nombre2 = Prec1 + Prec2
   Trouv := VRAI
  SINON
   Prec1 := Prec2
   Prec2 := Nombre
   I := I + 1
  FSI
 FINTANTQUE
 SI Trouv :
  ECRIRE(Nombre)
 SINON
  ECRIRE(' Néant ')
 FSI
FIN
```

# TRAVAUX DIRIGES

**I. Machine-caractères**

Sur la machine-caractères, écrire les algorithmes suivants :

**1.** Nombre de caractères

**2.** Nombre de blancs.
**3.** Nombre d’occurrences de 'LE'.
**4.** Les mots de longueur L donnée
**5.** Les mots contenant au moins une voyelle.
**6.** Nombre de mots se terminant par 'TION'.
**7.** Nombre de mots contenant 'TION'.
**8.** Nombre de mots commençant par 'A' et se terminant par 'S'.
**9.** Le mot le plus long.
**10.** Les mots ne contenant pas de voyelles.
**11.** Les mots contenant au moins trois voyelles.

**II. Machine-nombres**

Sur la machine-nombres, écrire les algorithmes suivants :

**1.** Le plus grand élément.
**2.** Le plus petit élément
**3.** 1. et 2.
**4.** La somme des nombres impairs.
**5.** La somme des nombres pairs.
**6.** Recherche d'un élément donné.
**7.** La liste des termes compris entre a et b donnés.
**8.** Les multiples de a donné.
**9.** Les diviseurs de a donné.
**10.** La somme des nombres positifs.
**11.** Le nombre de sous suites croissantes.

**Partie 4.**

# Programmation modulaire

## COURS 9. Actions composées

***Objectifs**: construire des modules ou actions composées quand un groupe d'actions se répète plusieurs fois dans un algorithme.*

### 9.1 Exemple d'introduction : calcul de Cnp

Considérons la formule Cnp = n! / p! (n-p)!.
L'algorithme qui calcule cette formule est le suivant :

```
ALGORITHME Cnp
VAR
 R1, R2, R3, N, P : ENTIER
 I : ENTIER
DEBUT  LIRE( N, P)
 R1 := 1
 POUR I:= 2, N
  R1 := R1 * I
 FINPOUR

 R2 := 1
 POUR I:= 2, P
  R2 := R2 * I
 FINPOUR

 R3 := 1
 POUR I:= 2, N-P
  R3 := R3 * I
 FINPOUR
 ECRIRE( " résultat : ", R1 / (R2 * R3)  )
FIN
```

Nous remarquons dans cet algorithme que la séquence

```
 R_ := 1
 POUR I:= 2, _
  R_ := R_ * I
 FINPOUR
```

se répète 3 fois où le symbole '_' désigne les parties qui changent.

Afin d'éviter cette duplication, on crée une **action composée**, puis il suffit de l'appeler plusieurs fois avec des données différentes ce que nous appellerons **paramètres**.

### 9.2 Actions composées

Quand une séquence d'actions se répète plusieurs fois dans un algorithme, il est avantageux et même nécessaire de créer une **action composée** (ou **module**). En langage algorithmique, on définit un module de la façon suivante :

```
      ACTION Nom ( V1, V2,       )

 Définition des paramètres d'entrée et de sortie
 Définition des objets locaux
DEBUT
 Corps
FIN
```

Au niveau de la définition de l'action , les paramètres sont dits **formels**.

La structure d'un  algorithme devient alors :

```
      ALGORITHME Nom
 Définition des objets
 Définition des modules
DEBUT
 Corps
FIN
```

L'appel se fait par Nom (P1, P2,....). Les paramètres sont dits **réels** ou **effectifs**.

Dans l'exemple précédent, on crée l'action Fact comme suit :

```
ACTION Fact (N, R)
 VAR  I, N, R : ENTIER
DEBUT
 R := N
 POUR I = N- 1, 2, -1
  R  :=  R *  I
 FINPOUR
FIN
```

L'algorithme devient :

```
ALGORITHME Cnp
VAR
 N, P, R1, R2, R3, Result : ENTIER

{ Définition de l'action Fact }
ACTION Fact (N, R)
 VAR  I, N, R : ENTIER
DEBUT
 R := N
 POUR I = N- 1, 2, -1
```

```
  R := R * I
 FINPOUR
FIN

DEBUT
 LIRE(N, P)
 Fact (N, R1)
 Fact( P, R2)
 Fact (N-P, R3)
 Result := ( R1 * R2 ) / R3
 ECRIRE ( Result )
FIN
```

### 9.3 Fonctions

Quand le résultat de l'action composée est unique et de type arithmétique (ou caractère) , il est préférable d'écrire l'action composée sous la forme d'une *fonction*.

Une fonction est définie comme suit :

```
FONCTION Nom ( Paramètres d'entrée ) : Type
 Définition des paramètres et des objets locaux
DEBUT
 Corps
FIN
```

Une fonction est utilisée directement dans une expression.

Dans le corps de la fonction il doit toujours exister une affectation du genre
" Nom de la fonction :=  valeur ".
C'est cette valeur qui est retournée par la fonction.

La fonction associée au module Fact précédent est la suivante :

```
        FONCTION Fact ( N ) : ENTIER
VAR N, I, R : entier
DEBUT
 R  := N
 POUR I=N- 1, 2, -1
  R := R *  I
 FINPOUR
 Fact := R
FIN
```

L'algorithme devient :

```
        ALGORITHME Cnp
VAR  N, P : ENTIER

{ Définition de la fonction Fact }
FONCTION Fact ( N ) : ENTIER
VAR  N, I, R : ENTIER
DEBUT
```

```
 R  := N
 POUR I=N- 1, 2, -1
  R := R *  I
 FINPOUR
 Fact := R
FIN

DEBUT
 LIRE (N, P)
 ECRIRE ( ( Fact(N)* Fact(N-P) ) / Fact(P)  )
FIN
```

### 9.4 Prédicat

Quand le résultat de l'action composée est unique et de type booléen, il est préférable d'écrire l'action composée sous la forme d'un*prédicat.*

```
        PREDICAT Nom ( Paramètres )
 Définition des paramètres et des objets locaux
DEBUT
 Corps
FIN
```

Dans le corps du prédicat il doit toujours exister une affectation du genre
" Nom du prédicat :=  valeur ".

**Exemple :**

```
        PREDICAT Egal (A, B)
VAR A, B : ENTIER
DEBUT
 Egal :=  (A=B)
FIN
```

L'algorithme qui suit utilise le prédicat Egal comme suit :

```
        ALGORITHME Exemple
VAR  I, Cpt, A, N  : ENTIER
DEBUT
 Cpt := 0
 LIRE(N)
 POUR I=1, N :
  LIRE( A)
  SI Egal (N, A) :
   Cpt  :=  Cpt + 1
  FSI
 FINPOUR
FIN
```

**Remarque**

La définition d'un module ( action composée, fonction ou prédicat) est récursive, ce qui veut dire qu'un module peut à son tour contenir un ou plusieurs autres modules.

## 9.5 Utilisation en PASCAL

En PASCAL, on parle de procédure et fonction. Les prédicats sont des fonctions de type booléen. les paramètres sont définis au niveau des en-têtes. Un résultat d'une procédure doit être déclaré avec l'option VAR.

**Définition d'une procédure**

```
        PROCEDURE Nom  ( [ VAR ] P1, P2, .... : Type; etc..)
              En-tête : définition des objets locaux
BEGIN
 Corps
        END
```

Dans l'en-tête on ne redéfinit pas les paramètres.
L'appel se fait *nom (N1, N2,....)*
Exemple :

```
PROGRAM Cnp;
VAR
 N, P, R1, R2, R3, Result : INTEGER;

{ Définition de l'action Fact }
PROCEDURE Fact (N : INTEGER; VAR R : INTEGER);
 VAR  I : INTEGER;
BEGIN
 R := N ;
 FOR  I = N- 1 DOWNTO  2 DO
  R := R * I
END;

BEGIN
 READ(N, P);
 Fact (N, R1);
 Fact( P, R2);
 Fact (N-P, R3);
 Result := ( R1 * R2 ) / R3;
 WRITE ( Result )
END.
```

**Définition d'une fonction**

```
        FUNCTION Nom ( P1, P2 ,.... : type;  etc...) : Type
 Définition des objets locaux
BEGIN
 Corps de la fonction
FIN
```

Dans le corps, il doit exister au moins une affectation du genre
" Nom de la fonction  :=  Valeur "

Une fonction est utilisée directement dans une expression.

Exemple :

```
        PROGRAM Cnp;
VAR  N, P : INTEGER;

{ Définition de la fonction Fact }
FUNCTION Fact ( N ) : INTEGER;
VAR  I, R : INTEGER;
BEGIN
 R  := N ;
 FOR I:=N- 1 DOWNTO 2 DO
  R := R *  I ;
 Fact := R ;
END;

BEGIN
 READ (N, P)
 WRITE ( ( Fact(N)* Fact(N-P) ) / Fact(P)  )
END.
```

# COURS 10. Analyse descendante

***Objectifs :***
*- diviser un problème en sous problèmes pour régner, telle est la devise de l'analyse descendante,*
- construire des modules ou des actions composées quand la technique de l'analyse descendante est employée pour la résolution d'un problème.

## 10.1 Définition

L'analyse descendante est un mode de pensée visant à construire des algorithmes en partant d'un niveau très général et en détaillant peu à peu chaque traitement, jusqu'à arriver au niveau de description le plus bas.

## 10.2 Caractéristique

C'est une méthode permettant d'aborder un problème, de reconnaître sa structure et donc les traitements qu'il faudra mettre en oeuvre pour le résoudre.

## 10.3 Technique

On peut énoncer la technique de l'analyse descendante comme suit :

(i) - Utilisation des structures de contrôle

(ii) - Essai de reconnaître dans un problème les traitements permettant de se ramener à un problème connu. Laisser à part les problèmes complexes P1, P2, ...

(iii) - Application itérative de (i) et (ii) jusqu'à arriver au niveau le plus bas.

A chaque étape, on suppose qu'il existe une machine abstraite capable de traiter les Pi.

La construction d'un algorithme par la méthode d'analyse descendante revient à définir des machines abstraites dont le fonctionnement est décrit par des algorithmes adaptés à chaque niveau de machine.

## 10.4 Exemple

Le problème : décomposer une liste de N nombres en facteurs premiers.

**Première étape**

On ne s'occupe que du problème du parcours des données.

```
POUR I:= 1 , N :
 LIRE ( Nombre )
 Décomposer le nombre Nombre
FINPOUR
```

**Deuxième étape**

Il s'agit de la décomposition d'un nombre. Pour cela, on  parcours tous les nombres premiers de 2 à Nombre / 2 en vue de déterminer les exposants.

```
Premiercourant := 2
TANTQUE Premiercourant <= Nombre/2 :
 Déterminer l'exposant Exposant
 SI Exposant # 0 :
  ECRIRE ( Premiercourant, Exposant)
 FSI
 Déterminer le prochain nombre premier, soit Premiercourant
FINTANTQUE
```

**Troisième étape**

Il s'agit de rechercher l'exposant associé à un nombre premier. Comme on est au niveau le plus bas, on donne l'algorithme au détail.

```
ACTION Exposant ( Nombre, Premiercourant, Exposant )
VAR
 Divise : BOOLEEN
 Exposant, Premiercourant, Nombre : ENTIER
DEBUT
  Divise := VRAI
   Exposant := 0
 TANTQUE Divise :
  A := Nombre / Premiercourant
  SI A * Premiercourant = Nombre :
   Exposant := Exposant + 1
   Nombre := A
  SINON
   Divise := FAUX
  FSI
 FINTANTQUE
FIN
```

**Quatrième étape**

Nous devons rechercher le prochain nombre premier à partir d'un nombre premier, soit Premiercourant.

```
I := Premiercourant + 2
TANTQUE I n'est pas un nombre premier
 I:=  I + 2
FINTANTQUE
```

Remarquez que nous prenons que les nombres impairs comme candidats pour le test : premier ou non.

**Cinquième étape**

Il s'agit de reconnaître un nombre premier. Comme on est au niveau le plus bas, on donne l'algorithme au détail. De plus, comme le résultat est booléen, nous l'écrivons sous forme de prédicat.

```
        PREDICAT Estilpremier ( Nombre )
 VAR
  Divisible : BOOLEEN
  J, A, Quotient: ENTIER
 DEBUT
  J := 2
  Divisible := FAUX
  A := Nombre DIV 2
  TANTQUE  J <= A  ET NON Divisible
   Quotient := Nombre DIV J
   SI Quotient * J = Nombre :
    Divisible := VRAI
   SINON
    J := J + 1
   FSI
  FINTANTQUE
  Estilpremier := NON Divisible
 FIN
```

# COURS 11. Communication entre modules

***Objectifs :***
*- introduire la notion de portée des objets,*
*- donner deux moyens de communication entre modules : par paramètres et par variables globales.*

### 11.1 Nomenclature

Dans notre contexte le mot bloc désigne soit un algorithme soit un module ( action, fonction, , prédicat )

Soit la structure de l'algorithme suivante:

```
        M1
 [ a, b, c, d
 M2
 [c, d, e
  M3
  [ a, b
    ]
  M4
  [ d, e
  ]
         ]
]
```

M1, M2, M3 et M4 désignent des blocs et a, b, c, d, e des variables.

On définit les termes suivants :

Blocs parallèles, blocs disjoints : M3 et M4 sont parallèles ou disjoints
Bloc contenant un autre bloc : M1 contient M2
Bloc contenu dans un autre bloc : M4 est contenu dans M2
Bloc intérieur à un autre bloc : M3 est un bloc intérieur à M1
Bloc extérieur à un autre bloc : M1 est un bloc extérieur à M2
Blocs emboîtés, bloc imbriqués : M1, M2 et M3 sont des blocs emboîtés

Tout objet défini dans un bloc est dit**local** à ce bloc. Il n'est pas connu à l'extérieur de ce bloc.

Tout objet défini dans un bloc est connu dans tout bloc intérieur à ce bloc sauf s'il est redéfini. On dira qu'il est **global** à ces blocs intérieurs.

### 11. 2 Portée des objets

On désigne par portée d'un objet le domaine de visibilité de l'objet. C'est donc l'ensemble des blocs ou l'objet est connu. Dans l'exemple précédent, les portées des objets sont présentées dans la table suivante:

| | a | b | c | d | e |
|---|---|---|---|---|---|
| M1 | l | l | l | l | nd |
| M2 | gb | gb | l | l | l |
| M3 | l | l | gb | gb | gb |
| M4 | gb | gb | gb | l | l |

l désigne Local, gb désigne Global et nd Non défini.

### 11.3 Communication entre modules

La communication entre les modules peut se faire de deux façons :
- à l'aide de paramètres
- à l'aide de variables globales

Nous reviendrons sur des exemples plus loin.

# COURS 12. Illustration de l'analyse descendante et la communication entre modules à travers un exemple.

***Objectifs :*** *illustrer les notions de module, d'analyse descendante et de communication entre modules à travers un exemple.*

## 12.1 Enoncé

Sur le ruban de la machine-caractères se trouve une suite de télégrammes. Chaque télégramme est terminé par le mot 'FINTEL'. Chaque télégramme est constitué de mots( suite de caractères non blancs) séparés par un ou plusieurs blancs. Ces mots sont divisés en catégories :
- les mots à facturer
- les mots de service non facturables qui sont 'STOP' et 'FINTEL'.

La suite de télégrammes est terminée par le télégramme vide ( télégramme ne contenant aucun mot ou que des 'STOP')

Ecrire un algorithme, qui pour chaque télégramme, imprime le texte du télégramme ( les mots étant séparés par un seul blanc) suivi du nombre de mots à facturer et du nombre de mots dépassant 12 caractères. Ces mots seront tronqués dans le texte imprimé.

Exemple de télégramme

"Bonjour stop je souhaite bon courage à tout le monde stop stop fintel stop fintel"

## 12.2 Les différents modules

Le premier problème qui nous préoccupe est celui du parcours des télégrammes. Supposons qu'on dispose de l'action composée Traiter-télégramme qui permet de reconnaître un télégramme et de nous renvoyer deux valeurs F et K désignant respectivement le nombre de mots facturables et le nombre de mots de longueur supérieure à 12.

On peut alors exprimer l'algorithme comme suit :

Algorithme_principal

```
Telegrammenonvide := VRAI
TANTQUE Telegrammenonvide :
 Traiter_telegramme
 ECRIRE(F, K)
 Telegrammenonvide :=  NON ( F=0)
FINTANTQUE
```

De même, on essaye de résoudre le problème de parcours au sein même d'un télégramme. Pour cela considérons les deux modules suivants  : l'un permet d'obtenir le mot suivant et l'autre permet de le traiter.

L'algorithme correspondant à Traiter_telegramme est donc :

Traiter_telegramme

```
F := 0
```

```
K := 0
Nonfindetelegramme := VRAI
TANTQUE Nonfindetelegramme :
 Obtenir_mot
 Traiter_mot
FINTANTQUE
```

Il nous reste plus qu'à détailler les deux modules :

Obtenir_mot récupère le prochain mot et détermine pour chaque mot sa longueur.

Obtenir_mot

```
LIRE(c)
I := 0
Mot := ''
TANTQUE C= ' ' :
   LIRE(C)
FINTANTQUE
TANTQUE C <> ' ' :
 I := I + 1
 SI I <= 12 : Mot := Mot + C FSI
 LIRE(c)
FINTANTQUE
```

Traiter_mot compte le nombre de mots facturables, le nombre de mots de longueur supérieure à 12.

Traiter_mot

```
ECRIRE ( Mot, ' ')
SI Mot <> 'Fintel' ET  Mot <> 'Stop' :
 F := F + 1
FSI
SI I > 12 :
 K := K + 1
FSI
SI mot = 'Fintel' :
 Nonfindetelegramme := FAUX
FSI
```

## 12.3 Rédaction de la solution

On peut rédiger la solution de diverses façons. Pour chaque cas donnons la structure modulaire et le programme PASCAL correspondant.

### 12.3.1 Communication par variables globales

**Structure modulaire**

*[ALGORITHME*
*F, K, Telegrammenonvide*

*[Traiter-telegramme*
*I, Mot, Nonfindetelegramme*

*[Obtenr-mot*
*C*
*]*
*]*
*[Traiter-mot*
*]*
*]*

F, K et Telegrammenonvide sont des variables globales aux modules Traiter_telegramme, Obtenir_mot et Traiter_mot.

I, Mot, Nonfindetelegramme sont des variables locales du module Traiter_telegramme et globales aux modules Obtenir_mot et Traiter_mot.

C est une variable locale du module Obtenir_mot.

**Programme PASCAL correspondant**

```
PROGRAM Telegrammes;
 TYPE Tchaine = STRING[12];
 VAR
  F, K : INTEGER;
  Telegrammenonvide : BOOLEAN;
  Fe, Fs : TEXT;

  PROCEDURE Traiter_telegramme;
   VAR
    I : INTEGER;
    Mot : Tchaine ;
    Nonfindetelegramme : BOOLEAN;

   PROCEDURE Obtenir_mot;
    VAR C : CHAR;
    BEGIN
     READ(Fe, C);
     Mot := '';
     I := 0;
     WHILE C =' ' DO READ(Fe, C);
     WHILE (C <> ' ') DO
      BEGIN
       I := I + 1;
       IF I <= 12 THEN Mot := Mot + C ;
       READ(Fe, C)
      END;
    END;
```

```
    PROCEDURE Traiter_mot ;
     BEGIN
      WRITE(Fs, Mot, ' ');
      IF (Mot <> 'FINTEL') AND (Mot <> 'STOP')
      THEN F := F + 1;
      IF I > 12 THEN K:= K + 1 ;
      IF Mot = 'FINTEL' THEN Nonfindetelegramme := FALSE
     END;

    BEGIN
     F := 0;
     K := 0;
     Nonfindetelegramme := TRUE;
     WHILE  Nonfindetelegramme DO
      BEGIN
       Obtenir_mot;
       Traiter_mot
      END;
    END;

 BEGIN
  ASSIGN(Fe, 'D_telegr.Pas');
  ASSIGN(Fs, 'R_telegr.Pas');
  RESET(Fe);
  REWRITE(Fs);
  Telegrammenonvide := TRUE;
  WHILE Telegrammenonvide DO
   BEGIN
    Traiter_telegramme;
    WRITELN(Fs, 'F=', F, '     K=',K)  ;
    Telegrammenonvide := NOT ( F=0)
   END;
  CLOSE(Fs);
 END.
```

### 12.3.2 Communication par paramètres

**Structure modulaire**

*[ALGORITHME*
*Freel, Kreel, Telegrammenonvidereel*

*[Traiter-telegramme  (Fformel, Kformel)*

*Ireel, Motreel, Nonfindetelegrammereel*

*[Obtenr-mot ( Iformel,  Motformel)*
*C*
*]*
*[Traiter-mot(Iformel,Motformel,Fformel,Kformel,*
*Nonfindetelegrammeformel)*
*]*

*]*
*]*

Freel, Kreel et *Telegrammevidereel* sont des variables locales de l'algorithme.

Ireel, Motreel et *Nonfindetelegrammereel* sont des variables locales du module Traiter_telegramme.

C est une variable locale du module Obtenir_mot.

**Programme PASCAL correspondant**

```
PROGRAM Telegrammes;
 TYPE Tchaine = STRING[12];
 VAR
  Freel, Kreel : INTEGER;
  Telegrammenonvidereel : BOOLEAN;
  Fe, Fs : TEXT;

  PROCEDURE Traiter_telegramme ( VAR Fformel, Kformel : INTEGER) ;
   VAR
    Ireel : INTEGER;
    Motreel : Tchaine;
    Nonfindetelegrammereel : BOOLEAN;

   PROCEDURE Obtenir_mot ( VAR Iformel : INTEGER;VAR Motformel : Tchaine);
    VAR C : CHAR;
    BEGIN
     READ(Fe, C);
     Motformel := '';
     Iformel := 0;
     WHILE C =' ' DO READ(Fe, C);
     WHILE (C <> ' ') DO
      BEGIN
       Iformel := Iformel + 1;
       IF Iformel <= 12 THEN Motformel := Motformel + C ;
       READ(Fe, C)
      END;
    END;

   PROCEDURE Traiter_mot (Iformel : INTEGER; Motformel : Tchaine; VAR Fformel,
Kformel : INTEGER;      VAR Nonfindetelegrammeformel : BOOLEAN);
    BEGIN
     WRITE(Fs, Motformel, ' ');
     IF (Motformel <> 'FINTEL') AND (Motformel <> 'STOP')
     THEN Fformel := Fformel + 1;
     IF Iformel > 12 THEN Kformel:= Kformel + 1 ;
     IF Motformel = 'FINTEL' THEN Nonfindetelegrammeformel := FALSE
    END;

   BEGIN
    Freel := 0;
```

```
      Kreel := 0;
      Nonfindetelegrammereel := TRUE;
      WHILE  Nonfindetelegrammereel DO
        BEGIN
          Obtenir_mot (Ireel, Motreel);
          Traiter_mot(Ireel, Motreel, Freel, Kreel, Nonfindetelegrammereel )
        END;
    END;

BEGIN
  ASSIGN(Fe, 'D_telegr.Pas');
  ASSIGN(Fs, 'R_telegr.Pas');
  RESET(Fe);
  REWRITE(Fs);
  Telegrammenonvidereel := TRUE;
  WHILE Telegrammenonvidereel DO
    BEGIN
      Traiter_telegramme (Freel, Kreel);
      WRITELN(Fs, 'F=', Freel, '     K=',Kreel)  ;
      Telegrammenonvidereel := NOT ( Freel=0)
    END;
  CLOSE(Fs);
END.
```

### 12.3.3 Communication par paramètres et par variables globales

**Structure modulaire**

```
[ALGORITHME
  Fglobal, Kglobal, Telegrammenonvideglobale
 [Traiter-telegramme
  Ireel, Motreel et Nonfindetelegrammereel
  [Obtenr-mot ( I, MOT)
   C
  ]
 ]
  [Traiter-mot(I,Mot, Nonfindetelegramme )
  ]
]
```

Fglobal, Kglobal et Telegrammevideglobal sont des variables locales de l'algorithme et sont donc globales à tous les modules.

Ireel, Motreel et Nonfindetelegrammereel sont des variables locales du module Traiter_telegramme.

C est une variable locale du module Obtenir_mot

**Programme PASCAL correspondant**

```
PROGRAM Telegrammes;
```

```
TYPE Tchaine = STRING[12];
VAR
  Fglobal, Kglobal : INTEGER;
  Telegrammenonvideglobal : BOOLEAN;
  Fe, Fs : TEXT;

  PROCEDURE Traiter_telegramme ;
    VAR
      Ireel : INTEGER;
      Motreel : Tchaine;
      Nonfindetelegrammereel : BOOLEAN;

    PROCEDURE Obtenir_mot ( VAR Iformel : INTEGER;VAR Motformel :Tchaine );
      VAR C : CHAR;
      BEGIN
        READ(Fe, C);
        Motformel := '';
        Iformel := 0;
        WHILE C =' ' DO READ(Fe, C);
        WHILE (C <> ' ') DO
          BEGIN
            Iformel := Iformel + 1;
            IF Iformel <= 12 THEN Motformel := Motformel + C ;
            READ(Fe, C)
          END;
      END;

    PROCEDURE Traiter_mot (Iformel : INTEGER; Motformel : Tchaine;
                      VAR Nonfindetelegrammeformel : BOOLEAN);
      BEGIN
        WRITE(Fs, Motformel, ' ');
        IF (Motformel <> 'FINTEL') AND (Motformel <> 'STOP')
        THEN Fglobal := Fglobal + 1;
        IF Iformel > 12 THEN Kglobal:= Kglobal + 1 ;
        IF Motformel = 'FINTEL' THEN Nonfindetelegrammeformel := FALSE
      END;

    BEGIN
      Fglobal := 0;
      Kglobal := 0;
      Nonfindetelegrammereel := TRUE;
      WHILE  Nonfindetelegrammereel DO
        BEGIN
          Obtenir_mot (Ireel, Motreel);
          Traiter_mot(Ireel, Motreel, Nonfindetelegrammereel )
        END;
    END;

BEGIN
  ASSIGN(Fe, 'D_telegr.Pas');
  ASSIGN(Fs, 'R_telegr.Pas');
  RESET(Fe);
```

```
  REWRITE(Fs);
  Telegrammenonvideglobal := TRUE;
  WHILE Telegrammenonvidereel DO
   BEGIN
     Traiter_telegramme ;
     WRITELN(Fs, 'F=', fglobal, '     K=',Kglobal)  ;
     Telegrammenonvideglobal := NOT ( Fglobal=0)
   END;
  CLOSE(Fs);
END.
```

**Remarque**

Les trois programmes présentés admettent comme fichier de données (D_telegr.pas)

"Bonjour STOP commencer à programmer en PASCAL STOP bon courage à tout le monde FINTEL  Réfléchissez avant de commencer STOP FINTEL STOP STOP N'oubliez pas de remettre un rapport FINTEL STOP FINTEL FINTEL"

et comme résultats (R_telegr.Pas)

Bonjour STOP commencer à programmer en PASCAL STOP bon courage à tout le monde FINTEL F=12  K=0
Réfléchissez avant de commencer STOP FINTEL F=4  K=0
STOP STOP N'oubliez pas de remettre un rapport FINTEL F=6  K=0
STOP FINTEL F=0  K=0

**TRAVAUX DIRIGES**

**1.** Ecrire les prédicats suivants :
(a) Egal(A, B) : vrai___ si A = B
(b) Parfait(A) : vrai ___ si A est un nombre parfait ( un nombre A est parfait s'il est égal à la somme de ses diviseurs, A exclu )
Utiliser ces prédicats pour déterminer dans une liste de nombres( machine-nombres) d'abord le nombre de doubles d'un nombre A donné puis les nombres parfaits.
**2.** Ecrire la fonction factorielle, puis l'utiliser pour écrire l'algorithme qui imprime le triangle de PASCAL.

**3.** Calculer l'expression ( Sin x + Sin 3x ) / Sin 4x avec Sin x = x - x3/3! + x5/5! - .....  (-1)k x 2k+ 1 / (2k+1)! en prenant n termes, n donné.

**4.** Considérons une liste de N nombres entiers ( N > 1). On veut écrire un seul algorithme qui répond aux questions suivantes :
- déterminer le troisième nombre premier s'il existe,
- déterminer le deuxième carré parfait s'il existe,
- déterminer le nombre de diviseurs du premier nombre pair et du dernier nombre impair.

Utiliser les modules suivants pour écrire cet algorithme :
- *prem( nombre )* : prédicat égal à vrai si nombre est premier, faux sinon.
- *carré (nombre)* : prédicat égal à vrai si nombre est carré parfait, faux sinon.
- *nbdiv(nombre)* : fonction qui donne le nombre de diviseurs de nombre.

- *pair(nombre)* : prédicat égal à vrai si nombre est pair, faux sinon.
Ecrire le programme PASCAL correspondant.

**5.** Donner la portée de toutes les variables définies dans la structure algorithmique suivante:

```
ALGORITHME Portee
VAR
 A, B, C, D, E  : ENTIER
 X, Y, Z, T  : BOOLEAN

 ACTION A1
 VAR
  A, B : CAR
  T: REEL
 DEBUT
  ...
 FIN

 ACTION A2 ( T)
 VAR
  T, E : ENTIER

  ACTION A2 ( A, B )
  VAR
   A, B : ENTIER
   Z, Y : REEL
  DEBUT
   .....
  FIN
 DEBUT
  ......
 FIN
DEBUT
 ......
FIN
```

**Partie 5.**

# Objets composés

## COURS 13. Les objets composés

***Objectifs :*** *introduire la notion d'objets composés ou complexes, c'est à dire des objets formés à partir de plusieurs autres objets de natures hétérogènes.*

### 13.1 Introduction

La mémoire est une suite finie de mots ( de longueur 16 bits pour les micro-ordinateurs ) adressée de 0 à N ( N dépend de la machine).

A chaque objet ( variable / constante) est associée une *représentation* : entier, réel, booléen, caractère...

A chaque mode de représentation est associé un ensemble d'opérations. Ainsi, on distingue le + entier du + réel, etc... ...

Le langage algorithmique est un outil de description puissant. De la même façon que l'on peut construire des actions composées et de les utiliser en tant que telles au niveaux des corps d'algorithmes on peut aussi définir des objets complexes appartenant à un autre type que l'on décrit au niveau de l'en-tête.

### 13.2 Définition de type

Au niveau du langage algorithmique, un type est construit comme suit :

```
      TYPE   Nomdutype = STRUCTURE
 Idf1 : Type1
 Idf2 : Type2
 ....
FIN
```

Typei peut être un type standard ou construit.

**Exemple**

```
      TYPE Article = STRUCTURE
 Nom  : CAR
 Prenom : CAR
 Age  : ENTIER
FIN
```

### 13.3 Déclaration des variables

On peut maintenant définir des objets de type Article comme suit:

*Var X : Article*

On peut aussi si nécessaire définir une relation d'ordre entre les éléments appartenant à une même classe d'objets.

### 13.4 Accès

Si X est une variable de type Article, X.Nom permet d'accéder au champ Nom de la variable composée X.

Noter l'imbrication de type ( arborescence )

### 13.5 Utilisation en PASCAL

Un type ou classe d'objet est défini par

```
TYPE Nom du type = RECORD
 Nom1 : Type1 ;
 Nom2 : Type2 ;
 ....
 Nomp : Typen
END
```

Les variables de cette classe sont définies par :

*VAR N1,N2, ..... : Nom du type*

L'accès se fait par le chemin et utilisation des points

N1.M1.P1.....

**Partie 6.**

# Vecteurs

## COURS 14. Notion de vecteur

***Objectifs :*** *introduire une structure de données élémentaire : vecteur ou tableau.*

### 14.1 Introduction

Un vecteur peut être défini comme un ensemble d'éléments de même nature ( type ). Le type peut être quelconque (entier, réel, booléen, caractère) ou un type construit. Dans ce dernier cas, on dira qu'on a un vecteur de structures ou d'objets composés.

### 14.2 Caractéristiques

Considérons l'exemple du dictionnaire :

Si l'on veut chercher un élément, on n'est pas obligé de parcourir toutes les pages précédentes. On dira que l'accès à un élément du dictionnaire est *direct.*

En général, l'élément recherché ( définition ) est caractérisé par son indicatif.

Les mots sont ordonnés. On ouvre le dictionnaire au hasard et au vu des mots, on recherche soit dans les pages antérieures soit dans les pages postérieures.

On peut aussi parcourir un dictionnaire du début vers la fin, c'est *l'accès séquentiel.*

En informatique certains dispositifs physiques de stockage possèdent cette propriété d'accès direct . On peut citer comme exemples les mémoires centrales des ordinateurs et les disques.

### 14.3 Définition formelle

Soient
- V, un ensemble fini quelconque d'éléments : l'ensemble des valeurs.
- n, un entier, nombre d'éléments du vecteur.

Un vecteur est une application d'un intervalle I de N formé de n éléments de V.
I est appelé l'ensemble des index ( indices ).

**Remarques**

- L'ensemble des indices est noté [ 1, N ]
- Un vecteur peut être vide
- On peut définir un vecteur par son graphe, c'est à dire l'ensemble des couples (Indice, valeur).
- Un élément peut être : simple, structuré ou même un autre vecteur.
- On peut définir une relation d'ordre dans l'ensemble des valeurs.

### 14.4 Terminologie et Notation

Définissons quelques termes sur les vecteurs.

**Borne inférieure** :

C'est le plus petit élément de l'intervalle des indices.

**Borne supérieure**:

C'est le plus grand élément de l'intervalle des indices.

**Taille** :

C'est le nombre d'éléments du vecteur.

**Notation**

Un vecteur est noté ( T[1,n], V)
Un élément est le couple ( i, T(i) )ou T(i)

**Sous vecteur** :

C'est la restriction de T à un intervalle de l'ensemble des indices.

## 14.5 Accès à un élément du vecteur

Il n'existe qu'une opération d'accès : l'opération d'indexation

Soit un vecteur T[1:N]

Quelque soit i dans l'intervalle[1, n] : T(i) délivre la valeur de l'élément d'indice i dans le vecteur

Si i n'est pas dans [1, n], T(i) est indéfini.

**Remarque**

- T(1) donne l'accès au premier élément.
- On peut faire les affectations a := T(i) ou T(i) := a

## 14.6 Vecteur ordonné

On parle d'un vecteur ordonné quand l'ensemble des valeurs possibles est muni d'une relation d'ordre. De plus, un vecteur est dit ordonné ou trié en ordre croissant si quelque soit i dans [1, n-1] : T(i) <= T(i+1).

## 14.7 Mesures

Mesurer un algorithme sur les vecteurs consiste à déterminer son encombrement et le nombre d'affectations et de comparaisons portant uniquement sur les éléments du vecteur.

On désigne par **encombrement** d'un vecteur le produit de sa taille par l'espace occupé par un de ses éléments.

### 14. 8 Description algorithmique

On décrit un vecteur en langage algorithmique comme suit :

*VAR  Nom : TABLEAU[1..N] DE Type*

Nom est le nom du tableau, et Type est un type quelconque.

# COURS 15. Algorithmes sur les vecteurs

***Objectif** : fournir les principaux algorithmes sur les vecteurs, particulièrement ceux qui réalisent les tris, en détaillant certains.*

## 15.1 Algorithmes traitant un seul vecteur

On distingue généralement deux classes d'algorithmes :
- les algorithmes de parcours en vu de réaliser un traitement donné,
- les algorithmes de parcours en vu de rechercher une valeur donnée.

**Exemples**

Considérons quelques algorithmes :

**1. Recherche dichotomique dans un vecteur**

On ne peut faire ce type de recherche que dans la cas où le vecteur est ordonné. Le principe est le suivant :

On utilise 3 variables Bi, Bs et Milieu, représentant respectivement la borne inférieure, la borne supérieure et l'indice du milieu, c'est à dire ( Bi + Bs ) / 2.

Initialement Bi est 1, Bs est M ( taille du vecteur).

Considérons la valeur se trouvant au milieu du vecteur V[Bi..Bs], c'est à dire V[Milieu]

Si V[Milieu] est la donnée recherchée, la recherche se termine avec succès.

Autrement, comme le vecteur est ordonné, à ce niveau on peut orienter la recherche uniquement dans l'un des deux sous vecteurs en modifiant Bi ou Bs selon que l'élément recherché est supérieur ou inférieur à V[Milieu]

On obtient l'algorithme suivant :

```
ALGORITHME Dichotomie
VAR
 V : TABLEAU[1..M] DE ENTIER
 D : ENTIER
   Bi, Bs, M, Milieu : ENTIER
 Trouv : BOOLEEN
DEBUT
 Bi := 1
 Bs := M
 Trouv := FAUX
 TANTQUE Bi <= Bs ET NON Trouv :
  Milieu:= ( Bi + Bs ) DIV 2
  SI V(Milieu) = D :
   Trouv := VRAI
  SINON
   SI D < V(Milieu) :
    Bs := Milieu - 1
```

```
   SINON
    Bi := Milieu + 1
   FSI
  FSI
 FINTANTQUE
FIN
```

**2. Recherche séquentielle dans un vecteur**

Il s'agit de rechercher un élément séquentiellement à partir de la première position.

```
ALGORITHME  Recherche1
VAR  V : TABLEAU[1..M] DE ENTIER
 D : ENTIER
 I : ENTIER
 Trouv : BOOLEEN
DEBUT
 LIRE(M)
 I := 1
 Trouv  := FAUX
 TANTQUE NON Trouv et I <= M
  SI V(I) = D :
   Trouv := VRAI
  SINON
   I := I + 1
  FSI
 FINTANTQUE
FIN
```

**3. Recherche séquentielle dans un vecteur ordonné**

Il s'agit de rechercher un élément séquentiellement à partir de la première position. De plus, si on rencontre un élément supérieur à l'élément recherché, on arrête le parcours du vecteur.

```
ALGORITHME  Recherche2
VAR
 V : TABLEAU[1..M] DE ENTIER
 D : ENTIER
 I : ENTIER
 Trouv, Arret : BOOLEEN
DEBUT
 I := 1
 Trouv, Arret  := FAUX
 TANTQUE NON Trouv ET NON Arret ET I <= N
  SI V(I) = D :
   Trouv := VRAI
  SINON
   SI V(I) > D :
    Arret := VRAI
   SINON
    I := I + 1
```

```
   FSI
  FSI
 FINTANTQUE
FIN
```

### 15.2 Algorithmes traitant plusieurs vecteurs

Ce sont les algorithmes qui opèrent sur 2 ou plusieurs vecteurs. Les algorithmes les plus courants sont la  fusion, l'interclassement et l'éclatement.

**Exemple : Interclassement de deux vecteurs**

L'interclassement de deux vecteurs exigent que les vecteurs soient ordonnés. Il consiste à construire un troisième vecteur ordonné contenant tous les éléments des deux vecteurs.

```
ALGORITHME  Interclasser
VAR
 N1, N2 : ENTIER
 V1 : TABLEAU[1..N1] DE ENTIER
 V2 : TABLEAU[1..N2] DE ENTIER
 V3 : TABLEAU[1..N1+N2] DE ENTIER
 I, J, K : ENTIER
 Trouv : BOOLEEN
DEBUT
 LIRE(N1, N2)
 I, J := 1
 K := 0
 TANTQUE I <= N1 ET J <= N2 :
  K := K + 1
  SI V1(I) < V2(J) :
   V3(K) := V1(I)
   I := I + 1
  SINON
   V3(K) := V2(J)
   J := J + 1
  FSI
 FINTANTQUE
 TANTQUE I <= N1 :
  K := K + 1
  V3(K) := V1(I)
  I := I + 1
 FINTANTQUE
 TANTQUE  J<= N2 :
  K := K + 1
  V3(K) := V2(J)
  J := J + 1
 FINTANTQUE
FIN
```

### 15.3 Algorithmes de mise à jour

On distingue 3 types de modifications :

**Modification**:

Elle consiste à remplacer une valeur par une autre.

**Insertion**

Elle consiste à ajouter une nouvelle valeur. On peut rajouter à la fin ou à n'importe quelle position du vecteur. Dans ce dernier cas, on fera des décalages.

Soit un vecteur V[1..M] contenant N éléments. L'algorithme qui suit insère la valeur D à la K-ième position :

```
ALGORITHME  Insérer
VAR
 M, N : ENTIER
 V : TABLEAU[1..M] DE ENTIER
 D : ENTIER
 I, K : ENTIER

DEBUT
 LIRE(M, N)
 { M : taille du vecteur et N : nombre d'éléments présents }
 LIRE(V) { Remplissage du vecteur }
 LIRE(K)
 LIRE(D)
 SI N < M :
  N := N + 1
  POUR I=N, K, -1
   V(I+1) := V(I)
  FINPOUR
  V(K) := D
 SINON
  Ecrire("Saturation du vecteur ")
 FSI
FIN
```

**Suppression**

La suppression peut  être logique ou physique.

- supprimer *logiquement* un élément consiste à utiliser un champ additionnel pour la présence ou non de élément.

- supprimer *physiquement* un élément consiste à l'éliminer complètement du vecteur.  On procédera donc par décalage.

## 15.4 Algorithmes de tri

Plusieurs méthodes de tri existent. Exposons dans ce qui suit quelques unes.

**Tri par sélection**

On utilise 2 vecteurs, soient VNT(vecteur à trier) et VT (vecteur résultat).

a) Chercher le maximum dans VNT, soit Max.
b) Répéter (n-1) fois les étapes suivantes :
- chercher le minimum dans VNT, soit Min
- remplacer-le par Max
- mettre Min dans VT successivement de la position 1 à n.

**Tri par sélection et permutation**

on utilise un seul vecteur.

a) Rechercher le maximum dans V[1..N], soit R son rang.
b) Permuter l'élément de rang R avec l'élément de rang N.

Répéter a) et b) successivement dans les vecteurs V[1..N-1], V[1..N-2], .....

**Tri par bulles**

Un seul vecteur est utilisé.

a) Par une série de permutations 2 à 2 ramener le maximum à la dernière position.
b) Répéter cette opération successivement dans les sous-vecteurs V[1..N-1], V[1..N-2], ....

Formellement, l'algorithme est le suivant :

```
ALGORITHME Tri_bulles
VAR
 I, J, N : ENTIER
 Temp : ENTIER
 V : ARRAY[1..N] DE ENTIER
DEBUT
 LIRE(N)
 LIRE(V)
 POUR I := 1, N-1  :
  POUR J:= N, I + 1, - 1 :
   SI V(J-1) > V(J) :
    Temp := V(J-1)
    V(J-1) := V(J)
    V(J) := Temp
   FSI
  FINPOUR
 FINPOUR
FIN
```

**Tri par Comptage**

On utilise deux vecteurs : V[1..N] et COMPT[1..N]

a) Initialisation globale de COMPT à 1.
b) Prendre successivement chaque élément, soit T(i), du vecteur à trier et le comparer avec les éléments suivants T(k), k=i+1, i+2, ...

Si T(i) > T(k) on incrémente COMPT(i) d'une unité
sinon on incrémente COMPT(k) d'une unité

### 15.5 Utilisation en PASCAL

Un vecteur V est défini en PASCAL par la déclaration

VAR V : ARRAY[ Bi..Bs] OF Typeqq

où Bi et Bs désignent les limites de l'intervalle des indices. Typeqq désigne un type quelconque.

L'accès au I-ième élément du vecteur se fait par

V[I]

Contentons-nous, dans ce qui suit, de traduire en PASCAL l'algorithme de recherche décrit précédemment.

```
PROGRAM   Recherche1;
CONST
 M = 30 ;
VAR
 V : ARRAY[1..M] OF INTEGER;
 D : INTEGER;
 I : INTEGER;
 Trouv : BOOLEAN;
BEGIN
 FOR I:=1 TO M DO READ( V[I] );
 I := 1;
 Trouv  := FALSE;
 WHILE (NOT Trouv) AND ( I <= N) DO
  IF V(.I.) = D
  THEN Trouv := TRUE
  ELSE   I := I + 1
END.
```

# COURS 16. Vecteurs de vecteurs

***Objectifs :*** *introduire les matrices et d'une façon générale les tableaux à n dimensions, puis donner leur représentation mémoire.*

## 16.1 Vecteurs de vecteurs ou matrices

Les éléments d'un vecteur peuvent être de type quelconque. On peut avoir par exemple un vecteur d'entiers, un vecteur de caractères ou un vecteur d'objets composés. Les éléments d'un vecteur peuvent aussi être des vecteurs. On parlera alors de vecteur de vecteurs ou tout simplement matrice ou tableau à deux dimensions.

## 16.2 Tableaux à n dimensions

De même, les éléments d'une matrice peuvent être de type quelconque. On peut avoir par exemple une matrice d'entiers, une matrice de caractères ou une matrice d'objets composés. Les éléments d'une matrice peuvent aussi être des vecteurs. On parlera alors de matrice de vecteurs ou tout simplement de tableau à trois dimensions.

De cette façon, on peut généraliser à l'ordre n.

## 16.3 Représentation mémoire

Considérons un tableau à n dimensions défini comme suit :
A(a1:b1; a2:b2; .....an:bn)
où a1, a2, ..., an désignent les bornes inférieures dans chaque dimension et b1, b2, ..., Bn les bornes supérieures.

En mémoire centrale, le rangement est fait par sous tableaux A(i, *, *, ..., *) pour i variant de a1 à b1 comme suit :

A(a1, *, *, ..., *),
A(a1+1, *, *, ..., *),
A(a1+2, *, *, ..., *),
.....,
A(b1, *, *, ..., *)

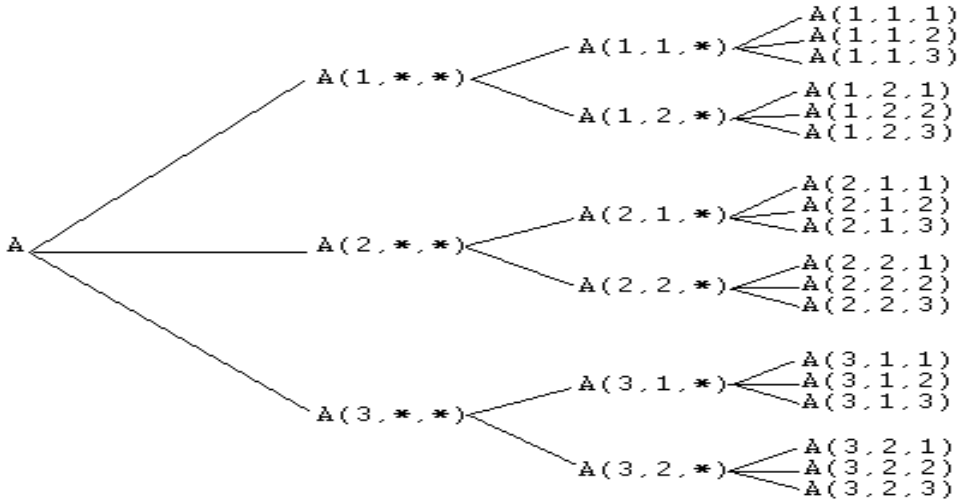
A l'intérieur de chaque sous tableau A(i, *, *, ...., *), l'ordre suivant des sous sous-tableaux est considéré :

A(i, a2, *, *, ..., *),
A(i, a2+1, *, *, ..., *),
.....,
A(i, b2, *, *, ..., *)

Et ainsi de suite ...

Donc le rangement de la matrice se fait ligne par ligne. De plus, ce sont Les derniers indices qui varient le plus rapidement.

**Exemple :**




Comment déterminer l'adresse d'un élément A(i1,i2 ..,in)?

Posons di = bi - ai + 1

L'adresse de A(i1, *, *, ....) est
AD1 = Base + (i1-a1) d2d3...dn

Base étant l'adresse début du tableau.

L'adresse de A(i1, i2, *, ...., *) est
AD2 = AD1 + (i2-a2)d3d4...dn

L'adresse de A(i1, i2, ...in) est par conséquent :
ADn = (i1-a1)d2d3..dn + (i2-a2)d3d4...dn + ....+(in-1-an-1)dn + (in-an)

Dans cette formule, il y a une partie constante et une variable.

Partie constante :

$$\text{Base} - (a1 \prod_{i=2,n} di + a2 \prod_{i=3,n} di + ..+an\text{-}1dn + an)$$

Partie variable

$$i1 \prod_{i=2,n} di + i2 \prod_{i=3,n} di + ... + in\text{-}1dn + in$$

Si a1 = a2 = ..... = an = 0, l'adresse de A(i1, i2, ...in) est donné par la formule :

i1d2d3..dn + i2d3d4...dn + ....+in-1dn + in

Ou bien :

$$\sum_{j=1,n} (ij \prod_{i=j+1,n} di)$$

### 16.4 Description algorithmique

On décrit un tableau à plusieurs dimensions en langage algorithmique comme suit :

*VAR Nom : TABLEAU[1..N1, 1..N2, ....., 1..Nd] DE Type*

Nom est le nom du tableau, Type est un type quelconque et d est la dimension du tableau.

# TRAVAUX DIRIGES

**1. Algorithmes traitant un seul vecteur**

- Recherche du maximum dans un vecteur.

- Moyenne des élèves dans une classe de n élèves pour une matière donnée.( Un élément du vecteur est une structure à 2 champs : Nom, Note ).Algorithme et Programme.

- Calcul d'un polynôme de degré n donné par le tableau de ses coefficients pour une valeur de x donnée.

- Admission dans une école avec baccalauréat C et moyenne supérieure ou égale à 10 ou Bac D et moyenne supérieure ou égale à 11 ou Bac B et moyenne supérieur ou égale à 13. Algorithme et Programme PASCAL.

**2. Algorithmes traitant plusieurs vecteurs**

- Soient 2 vecteurs A[1..N] et B[1..M] d'entiers. Construire un troisième vecteur C contenant tous les éléments de A et B qui sont des carrés.

- Construire le vecteur intersection à partir deux vecteurs donnés.

- Même chose pour le vecteur différence et le vecteur union.

- Eclater un vecteur V d'entiers positifs en 2 vecteurs A et B selon le critère ( premier ou pas). On utilisera le prédicat Est-il-premier .

**3. Algorithmes de mise à jour**

- Insérer un élément donné après une valeur donnée du vecteur.

- Insérer un élément donné à la position p donnée du vecteur.

- Quelles modifications sont à faire dans les deux algorithmes précédents si le vecteur est ordonné.

- Supprimer *logiquement* un élément consiste à utiliser un champs additionnel pour la présence ou non de élément. Définir la structure du vecteur et écrire l'algorithme correspondant.

- Supprimer *physiquement* un élément consiste à l'éliminer complètement du vecteur. Ecrire l'algorithme correspondant.

**4. Algorithmes de tri**

Retrouver les algorithmes de tri correspondants aux méthodes :
. tri par sélection
. tri par sélection et permutation
. tri par comptage

Chaque algorithme est à mesurer en dénombrant les comparaisons et les affectations portant uniquement sur les éléments du vecteur à trier.

**5. Vecteurs de vecteurs**

- *Triangle de PASCAL*

Construire le triangle de PASCAL dans un tableau à deux dimensions. Ecrire la procédure PASCAL qui l'imprime.

- *Produit de 2 matrices.*

Algorithme et programme PASCAL correspondant au produit de deux matrices.

- *Histogramme*

Une assemblée vote en choisissant une possibilité sur 6. Fabriquer dans un tableau à deux dimensions l'histogramme du vote. Ecrire la procédure PASCAL qui l'imprime. Améliorer la solution en donnant des noms aux candidats.

- *Résolution d'un système de n équations à n inconnues.*

Algorithme et programme PASCAL permettant de résoudre un système de n équations à n inconnues.

- *Calcul de la moyenne*

Dans une classe de 30 élèves, en vue d'établir un classement à partir du tableau des notes obtenues en 9 compositions et des coefficients correspondants, calculer pour chaque élève la somme des produits de chaque note par son coefficient et la moyenne correspondante. Imprimer la liste des élèves par ordre de mérite. Algorithme et programme.

**Partie 7.**

# Listes linéaires chaînées

## COURS 17. Les listes linéaires chaînées

***Objectifs** : introduire le concept d'allocation dynamique afin de présenter une structure de données évolutive : liste linéaire chaînée.*

### 17.1 Notion d'allocation dynamique

L'utilisation des tableaux tels que nous les avons définis dans le chapitre précédent implique que l'allocation de l'espace se fait tout à fait au début d'un traitement, c'est à dire que l'espace est connu à la compilation.

Pour certains problèmes, on ne connaît pas à l'avance l'espace utilisé par le programme. on est donc contraint à utiliser une autre forme d'allocation. L'allocation de l'espace se fait alors au fur et à mesure de l'exécution du programme.

Afin de pratiquer ce type d'allocation, deux opérations sont nécessaires : allocation et libération de l'espace.

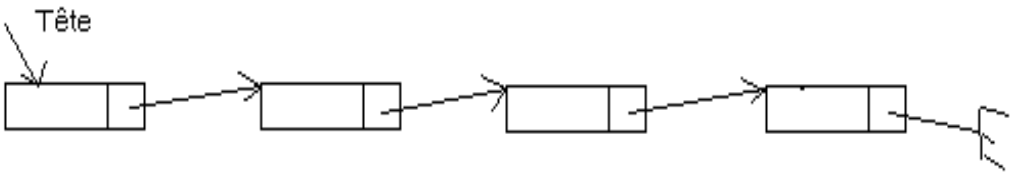
### 17.2 Exemple

Supposons que l'on veuille résoudre le problème suivant : " Trouver tous les nombres premiers de 1 a n et les stocker en mémoire."
problème réside dans le choix de la structure d'accueil. Si on utilise un tableau, il n'est pas possible de définir la taille de ce vecteur avec précision même si nous connaissons la valeur de n (par exemple 10000). Ne connaissant pas la valeur de n, on a aucune idée sur le choix de sa taille. On est donc, ici, en face d'un problème où la réservation de l'espace doit être dynamique.

### 17.3 Définition d'une liste linéaire chaînée

Une liste linéaire chaînée (Llc) est un ensemble de maillons (alloués dynamiquement)chaînés entre eux.

Schématiquement, on peut la représenter comme suit :




Un élément d'une Llc est toujours une structure ( objet composé) avec deux champs : un champ Valeur contenant l'information et un champ Adresse donnant l'adresse du prochain maillon. A chaque maillon est associée une adresse. On introduit ainsi une nouvelle classe d'objet: le type

POINTEUR en langage algorithmique.  Une Llc est caractérisée par l'adresse de son premier élément. NIL constitue l'adresse qui ne pointe aucun maillon.

Dans le langage algorithmique, on définira le type d'un maillon comme suit :

```
        TYPE Typedumaillon = STRUCTURE
  Valeur  : Typeqq    { désigne un type quelconque }
  Adresse  : POINTEUR(Typedumaillon)
 FIN
```

### 17.4 Modèle sur les listes linéaires chaînées

Afin de développer des algorithmes sur les Llcs, on construit une machine abstraite avec les opérations suivantes :

Allouer, Libérer, Aff_Adr, Aff_Val, Suivant, Valeur

définies comme suit :

Allouer(T, P) : allocation d'un espace de taille spécifiée par le type T. L'adresse de
   cet espace est rendue dans la variable POINTEUR P.
Libérer(P)  : libération de l'espace pointé par P.
Valeur(P)  : consultation du champ Valeur du maillon d'adresse P.
Suivant(P)  : consultation du champ Adresse du maillon d'adresse P.
Aff_Adr(P, Q): dans le champ Adresse du maillon d'adresse P, on range l'adresse Q.
Aff_Val(P,Val): dans le champ Valeur du maillon d'adresse P, on range la valeur Val.

Cet ensemble est appelé modèle.

# COURS 18. Algorithmes sur les listes et programmation en PASCAL

***Objectif** : présenter les algorithmes classiques sur les listes en détaillant quelques uns incluant leur utilisation en PASCAL.*

## 18. 1 Algorithmes sur les listes

De même que sur les vecteurs, on peut classer les algorithmes sur les LLCs comme suit :
a) parcours : accès par valeur, accès par position,
b) mise à jour : insertion, suppression,
c) algorithmes sur plusieurs LLCs : fusion, interclassement, éclatement,...
d) tri sur les LLCs.

Donnons quelques algorithmes :

**1. Création d'une liste et listage de ses éléments**

L'algorithme qui suit crée une liste linéaire chaînée à partir d'une suite de valeurs données, puis imprime la liste ainsi créée.

```
ALGORITHME Creer
VAR
 P, Q, Liste : POINTEUR( Typedumaillon)
 I, Nombre : ENTIER
 Val : Typeqq
DEBUT
 Liste := NIL
 P := NIL
 LIRE(Nombre)
 POUR I:= 1, Nombre :
  LIRE ( Val )
  Allouer ( Q )
  Aff_val(Q, val)
  Aff_adr(Q, NIL)
  SI Liste  # NIL :
   Aff_adr(P, Q)
  SINON
   Liste := Q
  FSI
  P := Q
 FINPOUR
 { Listage }
 P := Liste
 TANTQUE P # NIL :
  ECRIRE( Valeur( P) )
  P := Suivant(P)
 FINTANTQUE
FIN
```

**2. Recherche d'un élément**

Il s'agit bien entendu de la recherche séquentielle d'une valeur donnée.

```
        ALGORITHME Rechercher
VAR
 P, Liste : POINTEUR( Typedumaillon)
 Trouv : BOOLEEN
 Val : Typeqq
DEBUT
 LIRE(Val)
 P := Liste
 Trouv := FAUX
 TANTQUE P # NIL ET NON Trouv :
  SI Valeur(P) = Val :
   Trouv := VRAI
  SINON
   P :=Suivant(P)
  FSI
 FINTANTQUE
 SI Trouv :
  ECRIRE ( " L'élément existe " )
 SINON
  ECRIRE ( " L'élément n'existe pas ")
 FSI
FIN
```

**3. Interclassement de deux listes triées**

L'algorithme qui suit construit une liste linéaire chaînée ordonnée à partir de deux listes linéaires chaînées aussi ordonnées.

```
 ALGORITHME Interclasser
 VAR
  Q, P, P1, P2, Liste : POINTEUR( Typedumaillon)
 DEBUT
  P1 := Liste1
  P2 := Liste2
  P := NIL
  Liste := NIL
  TANTQUE P1 # NIL ET P2 # NIL :
   Allouer(Q)
   Aff_adr(Q, NIL)
   SI Valeur(P1) < Valeur(P2) :
    Aff_val( Q, Valeur(P1) )
    P1 := Suivant(P1)
   SINON
    Aff_val(Q, Valeur(P2))
    P2 := Suivant(P2)
   FSI
   SI P # NIL :
    Aff_adr(P, Q)
   SINON
    Liste := Q
```

```
 FSI
 P := Q
FINTANTQUE

TANTQUE P1 # NIL :
 Allouer (Q)
 Aff_val(Q, valeur(P1)
 Aff_adr(Q, NIL)
 SI P # NIL :
  Aff_adr(P, Q)
 SINON
  Liste := Q
 FSI
 P := Q
 P1 := Suivant(P1)
FINTANTQUE

TANTQUE P2 # NIL :
 Allouer (Q)
 Aff_val(Q, valeur(P2)
 Aff_adr(Q, NIL)
 SI P # NIL :
  Aff_adr(P, Q)
 SINON
  Liste := Q
 FSI
 P := Q
 P2 := Suivant(P2)
FINTANTQUE
```

### 18. 2 Utilisation en PASCAL

On définit le type d'un maillon comme suit :

```
      TYPE Pointeur = @ T
TYPE T = RECORD
Val : Typeqq  ;
Adr : Pointeur
End
```

et les variables de type Pointeur de la façon suivante :

```
      VAR P, Q, R, ..   : @ T
```

Le langage est doté des deux opérations suivantes permettant de faire l'allocation dynamique :

*NEW*(P) et *DISPOSE*(P)

L'accès à un champ d'un maillon se fait comme suit :

Si P est l'adresse d'un maillon, P@.Val permet l'accès au champ valeur de ce maillon. P@.Adr permet l'accès au champ adresse.

**Exemple : traduction de l'algorithme de création**

```
PROGRAM Creer;
TYPE
 Typeqq = INTEGER;
 Pointeur = @Typedumaillon  ;
 Typedumaillon = RECORD
  Val : Typeqq ;
  Adr : Pointeur
 END;

  PROCEDURE Allouer( VAR P : Pointeur )    ;
 BEGIN NEW(P) END                          ;

  PROCEDURE Aff_val( VAR P : Pointeur; Val :INTEGER )   ;
 BEGIN P^.Val := Val END                                ;

  PROCEDURE Aff_adr( VAR P : Pointeur; Q : Pointeur )      ;
 BEGIN P^.Adr := Q END                                    ;

FUNCTION Suivant( P : Pointeur ) : Pointeur                ;
 BEGIN Suivant := P^.Adr END                              ;

FUNCTION Valeur( P : Pointeur ) : INTEGER                  ;
 BEGIN Valeur := P^.Val END                               ;


VAR
 P, Q, Liste : Pointeur;
 I, Nombre : INTEGER;
 Val : Typeqq
BEGIN
 Liste := NIL;
 P := NIL;
 READ(Nombre) ;
 FOR I:= 1 TO Nombre DO
  BEGIN
    READ ( Val ) ;
   Allouer(Q ) ;
   Aff_val(Q, val);
   Aff_adr(Q, NIL);
   IF Liste  # NIL
   THEN Aff_adr(P, Q)
   ELSE   Liste := Q
   P := Q
  END;
      WRITE(Fs, '+ {')
 P := Liste ;
 WHILE (P <> NIL) DO
  BEGIN
   WRITELN( Valeur(P), ',' )  ;
   L := Suivant(P)
```

```
  END  ;
 WRITELN(Fs, '}')
END.
```

## TRAVAUX DIRIGES

Développer les algorithmes suivants sur les listes linéaires chaînées :

**1**. Longueur d'une liste.
**2**. Rechercher l'élément qui a le plus grand nombre d'occurrences.
**3**. Accès par valeur.
**4**. Accès par position.
**5**. Suppression par valeur.
**6**. Suppression par position.
**7**. Insertion par position.
**8**. Tri par la méthode  "sélection et permutation".
**9**. Tri par la méthode des bulles.

**Partie 8.**

# Fichiers

## COURS 19. Introduction aux fichiers et opérations fondamentales sur les fichiers

***Objectifs :***

- fournir les raisons d'utilisation des mémoires secondaires et introduire les notions de fichiers et de structures de fichiers,

*- décrire le lien entre un fichier logique à l'intérieur d'un programme avec un fichier physique réel sur le support externe,*

*- décrire les opérations fondamentales sur des systèmes de fichiers : OUVRIR(), CREER(), FERMER(), LIRE(), ECRIRE() et POSITIONNER().*

### 19.1 Raisons de l'utilisation des mémoires secondaires

Bien que la mémoire centrale (RAM) fournit une forme d'accès simple et rapide, il existe plusieurs raisons qui nous poussent à ranger les informations sur mémoire externe plutôt qu'en RAM :

- l'espace RAM est assez limité comparé à l'espace mémoire externe,
- la RAM est beaucoup plus chère que la mémoire externe,
- la RAM est volatile( légère), la mémoire externe ne l'est pas.

### 19.2 Fichier

L'information en mémoire externe est organisée sous forme de fichiers. Un fichier peut être vu comme une collection d'octets représentant des informations. Ces octets sont organisés hiérarchiquement en structures afin de faciliter les opérations sur le fichier.

### 19.3 Structure de fichier

Le problème avec le stockage externe est la*lenteur* des accès aux données. Aussi, faut-il trouver une stratégie intelligente de rangement et de recherche de données. C'est ce que nous appelons structure de fichier (organisation imposée à un fichier afin de faciliter le traitement du fichier). L'idéal c'est d'avoir en un seul essai, l'information désirée. Ceci revient à dire que si nous désirons avoir plusieurs éléments d'informations, il faut les avoir tous à la fois plutôt que de faire des accès séparés pour chaque élément.

### 19.4 Fichiers physiques et fichiers logiques

Au niveau du programme on travaille sur un *fichier logique* ( vue abstraite). Un lien doit toujours être établi entre ce fichier logique et un *fichier physique* réel se trouvant sur un support externe. Dans certains langages, ceci est fait dans le programme (en PASCAL ceci se fait avec l'instruction *ASSIGN)* ou à l'extérieur au niveau du langage de commande. Dans d'autres langages, le lien est établi à partir de l'opération CREER ou OUVRIR. L'opération FERMER détruit le lien entre le fichier logique et le fichier physique.

### 19.5 Ouverture, création et fermeture

Ouverture : rend le fichier prêt pour son utilisation. Il y a positionnement au début du fichier.

Création : ouvre le fichier dans le sens qu'il devient prêt pour son utilisation.

Fermeture : rend l'accès impossible au fichier. Généralement la fermeture est automatique, pris en charge par le système de gestion de fichiers.

### 19.6 Lecture et écriture

Les opérations LIRE et ECRIRE exigent 3 informations :

- le nom logique du fichier
- zone mémoire interne
- une indication sur la quantité d'informations à lire ou à écrire.

Les opérations LIRE et ECRIRE sont suffisantes pour traiter le fichier séquentiellement mais cette forme d'accès est inefficace dans la plupart des cas.

### 19.7 Détection de la fin de fichier

Une autre opération utile sur les fichiers est celle de la détection de fin de fichier. Elle prend des formes variées d'un langage à un autre.

### 19.8 L'opération Positionner ( "SEEK" )

Cette opération, quand elle existe, permet de positionner la tête de lecture/écriture à une position spécifique du fichier. Les langages qui fournissent cette opération autorisent l'accès direct. Le fichier est alors vu comme un grand tableau.

### 19.9 Utilisation en PASCAL

Les principales opérations permettant de traiter un fichier en PASCAL sont les suivantes:

- RESET() : ouvre un fichier en lecture/écriture.
- REWRITE() : crée un fichier.
- CLOSE() : ferme un fichier.
- SEEK() : se positionne à un endroit précis du fichier.
- READ() : lit un enregistrement.
- WRITE() : écrit un enregistrement.

### 19.10 Exemple : programmation des opérations de base

Le programme qui suit

- crée un fichier d'enregistrements,
- le parcours ,
- lui rajoute des enregistrements en fin de fichier et
- accède directement à certains enregistrements.

```
PROGRAM Fichiers;
 TYPE
  Type_article = RECORD
   Nom  : STRING[12];
   Age  : INTEGER;
   Poids : REAL
   END;
 VAR
  Nom : STRING[12];
  Age : INTEGER;
  Poids : REAL;
  I : INTEGER;
  Article : Type_article;
  Fichier : File OF Type_article;

 BEGIN
  ASSIGN(Fichier, 'Fich.Pas');

  { Création d'un fichier avec 5 enregistrements }
  REWRITE(Fichier);
  FOR I:= 1 TO 5 DO
   BEGIN
    WRITELN('Introduire Article N_ ', I );
    WRITE('Nom ?'); READLN (Nom);
    WRITE('Age ?'); READLN (Age);
    WRITE('Poids ?'); READLN (Poids);
    Article.Nom := Nom;
    Article.Age := Age;
    Article.Poids := Poids;
    WRITE(Fichier, Article)
   END;

  { Parcours du fichier créé et listage de ses éléments }
  RESET(Fichier);
  WHILE NOT EOF(Fichier) DO
   BEGIN
    READ(Fichier, Article);
    WRITELN( Article.Nom, ' ',Article.Age, ' ', Article.Poids:4:2);
    READLN;
   END;

  { Positionnement à la fin du fichier et rajout de deux articles }
  SEEK(Fichier, Filesize(Fichier));
  FOR I:= 1 TO 2 DO
   BEGIN
    WRITELN('Introduire Un Autre Article' );
    WRITE('Nom ?'); READLN (Nom);
```

```
    WRITE('Age ?'); READLN (Age);
    WRITE('Poids ?'); READLN (Poids);
    Article.Nom := Nom;
    Article.Age := Age;
    Article.Poids := Poids;
    WRITE(Fichier, Article)
  END;

 { Récupération du troisième enregistrement }
 SEEK(Fichier, 2);
 READ(Fichier, Article);
 WRITELN( Article.Nom, ' ',Article.Age, ' ', Article.Poids:4:2);
 READLN;

 { Récupération du septième enregistrement }
 SEEK(Fichier, 6);
 READ(Fichier, Article);
 WRITELN( Article.Nom, ' ',Article.Age, ' ', Article.Poids:4:2);
 READLN;
END.
```

# COURS 20. Concepts fondamentaux des structures de fichiers

***Objectifs :***

*Introduire les concepts de structures de fichiers suivants :*
*- fichier continu ( stream en anglais ),*
*- limites des champs et des articles,*
*- articles et champs de longueur fixe et variable,*
*- recherche séquentielle,*
*- accès direct,*
*- accès au fichier et organisation du fichier.*

## 20.1 Fichiers continus ( Stream )

Le niveau le plus bas d'organisation de fichier est de considérer le fichier comme une *suite continue de caractères* (fichier "Stream").

Exemple : lecture d'un fichier TEXT PASCAL.

Le fichier est une suite continue de caractères (octets). Aucune information supplémentaire est ajoutée pour distinguer les agrégats. ces derniers sont appelés champs. Un champ est la plus petite unité logique d'information dans un fichier. Un champ est une notion logique, c'est un outil conceptuel.

## 20.2 Structures des champs

On peut délimiter les champs de plusieurs façons :
- fixer la longueur du champ,
- commencer chaque champ par un indicateur de longueur,
- placer un délimiteur à la fin de chaque champ pour le séparer du prochain.

## 20.3 Structures des articles

Un article est un ensemble de champs formant une même entité quand le fichier est vu en terme d'un haut niveau d'organisation.

Un article est un outil logique conceptuel.

Les structures possibles d'un article peuvent être comme suit :

- fixer la longueur de l'article en nombre d'octets ou en nombre de champs,
- commencer chaque article avec un indicateur de longueur ( nombre d'octets ),
- utiliser un second fichier (index) pour garder la trace des adresses octets de chaque article,
- placer un délimiteur à la fin de chaque article pour le séparer du prochain article.

## 20.4 L'accès séquentiel

Il s'agit de rechercher un élément en commençant par le début du fichier. L'algorithme utilise alors une boucle avec comme critère d'arrêt : fin de fichier rencontrée ou élément trouvé.

Le critère principal dévaluation est le nombre de lecture ( nombre d'accès au disque ).

Si le fichier possède n articles, il faut au plus n accès pour retrouver un élément et en moyenne n/2 accès. Une recherche séquentielle est aussi dite de l'ordre de O(n), c'est à dire proportionnelle à n.

**"Blocage" des articles**

Une façon d'améliorer la recherche séquentielle est de regrouper plusieurs articles par bloc, c'est ce qu'on appelle blocage des articles.

**Un bloc** peut être perçu comme une collection d'articles rangés comme une unité physiquement contiguë.

Sur le disque, l'opération de positinnement du bras (SEEK') est beaucoup plus lente que le transfert lui même. Le coût de positionnement du bras pour un article et de son transfert, puis le positionnement du bras d'un autre article et de son transfert est plus grand que celui du positionnement du bras juste une fois suivi du transfert des 2 articles à la fois.
On peut donc améliorer le coût de la recherche en lisant dans un bloc plusieurs articles à la fois, puis traiter le bloc d'articles en RAM.

Comme exemple, si on a fichier de 1000 articles, une recherche séquentielle non bufferisée exige en moyenne 5OO lectures. En utilisant des buffers de 32 articles, le nombre moyen de lectures devient égal à 15. Chaque lecture exige alors un peu plus de temps, puisque plus de données sont transférées, ce qui meilleur que des lectures individuelles pour chaque article.
On pourra alors parler de lecture physique et de lecture logique.

## 20.5 L'accès direct

Le plus simple format d'article permettant l'accès direct par numéro relatif ( déplacement) implique l'utilisation des articles de longueur fixe. Quand la donnée elle-même est composée de quantité de taille fixe ( par exemple des codes), les articles de longueur fixe fournissent de bonne performance et une bonne utilisation de l'espace.
A chaque article est associé un numéro relatif de l'article dans le fichier. Ce dernier est considéré comme un tableau d'articles. Le premier article a un numéro égal à 0, le second 1, etc.....

Si r est la taille d'un article (en octets), le déplacement octet de l'article de numéro n est : $d = nr$

Si la taille des données dans un article est variable, beaucoup d'espace sera perdu. Dans cette situation, le concepteur doit analyser le problème avec prudence sur la possibilité d'utiliser des articles de longueur variable.

## 20.6 Article d'en-tête

Quelquefois, il est utile de garder la trace de certaines informations du fichier telles que le nombre d'articles, date de création, dernière version, etc... Un *article d'en-tête* est alors utilisé.

C'est un article placé au début du fichier donnant des informations sur le fichier permettant de l'exploiter.

### 20.7 Accès aux fichiers et organisation de fichier.

Il est important de savoir la difference entre l'*accès* au fichier et l'*organisation* du fichier.

Méthode d'accès du fichier : approche utilisée pour localiser un article dans le fichier. En général, deux alternatives : l'accès séquentiel et l'accès direct.

Méthode d'organisation du fichier : Combinaisons de structures conceptuelles et physiques utilisées pour distinguer un champ d'un autre et un article d'un autre. Un exemple de sorte d'organisation d'un fichier est : article de longueur fixe contenant un nombre variable de champs.

### 20.8 Exemple d'utilisation en PASCAL

Le programme PASCAL qui suit réalisent les opérations suivantes :
- Création
- Ouverture
- Fermeture
- Insertion de clés générées aléatoirement
- Parcours séquentiel et listage
- Recherche séquentielle d'articles

Les résultats de ce programme sont également fournis.

```
PROGRAM Fichiers;
 CONST
   B = 10;
 TYPE
   Typecle = INTEGER  ;
   Typeqq = STRING[20] ;
   Nature = (Inf, Table);
   Typearticle = RECORD
     Cle : Typecle;
     Info : Typeqq
   END;
   Typebloc = RECORD
     CASE Diff : Nature OF
       1 : (   Nombre : INTEGER;
             Tab : ARRAY[1..B] OF Typearticle );
       2 : (   Nombre_elements, Efface, Dernierblocseq, Dernierseq: INTEGER )
     END;

 VAR
   Fs : TEXT;
   I : INTEGER;
   Efface, Nombre_elements : INTEGER;
```

```
  F : File OF Typebloc;
  Dernierseq,Dernierblocseq : INTEGER;
  Bloc : Typebloc;

{ Création du fichier }

PROCEDURE Creation;
  BEGIN
    Bloc.Diff := Inf;
    Bloc.Nombre_elements := 0;
    Bloc.Efface := 0;
    Bloc.Dernierblocseq := 1;
    Bloc.Dernierseq := 0;
    REWRITE(F);
    SEEK(F, 0);
    WRITE(F, Bloc);
    CLOSE(F)
  END;

{ Ouverture du fichier }

PROCEDURE Ouvrir;
  BEGIN
    Bloc.Diff := Inf;
    RESET(F);
    SEEK(F, 0);
    READ(F, Bloc);
    Efface := Bloc.Efface;
    Nombre_elements := Bloc.Nombre_elements;
    Dernierblocseq := Bloc.Dernierblocseq;
    Dernierseq := Bloc.Dernierseq;
  END;

{ Fermeture du fichier }

PROCEDURE Fermer;
  BEGIN
    Bloc.Diff := Inf;
    Bloc.Nombre_elements := Nombre_elements;
    Bloc.Efface := Efface;
    Bloc.Dernierblocseq := Dernierblocseq;
    Bloc.Dernierseq := Dernierseq;
    RESET(F);
    SEEK(F, 0);
    WRITE(F, Bloc);
  END;

{ Parcours séquentiel du fichier et listage de ses éléments sur un fichier TEXT }
PROCEDURE Impseq;
  VAR
    I, J : INTEGER;
  BEGIN
```

```
  RESET(F);
  { Les caractéristiques }
  WRITELN(Fs, 'Caractéristiques du fichier : ');
  WRITELN(Fs);
  WRITELN(Fs, '- Nombre d''articles : ', Nombre_elements );
  WRITELN(Fs, '- Nombre d''articles effacés :', Efface);
  WRITELN(Fs, '- Dernier bloc :', Dernierblocseq);
  WRITELN(Fs, '- Dernier indice dans le dernier bloc :', Dernierseq);
  WRITELN(Fs);
  WRITELN(Fs);
  WRITELN(Fs, 'Contenu du fichier : ');

  FOR I:=2 TO Dernierblocseq DO
   BEGIN
    SEEK(F, I-1);
    READ(F, Bloc);
    WRITELN(Fs);
    WRITELN(Fs, 'Bloc ',I, ' :');
    FOR J:=1 TO Bloc.Nombre DO
     WRITE(Fs, Bloc.Tab[J].Cle, '  ' );
    WRITELN(Fs);
   END;
  END;

 { Insertion en fin de fichier }

 PROCEDURE Add (Cle : INTEGER);
  BEGIN
   Nombre_elements := Nombre_elements + 1 ;
   Inc(Dernierseq);
   IF Dernierseq <= B
   THEN
    BEGIN
     Bloc.Tab[Dernierseq].Cle:= Cle;
     Inc(Bloc.Nombre)
    END
   ELSE
    BEGIN
     Inc(Dernierblocseq);
     SEEK(F, Dernierblocseq - 1);
     WRITE(F, Bloc);
     Bloc.Tab[1].Cle := Cle;
     Bloc.Nombre := 1;
     Dernierseq := 1;
    END;
  END;
{ Recherche séquentielle d'un article de clé donnée }

FUNCTION Recherche (Clef : INTEGER) : BOOLEAN;
 VAR
  I, J : INTEGER;
  Trouv : BOOLEAN;
```

```
 BEGIN
  WRITELN(Fs, ' Recherche de ', Clef, ' ...');
  RESET(F);
  I:= 2;
  Trouv := FALSE;
  WHILE  (I <= Dernierblocseq ) AND NOT Trouv DO
   BEGIN
    SEEK(F, I-1);
    READ(F, Bloc);
    J:= 1;
    WHILE (J <=Bloc.Nombre) AND NOT Trouv DO
     IF Bloc.Tab[J].Cle = Clef
     THEN Trouv := TRUE
     ELSE J := J + 1;
    I := I + 1
   END;
   Recherche := Trouv;
 END;

BEGIN
 ASSIGN(Fs, 'R_F1.Pas');
 REWRITE(Fs);
 ASSIGN(F, 'Fichier');
 WRITELN(Fs, 'Création du fichier...');
 WRITELN(Fs);
 Creation;
 WRITELN(Fs, 'Ouverture du fichier...');
 WRITELN(Fs);
 Ouvrir;

 WRITELN(Fs, 'Insertion de 55 articles de clés générées aléatoirement...');
 WRITELN(Fs);
 Bloc.Diff := Table;
 Bloc.Nombre := 0;
 FOR I:=1 TO 55 DO
  Add( Random(1000) );

 { Ecriture Dernier Bloc }
 IF Dernierseq > 0
 THEN
  BEGIN
   Inc(Dernierblocseq);

   SEEK(F, Dernierblocseq - 1);
   WRITE(F, Bloc);
  END;
 CLOSE(F);

 { Parcours séquentiel du fichier et listage de ses éléments }
 Impseq;

 WRITELN(Fs);
```

```
   WRITELN(Fs, 'Recherche d''articles...');
   WRITELN(Fs);
   IF Recherche(993)
   THEN WRITELN(Fs, 'Article existe')
   ELSE WRITELN(Fs, 'Article n''existe pas');

   IF Recherche(374)
   THEN WRITELN(Fs, 'Article existe')
   ELSE WRITELN(Fs, 'Article n''existe pas');

   IF Recherche(164)
   THEN WRITELN(Fs, 'Article existe')
   ELSE WRITELN(Fs, 'Article n''existe pas');

   IF Recherche(858)
   THEN WRITELN(Fs, 'Article existe')
   ELSE WRITELN(Fs, 'Article n''existe pas');

   IF Recherche(402)
   THEN WRITELN(Fs, 'Article existe')
   ELSE WRITELN(Fs, 'Article n''existe pas');

   IF Recherche(616)
   THEN WRITELN(Fs, 'Article existe')
   ELSE WRITELN(Fs, 'Article n''existe pas');

   WRITELN(Fs);
   WRITELN(Fs, 'Fermeture du fichier...');
   Fermer;
   CLOSE(Fs);
 END.
```

**Contenu du fichier R_F1.PAS :**

```
Création du fichier...

Ouverture du fichier...

Insertion de 55 articles de clés générées aléatoirement...

Caractéristiques du fichier :

- Nombre d'articles : 55
- Nombre d'articles effacés :0
- Dernier bloc :7
- Dernier indice dans le dernier bloc :5

Contenu du fichier :

Bloc 2 :
0  56  429  276  886  17  885  603  395  896
```

Bloc 3 :
374 116 624 106 914 221 115 111 768 490

Bloc 4 :
722 323 53 96 657 593 541 164 111 286

Bloc 5 :
573 776 300 828 643 993 278 191 216 330

Bloc 6 :
244 404 820 420 858 632 756 641 494 406

Bloc 7 :
536 502 635 853 616

Recherche d'articles...

Recherche de 993 ...
Article existe
Recherche de 374 ...
Article existe
Recherche de 164 ...
Article existe
Recherche de 858 ...
Article existe
Recherche de 402 ...
Article n'existe pas
Recherche de 616 ...
Article existe

Fermeture du fichier...

# COURS 21. Maintenance de fichiers

***Objectifs** : Définir les notions de réorganisation statique et dynamique comme des moyens de réutiliser l'espace dans un fichier.*

## 21.1 Maintenance du fichiers

Le concepteur doit définir :
- la méthode d'accès et
- l'organisation du fichier.

Le concepteur doit aussi définir les sortes de *changement* que le fichier subit pendant sa vie.

Si le fichier est très *volatile* ( ou *dynamique*, c'est à dire qui subit beaucoup d'insertions et de suppressions ) et est utilisé dans un environnement temps-réel, l'organisation du fichier devrait faciliter les changements rapides. Un fichier de réservation dans un système de réservation on line est un exemple de fichier volatile utilisé en temps réel.

A l'autre extrême, on a des *fichiers off-line* qui subissent très peu de changements et n'exigent pas d'être à jour momentanément. Dans ses fichiers, on n'a pas besoin d'inclure des extra-structures pour permettre les changements rapides. Comme exemple, on peut citer un fichier de courriers.

La maintenance d'un fichier est importante car, en général, les performances se détériorent quand les changements sont faits sur le fichier. par exemple, une modification d'un article dans un fichier de longueur variable est telle que le nouveau article est de taille plus grande. Que faire ?

- On pourra utiliser un chaînage vers la fin du fichier, puis ajouter l'information supplémentaire.

- On pourra réécrire tout l'article à la fin du fichier ( à moins que le fichier est ordonné) laissant ainsi un "trou" dans l'emplacement original de l'article.

Les modifications sont de 3 formes :

- ajout d'un article,
- mise à jour d'un article,
- suppression d'un article.

Si le seul changement dans le fichier est un ajout d'articles, il n'y a pas de détérioration. C'est seulement quand des articles de longueur variable sont modifiés, ou quand des articles de longueur fixe ou variable sont supprimés que la maintenance du fichier devient compliquée et intéressante. Comme une mise à jour peut toujours être traitée comme une suppression suivie d'un ajout, notre intérêt portera uniquement sur la suppression d'articles. Quand un article est supprimé, nous devons réutiliser l'espace. Regardons de plus près comment récupérer cet espace.

## 21.2 Réorganisation

Quand on supprime un article, on place généralement une marque spéciale .

Maintenant, comment récupérer l'espace ?

Les articles ainsi effacés ( *effacement logique* ), restent dans cet état une période de temps. On pourra alors les récupérer.

Un programme spécial de réorganisation ou de compactage est alors utilisé pour reconstruire le fichier sans les articles effacés (*effacement physique*). Le critère de lancement peut être le nombre d'articles effacés ou selon un calendrier.

Pour certaines applications, on aimerait récupérer l'espace le plus tôt possible ( fichiers dynamiques , temps réel, ..). C'est ce qu'on appelle réorganisation dynamique.

Il existe des méthodes qui permettent la récupération dynamique des articles effacés pour un fichier où les articles sont de longueur fixe ou variables. Ces techniques utilisent les concept de *liste linéaire chaînée.*

### 21.3 Exemple : programmation du module de réorganisation

Le programme qui suit
- ouvre un fichier existant ( fichier créé par le programme précédent)
- rajoute des articles en fin de fichier
- suppression logiquement certains articles
- ferme le fichier
- réorganise le fichier par la construction d'un autre.

Les résultats de ce programme sont également fournis.

```
PROGRAM Fichier2;
 CONST
   B = 10;
 TYPE
   Typecle = INTEGER  ;
   Typeqq = STRING[20] ;
   Nature = (Inf, Table);
   Typearticle = RECORD
     Cle : Typecle;
     Info : Typeqq
   END;
   Typebloc = RECORD
     CASE Diff : Nature OF
       1 : (   Nombre : INTEGER;
             Tab : ARRAY[1..B] OF Typearticle );
       2 : (   Nombre_elements, Efface, Dernierblocseq, Dernierseq: INTEGER )
     END;

 VAR
   Fs : TEXT;
   Efface, Nombre_elements : INTEGER;
   F : File OF Typebloc;
   Dernierseq,Dernierblocseq : INTEGER;
   Bloc : Typebloc;
```

```
{ Ouverture du fichier }

PROCEDURE Ouvrir;
 BEGIN
   Bloc.Diff := Inf;
   RESET(F);
   SEEK(F, 0);
   READ(F, Bloc);
   Efface := Bloc.Efface;
   Nombre_elements := Bloc.Nombre_elements;
   Dernierblocseq := Bloc.Dernierblocseq;
   Dernierseq := Bloc.Dernierseq;
 END;

{ Fermeture du fichier }

PROCEDURE Fermer;
 BEGIN
   Bloc.Diff := Inf;
   Bloc.Nombre_elements := Nombre_elements;
   Bloc.Efface := Efface;
   Bloc.Dernierblocseq := Dernierblocseq;
   Bloc.Dernierseq := Dernierseq;
   RESET(F);
   SEEK(F, 0);
   WRITE(F, Bloc);
 END;

{ Parcours séquentiel du fichier et listage de ses éléments sur un fichier TEXT }

PROCEDURE Impseq;
 VAR
   I, J : INTEGER;
 BEGIN
   RESET(F);
   { Les caractéristiques }
   WRITELN(Fs, '* * * Caractéristiques du fichier : ');
   WRITELN(Fs);
   WRITELN(Fs, '- Nombre d''articles : ', Nombre_elements );
   WRITELN(Fs, '- Nombre d''articles effacés :', Efface);
   WRITELN(Fs, '- Dernier bloc :', Dernierblocseq);
   WRITELN(Fs, '- Dernier indice dans le dernier bloc :', Dernierseq);
   WRITELN(Fs);
   WRITELN(Fs, '* * * Contenu du fichier : ');

   FOR I:=2 TO Dernierblocseq DO
     BEGIN
       SEEK(F, I-1);
       READ(F, Bloc);
       WRITELN(Fs);
       WRITELN(Fs, 'Bloc ',I, ' :');
```

```
      FOR J:=1 TO Bloc.Nombre DO
        WRITE(Fs, Bloc.Tab[J].Cle, '  ' );
      WRITELN(Fs);
    END;
  END;

{ Recherche séquentielle d'un article de clé donnée }

FUNCTION Recherche (Clef : INTEGER) : BOOLEAN;
 VAR
  I, J : INTEGER;
  Trouv : BOOLEAN;
 BEGIN
  RESET(F);
  I:= 2;
  Trouv := FALSE;
  WHILE  (I <= Dernierblocseq ) AND NOT Trouv DO
   BEGIN
    SEEK(F, I-1);
    READ(F, Bloc);
    J:= 1;
    WHILE (J <=Bloc.Nombre) AND NOT Trouv DO
     IF Bloc.Tab[J].Cle = Clef
     THEN Trouv := TRUE
     ELSE J := J + 1;
    I := I + 1
   END;
   Recherche := Trouv;
 END;

{ Ajouter un article à un fichier }

PROCEDURE Ajouter (Cle : INTEGER);
 BEGIN
  WRITELN(Fs, 'Ajout de l''article ', Cle, '...');
  Nombre_elements := Nombre_elements + 1 ;
  Inc(Dernierseq);
  IF Dernierseq <= B
  THEN
   BEGIN
    SEEK(F, Dernierblocseq - 1);
    READ(F, Bloc);
    Bloc.Tab[Dernierseq].Cle:= Cle;
    Inc(Bloc.Nombre);
    SEEK(F, Dernierblocseq - 1);
    WRITE(F, Bloc);
   END
  ELSE
   BEGIN
    Inc(Dernierblocseq);
    Bloc.Tab[1].Cle := Cle;
    Bloc.Nombre := 1;
```

```
      Dernierseq := 1;
      SEEK(F, Dernierblocseq - 1);
      WRITE(F, Bloc);
    END;
 END;

{ Suppression d'un article de clé donnée }

PROCEDURE Supprimer (Clef : INTEGER) ;
 VAR
  I, J : INTEGER;
  Trouv : BOOLEAN;
 BEGIN
  WRITELN(Fs, 'Supprimer l''article ', Clef, ' ...');
  RESET(F);
  I:= 2;
  Trouv := FALSE;
  WHILE  (I <= Dernierblocseq ) AND NOT Trouv DO
   BEGIN
    SEEK(F, I-1);
    READ(F, Bloc);
    J:= 1;
    WHILE (J <=Bloc.Nombre) AND NOT Trouv DO
     IF Bloc.Tab[J].Cle = Clef
     THEN
      BEGIN
       Bloc.Tab[J].Cle := - 1;
       SEEK(F, I-1);
       WRITE(F, Bloc);
       Efface := Efface + 1;
       Trouv := TRUE
      END
     ELSE J := J + 1;
    I := I + 1
   END;
  IF NOT Trouv THEN WRITELN(Fs, 'Article inexistant');
 END;

{ Réorganisation }

PROCEDURE Reorganiser;
 VAR
  I, J : INTEGER;
  Dernierbloc, Dernier : INTEGER;
  Fnouv : File OF Typebloc;
  Blocnouv : Typebloc;

  PROCEDURE Add (Cle : INTEGER);
   BEGIN
    Inc(Dernier);
    IF Dernier <= B
    THEN
```

```
        BEGIN
          Blocnouv.Tab[Dernier].Cle:= Cle;
          Inc(Blocnouv.Nombre)
        END
      ELSE
        BEGIN
          Inc(Dernierbloc);
          SEEK(Fnouv, Dernierbloc - 1);
          WRITE(Fnouv, Blocnouv);
          Blocnouv.Tab[1].Cle := Cle;
          Blocnouv.Nombre := 1;
          Dernier := 1;
        END;
    END;

BEGIN
  Blocnouv.Diff := Table;
  Blocnouv.Nombre := 0;
  Dernierbloc := 1;
  Dernier := 0;
  ASSIGN(Fnouv, 'Fnouveau');
  REWRITE(Fnouv);

  RESET(F);
  FOR I:=2 TO Dernierblocseq DO
    BEGIN
      SEEK(F, I-1);
      READ(F, Bloc);
      FOR J:=1 TO Bloc.Nombre DO
        IF Bloc.Tab[J].Cle <> - 1
        THEN  Add ( Bloc.Tab[J].Cle )
    END;

  { Ecriture Dernier Bloc }
  IF Dernier > 0
  THEN
    BEGIN
      Inc(Dernierbloc);
      SEEK(Fnouv, Dernierbloc - 1);
      WRITE(Fnouv, Blocnouv);
    END;
  Blocnouv.Diff := Inf;
  Blocnouv.Dernierseq := Dernier;
  Blocnouv.Dernierblocseq := Dernierbloc;
  Blocnouv.Efface := 0;
  Blocnouv. Nombre_elements :=  Nombre_elements  - Efface;

  WRITELN(Fs);
  WRITELN(Fs, '* * * Caractéristiques du nouveau fichier : ');
  WRITELN(Fs);
  WRITELN(Fs, '- Nombre d''articles : ', Blocnouv.Nombre_elements );
  WRITELN(Fs, '- Nombre d''articles Effacés :', Blocnouv.Efface);
```

```
    WRITELN(Fs, '- Dernier bloc :', Blocnouv.Dernierblocseq);
    WRITELN(Fs, '- Dernier indice dans le dernier bloc :', Blocnouv.Dernierseq);
    WRITELN(Fs);
    WRITELN(Fs, '* * * Contenu du nouveau fichier : ');

    SEEK(Fnouv, 0);
    WRITE(Fnouv, Blocnouv);
    CLOSE(Fnouv);
    RESET(Fnouv);

    FOR I:=2 TO Dernierbloc DO
      BEGIN
        SEEK(Fnouv, I-1);
        READ(Fnouv, Blocnouv);
        WRITELN(Fs, 'Bloc ',I, ' :');

        FOR J:=1 TO Blocnouv.Nombre DO
          WRITE(Fs, Blocnouv.Tab[J].Cle, '  ' );
        WRITELN(Fs);
      END;
  END;

BEGIN
  ASSIGN(Fs, 'R_F2.Pas');
  REWRITE(Fs);
  ASSIGN(F, 'Fichier');
  WRITELN(Fs, 'Ouverture du fichier...');
  WRITELN(Fs);
  Ouvrir;
  Bloc.Diff := Table;

  WRITELN(Fs, 'Rajout d''articles en fin de fichier...');

  IF NOT( Recherche(333) ) THEN Ajouter(333);
  IF NOT( Recherche(444)) THEN  Ajouter(444);
  WRITELN(Fs);

  WRITELN(FS, 'Parcours séquentiel du fichier et listage de ses éléments...');
  WRITELN(Fs);
  Impseq;

  WRITELN(Fs);
  WRITELN(Fs, 'Suppression d''articles...');
  WRITELN(Fs);

  Supprimer(993);
  Supprimer(374);
  Supprimer(164);
  Supprimer(858);
  Supprimer(402);
  Supprimer(616);
```

```
   WRITELN(Fs);
   WRITELN(FS, 'Parcours séquentiel du fichier et listage de ses éléments...');
   WRITELN(Fs);
   Impseq;
   WRITELN(Fs, 'Fermeture du fichier...');
   WRITELN(Fs);
   Fermer;

   WRITELN(Fs, 'Reorganisation...');
   Reorganiser;
   WRITELN(Fs);
   CLOSE(Fs)
 END.
```

**Contenu du fichier R_F2.PAS :**

```
Ouverture du fichier...

Rajout d'articles en fin de fichier...
Ajout de l'article 333...
Ajout de l'article 444...

Parcours séquentiel du fichier et listage de ses éléments...

* * * Caractéristiques du fichier :

- Nombre d'articles : 57
- Nombre d'articles effacés :0
- Dernier bloc :7
- Dernier indice dans le dernier bloc :7

* * * Contenu du fichier :

Bloc 2 :
0  56  429  276  886  17  885  603  395  896

Bloc 3 :
374  116  624  106  914  221  115  111  768  490

Bloc 4 :
722  323  53  96  657  593  541  164  111  286

Bloc 5 :
573  776  300  828  643  993  278  191  216  330

Bloc 6 :
244  404  820  420  858  632  756  641  494  406

Bloc 7 :
536  502  635  853  616  333  444

Suppression d'articles...
```

Supprimer l'article 993 ...
Supprimer l'article 374 ...
Supprimer l'article 164 ...
Supprimer l'article 858 ...
Supprimer l'article 402 ...
Article inexistant
Supprimer l'article 616 ...

Parcours séquentiel du fichier et listage de ses éléments...

* * * Caractéristiques du fichier :

- Nombre d'articles : 57
- Nombre d'articles effacés :5
- Dernier bloc :7
- Dernier indice dans le dernier bloc :7

* * * Contenu du fichier :

Bloc 2 :
0 56 429 276 886 17 885 603 395 896

Bloc 3 :
-1 116 624 106 914 221 115 111 768 490

Bloc 4 :
722 323 53 96 657 593 541 -1 111 286

Bloc 5 :
573 776 300 828 643 -1 278 191 216 330

Bloc 6 :
244 404 820 420 -1 632 756 641 494 406

Bloc 7 :
536 502 635 853 -1 333 444
Fermeture du fichier...

Reorganisation...

* * * Caractéristiques du nouveau fichier :

- Nombre d'articles : 52
- Nombre d'articles Effacés :0
- Dernier bloc :7
- Dernier indice dans le dernier bloc :2

* * * Contenu du nouveau fichier :
Bloc 2 :
0 56 429 276 886 17 885 603 395 896
Bloc 3 :
116 624 106 914 221 115 111 768 490 722

Bloc 4 :
323 53 96 657 593 541 111 286 573 776
Bloc 5 :
300 828 643 278 191 216 330 244 404 820
Bloc 6 :
420 632 756 641 494 406 536 502 635 853
Bloc 7 :
333 444

# COURS 22. Tri de fichiers

***Objectif** : décrire la manière de trier un fichier en présentant l'algorithme le plus usité qu'est le tri par fusion.*

## 22. 1 Tri de tout le fichier entièrement en mémoire

Cette méthode est utilisée quand la taille du fichier n'est pas importante. Le fichier peut alors être entièrement contenu en mémoire centrale.

La technique consiste donc à charger entièrement le fichier en mémoire dans un vecteur, puis réaliser le tri. Le tri se fait bien entendu uniquement sur la clé.

L'algorithme pourrait être le suivant :

*Lire les articles du fichier à trier dans le tableau ARTICLE*
*Extraire les clés des articles afin de construire la table CLE*
*Former le tableau INDEX des indices*
*Trier le tableau INDEX en se basant sur le tableau CLE*
*Construire le fichier trié à partir du vecteur INDEX*

Illustrons la technique à travers un exemple

**Le fichier de départ :**

654 aaaa...
324 bbbbbb...
653 cccc...
123 ddddd...
876 eee.....
112 fffff.....

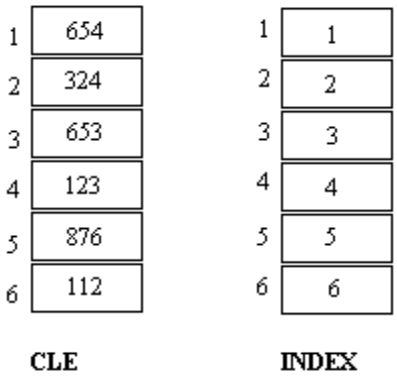
Les valeurs numériques désignent les clés.

**Lecture des articles du fichier à trier dans le tableau ARTICLE**

| | | |
|---|---|---|
| 1 | 654 | aaaa.... |
| 2 | 324 | bbbbbb..... |
| 3 | 653 | cccc..... |
| 4 | 123 | ddddd..... |
| 5 | 876 | eee..... |
| 6 | 112 | fffff..... |

**ARTICLE**

**Extraction des clés des articles afin de construire la table CLE et formation du tableau INDEX des indices**




**Tri du tableau INDEX en se basant sur le tableau CLE**


**Le fichier trié :**

112 fffff.....
123 ddddd...
324 bbbbbb...
653 cccc...
654 aaaa...
876 eee.....

### 22. 2 Tri entièrement en mémoire uniquement des clés du fichier

On ne charge que les clés du fichier avant de réaliser le tri entièrement en mémoire.

L'algorithme est le suivant :

*Récupérer toutes les clés dans le tableau CLE*
*Former le tableau INDEX des indices*
*Trier le vecteur Index en se basant sur le tableau CLE*
*Récupérer l'article du fichier de données pour l'écrire sur le fichier résultats*

L'inconvénient de cette technique réside dans les positionnements ("SEEK") aléatoires des têtes de lecture-écriture dus aux deuxièmes lectures obligatoires des articles. L'avantage par rapport à la première méthode présentée est que l'on peut trier des fichiers de tailles beaucoup plus importantes.

### 22. 3 Tri par fusion

C'est la technique la plus générale. Elle est utilisée pour trier les grands fichiers.

Si on dispose d'un méga RAM, avec des articles de 100 octets, on pourra trier en mémoire centrale des portions de

1000 000 octets RAM / 100 octets = 10 000 articles

On peut énoncer l'algorithme comme suit :

1. *Lire les articles en RAM dans un tableau tantqu'il y a de la place*
2. *Trier ce tableau*
3. *Le mettre dans un fichier*

*Répéter ces trois opérations autant de fois qu'il faut.*

*On construit ainsi un ensemble de n fichiers triés.*

*Appliquer l'algorithme de fusion à ces n fichiers.*

Dans l'exemple précédent, pour un fichier de 400 000, on construit 40 fichiers triés.

**Avantages**

- Chaque article est lu seulement une seule fois
- Quelque soit la taille du fichier, on peut appliquer cette méthode
- La lecture du fichier d'entrée est complètement séquentielle, ce qui est beaucoup plus avantageux que la deuxième méthode exposée.
- Les lectures durant la phase de fusion et les écritures sont aussi séquentielles.
- On utilise les opérations de positionnement ('SEEK') uniquement quand on charge un buffer.
- On peut même mettre les fichiers d'entrée et de sortie sur des bandes magnétiques.

Supposons que la taille d'un fichier soit réservé pour tous les buffers utilisés.

40 fichiers * 10 SEEK/fichier = 1600 SEEK

Comme il y a 40 fichiers, on aura 40 buffers de 1/40 de la taille du fichier.

**Amélioration**

On effectue la fusion sur plusieurs étapes en réduisant le degré de fusion et en incrémentant la taille des buffers.

Au lieu de fusionner les 40 fichiers d'un seul coup on procède autrement.

On fait par exemple une fusion 8:8:8:8:8

Une fois qu'on a construit les 40 fichiers initiaux, On applique 5 fois la fusion pour construire donc 5 fichiers intermédiaires. Enfin, on applique une seule fusion sur ces fichiers pour obtenir le fichier désiré.

L'inconvénient de cette méthode est que chaque article est lu deux fois. Par contre, l'avantage est que l'on peut utiliser de grands buffers, ce qui permet de réduire les opérations de positionnement ('SEEK').

Pour chacune des 5 fusions ( sur 8 fichiers), on utilise 8 buffers de 1/8 de la taille du fichier.
Donc 8 SEEK/Fichier * 8 Fichiers = 64 SEEK.
Pour toutes les fusions on aura 64 * 5 = 320 SEEK.

Chaque fichier intermédiaire est 8 fois plus grand que le fichier initial.

Chacun des 5 buffers d'entrée peut contenir 1/40 de la taille d'un fichier intermédiaire.
Donc 5 Fichiers * 40 SEEK/Fichiers = 200 SEEK.

Le nombre total de SEEK est alors 320 + 200 = 520.

En acceptant de lire deux fois le même article, on réduit le nombre de SEEK par 2/3.

Si nous prenons un fichier de 800 000 articles, on aura 80 fichiers initiaux, ce qui nous fait 6400 SEEK.

En utilisant une fusion 10:10:10:10:10:10:10:10, on réduira la nombre de SEEK à 640 + 800 = 1440 SEEK!

### 22.4 Exemple :Programmation du tri par fusion

Dans ce qui suit, nous présentons deux programmes. Le premier construit n fichiers triés à partir d'un fichier non trié. Le second fusionne ces n fichiers.

Nous fournissons également les résultats détaillés pour les programmes en question.

**1. Premier programme : construction de n fichiers triés à partir d'un fichier non trié**

```
PROGRAM Fichier3;
 CONST
   B = 10;
 TYPE
   Typecle = INTEGER  ;
   Typeqq = STRING[20] ;
   Nature = (Inf, Table);
   Typearticle = RECORD
     Cle : Typecle;
     Info : Typeqq
   END;
   Typebloc = RECORD
     CASE Diff : Nature OF
       1 : (   Nombre : INTEGER;
             Tab : ARRAY[1..B] OF Typearticle );
       2 : (   Nombre_elements, Efface, Dernierblocseq, Dernierseq: INTEGER )
     END;
```

```
  Typefichier = File OF Typebloc;
VAR
  Fs : TEXT;
  I, J : INTEGER;
  Efface, Nombre_elements : INTEGER;
  F : File OF Typebloc;
  Dernierseq,Dernierblocseq : INTEGER;
  Bloc : Typebloc;

  F1 : Typefichier;
  Dernierseq1,Dernierblocseq1 : INTEGER;
  Bloc1 : Typebloc;

  Depl, Numbloc : INTEGER;
  Temp, S : Typearticle;
  Tab : ARRAY[1..20] OF Typearticle;
  K : INTEGER;
  Num : INTEGER;

{ Ouverture du fichier }

PROCEDURE Ouvrir;
  BEGIN
    Bloc.Diff := Inf;
    RESET(F);
    SEEK(F, 0);
    READ(F, Bloc);
    Efface := Bloc.Efface;
    Nombre_elements := Bloc.Nombre_elements;
    Dernierblocseq := Bloc.Dernierblocseq;
    Dernierseq := Bloc.Dernierseq;
  END;

{ Parcours séquentiel du fichier et listage de ses éléments sur un fichier TEXT }

PROCEDURE Impseq;
  VAR
    I, J : INTEGER;
  BEGIN
    RESET(F1);
    { Les caractéristiques }
    WRITELN(Fs, ' + Caractéristiques du fichier : ');
    WRITELN(Fs);
    WRITELN(Fs, '  >> Nombre d''articles : ', Nombre_elements );
    WRITELN(Fs, '  >> Nombre d''articles effacés :', Efface);
    WRITELN(Fs, '  >> Dernier bloc :', Dernierblocseq1);
    WRITELN(Fs, '  >> Dernier indice dans le dernier bloc :', Dernierseq1);
    WRITELN(Fs);
    WRITELN(Fs);
    WRITELN(Fs, ' + Contenu du fichier : ');

    FOR I:=2 TO Dernierblocseq1 DO
```

```
    BEGIN
      SEEK(F1, I-1);
      READ(F1, Bloc1);
      WRITELN(Fs);
      WRITELN(Fs, 'Bloc ',I, ' :');
      FOR J:=1 TO Bloc1.Nombre DO
        WRITE(Fs, Bloc1.Tab[J].Cle, '  ' );
      WRITELN(Fs);
    END;
  END;

 { Insertions répétées en fin de fichier }

 PROCEDURE Add (Article : Typearticle);
  BEGIN
    Nombre_elements := Nombre_elements + 1 ;
    Inc(Dernierseq1);
    IF Dernierseq1 <= B
    THEN
      BEGIN
        Bloc1.Tab[Dernierseq1]:= Article;
        Inc(Bloc1.Nombre)
      END
    ELSE
      BEGIN
        Inc(Dernierblocseq1);
        SEEK(F1, Dernierblocseq1 - 1);
        WRITE(F1, Bloc1);

        Bloc1.Tab[1] := Article;
        Bloc1.Nombre := 1;
        Dernierseq1 := 1;
      END;
 END;

PROCEDURE Suivant ( VAR S: Typearticle);
 BEGIN
   IF (Numbloc= Dernierblocseq) AND (Depl >= Dernierseq)
   THEN S.Cle := -1
   ELSE
     BEGIN
       Depl := Depl + 1;
       IF Depl <= B
       THEN
         S := Bloc.Tab(.Depl.)
       ELSE
         IF Numbloc = Dernierblocseq
         THEN S.Cle := - 1
         ELSE
           BEGIN
             Numbloc := Numbloc + 1;
             SEEK(F, Numbloc-1);
```

```
            READ(F, Bloc);
            Depl := 1;
            S := Bloc.Tab(.Depl.)
          END;
    END;
  END;

BEGIN
  ASSIGN(Fs, 'R_f3.Pas');
  REWRITE(Fs);
  ASSIGN(F, 'Fichier');

  WRITELN(Fs, 'Ouverture du fichier...');
  WRITELN(Fs);
  Ouvrir;

  WRITELN(Fs, 'Construction des fichiers...');
  Depl := 0;
  Numbloc := 2;
  SEEK(F, 1);
  READ(F, Bloc);
  Num:= 0;
  Suivant(S);
  WHILE S.Cle <> -1 DO
   BEGIN
    WRITELN(Fs);
    WRITELN(Fs, '* * * Chargement d''une partie du fichier dans un tableau...');
    WRITELN(Fs);
    K:=1;
    WHILE (K <= 20) AND (S.Cle <> -1)  DO
     BEGIN
      Tab(.K.) := S;
      Suivant(S);
      K:= K + 1
     END;
    K:= K-1;
    WRITELN(Fs, '* * * Tri en mémoire...');
    WRITELN(Fs);
    FOR I:= 1 TO K-1 DO
     FOR J:=K DOWNTO I+1 DO
      IF Tab(.J-1.).Cle > Tab(.J.).Cle
      THEN
       BEGIN
        Temp := Tab(.J-1.);
        Tab(.J-1.) := Tab(.J.);
        Tab(.J.) := Temp
       END;

    WRITELN(Fs, 'Résultat du tri :');
    WRITELN(Fs);
    FOR I:=1 TO K DO
     WRITELN(Fs, Tab(.I.).Cle );
```

```
      WRITELN(Fs);
      WRITELN(Fs, '* * * Recopier le tableau sur un fichier...');
      WRITELN(Fs);
      Num := Num + 1;
      ASSIGN(F1, 'Fichier'+CHR( 48+Num) );
      REWRITE(F1);
      Bloc1.Diff := Table;
      Bloc1.Nombre := 0;
      Dernierblocseq1 := 1;
      Dernierseq1 := 0;
      Nombre_elements := 0;
      Efface := 0;
      FOR I:=1 TO K DO
        Add( Tab(.I.) );
      { Ecriture Dernier Bloc }
      IF Dernierseq1 > 0
      THEN
        BEGIN
          Inc(Dernierblocseq1);
          SEEK(F1, Dernierblocseq1 - 1);
          WRITE(F1, Bloc1);
        END;
      Bloc1.Diff := Inf;
      Bloc1.Nombre_elements := Nombre_elements;
      Bloc1.Efface := Efface;
      Bloc1.Dernierblocseq := Dernierblocseq1;
      Bloc1.Dernierseq := Dernierseq1;
      RESET(F1);
      SEEK(F1, 0);
      WRITE(F1, Bloc1);
      WRITELN(Fs, 'Parcours séquentiel du fichier et listage de ses éléments');
      WRITELN(Fs);
      Impseq;
    END;
  CLOSE(Fs);
END.
```

**Contenu du fichier R_F3.PAS :**

```
Ouverture du fichier...

Construction des fichiers...

* * * Chargement d'une partie du fichier dans un tableau...

* * * Tri en mémoire...

Résultat du tri :

0
17
56
```

106
111
115
116
221
276
374
395
429
490
603
624
768
885
886
896
914

* * * Recopier le tableau sur un fichier...

Parcours séquentiel du fichier et listage de ses éléments

+ Caractéristiques du fichier :

>> Nombre d'articles : 20
>> Nombre d'articles effacés :0
>> Dernier bloc :3
>> Dernier indice dans le dernier bloc :10

+ Contenu du fichier :

Bloc 2 :
0 17 56 106 111 115 116 221 276 374

Bloc 3 :
395 429 490 603 624 768 885 886 896 914

* * * Chargement d'une partie du fichier dans un tableau...

* * * Tri en mémoire...

Résultat du tri :

53
96
111
164
191
216
278
286

300
323
330
541
573
593
643
657
722
776
828
993

* * * Recopier le tableau sur un fichier...

Parcours séquentiel du fichier et listage de ses éléments

+ Caractéristiques du fichier :

>> Nombre d'articles : 20
>> Nombre d'articles effacés :0
>> Dernier bloc :3
>> Dernier indice dans le dernier bloc :10

+ Contenu du fichier :

Bloc 2 :
53 96 111 164 191 216 278 286 300 323

Bloc 3 :
330 541 573 593 643 657 722 776 828 993

* * * Chargement d'une partie du fichier dans un tableau...

* * * Tri en mémoire...

Résultat du tri :

244
404
406
420
494
502
536
616
632
635
641
756
820

853
858

* * * Recopier le tableau sur un fichier...

Parcours séquentiel du fichier et listage de ses éléments

+ Caractéristiques du fichier :

>> Nombre d'articles : 15
>> Nombre d'articles effacés :0
>> Dernier bloc :3
>> Dernier indice dans le dernier bloc :5

+ Contenu du fichier :

Bloc 2 :
244 404 406 420 494 502 536 616 632 635

Bloc 3 :
641 756 820 853 858

**2. Deuxième programme : Fusion de n fichiers triés**

```
PROGRAM Fichier4;
 CONST
   B = 10;
   Infini = 10000;
 TYPE
   Typecle = INTEGER  ;
   Typeqq = STRING[20] ;
   Nature = (Inf, Table);
   Typearticle = RECORD
     Cle : Typecle;
     Info : Typeqq
   END;   Typebloc = RECORD
     CASE Diff : Nature OF
       1 : (   Nombre : INTEGER;
             Tab : ARRAY[1..B] OF Typearticle );
       2 : (   Nombre_elements, Efface, Dernierblocseq, Dernierseq: INTEGER )
     END;
   Typefichier = File OF Typebloc;
   Type_element = RECORD
      Numbloc : INTEGER;
      Depl : INTEGER;
      Dernierbloc : INTEGER;
      Dernier : INTEGER;
      Nombre_elements : INTEGER;
      Ptbloc : ^Typebloc;
    END;
```

```
VAR
  Fichier : ARRAY[1..10] OF Type_element;
  Tabmin : ARRAY[1..10] OF Typecle;
  Pluspetit : Typecle;
  Indice : INTEGER;
  Exist : BOOLEAN;
  Fs : TEXT;
  I : INTEGER;

  Efface, Nombre_elements : INTEGER;
  F, Fich, Fich1, Fich2, Fich3 : File OF Typebloc;
  Dernierseq,Dernierblocseq : INTEGER;
  Bloc : Typebloc;

  Depl, Numbloc : INTEGER;
  Temp, S : Typearticle;
  Tab : ARRAY[1..20] OF Typearticle;
  K : INTEGER;
  Num : INTEGER;

{ Création Du Fichier }

PROCEDURE Creation;
  BEGIN
    Bloc.Diff := Inf;
    Bloc.Nombre_elements := 0;
    Bloc.Efface := 0;
    Bloc.Dernierblocseq := 1;
    Bloc.Dernierseq := 0;
    REWRITE(F);
    SEEK(F, 0);
    WRITE(F, Bloc);
    CLOSE(F)
  END;

{ Ouverture du fichier }

PROCEDURE Ouvrir;
  BEGIN
    Bloc.Diff := Inf;
    RESET(F);
    SEEK(F, 0);
    READ(F, Bloc);
    Efface := Bloc.efface;
    Nombre_elements := Bloc.Nombre_elements;
    Dernierblocseq := Bloc.Dernierblocseq;
    Dernierseq := Bloc.Dernierseq;
  END;

{ Le minimum courant }
```

```
  PROCEDURE Min ( VAR Inf : Typecle; VAR Indice : INTEGER; VAR Exist :
BOOLEAN);
    VAR
      I, NInfini : INTEGER;
    BEGIN
      Inf := Tabmin[1];
      Indice := 1;
      IF Tabmin[1] = Infini
      THEN NInfini := 1
      ELSE NInfini := 0;
      FOR I:=2 TO Num DO
        IF Tabmin[I] = Infini
        THEN NInfini := NInfini + 1
        ELSE IF Inf > Tabmin[I]
            THEN BEGIN
                 Inf := Tabmin[I];
                 Indice := I
                 END;
      Exist := NOT ( Num = NInfini )
    END;

  { Parcours séquentiel du fichier et listage de ses éléments sur un fichier TEXT }

  PROCEDURE Impseq;
    VAR
      I, J : INTEGER;
    BEGIN
      RESET(F);
      { Les caractéristiques }
      WRITELN(Fs, ' + Caractéristiques du fichier : ');
      WRITELN(Fs);
      WRITELN(Fs, '  >> Nombre d''articles : ', Nombre_elements );
      WRITELN(Fs, '  >> Nombre d''articles effacés :', Efface);
      WRITELN(Fs, '  >> Dernier bloc :', Dernierblocseq);
      WRITELN(Fs, '  >> Dernier indice dans le dernier bloc :', Dernierseq);
      WRITELN(Fs);
      WRITELN(Fs);
      WRITELN(Fs, ' + Contenu du fichier : ');

      FOR I:=2 TO Dernierblocseq DO
        BEGIN
          SEEK(F, I-1);
          READ(F, Bloc);
          WRITELN(Fs);
          WRITELN(Fs, 'Bloc ',I, ' :');
          FOR J:=1 TO Bloc.Nombre DO
            WRITE(Fs, Bloc.Tab[J].Cle, '  ' );
          WRITELN(Fs);
        END;
      END;

    { Insertions répétées en fin de fichier }
```

```
 PROCEDURE Add (Article : Typearticle);
  BEGIN
    Nombre_elements := Nombre_elements + 1 ;
    Inc(Dernierseq);
    IF Dernierseq <= B
    THEN
     BEGIN
       Bloc.Tab[Dernierseq]:= Article;
       Inc(Bloc.Nombre)
     END
    ELSE
     BEGIN
       Inc(Dernierblocseq);
       SEEK(F, Dernierblocseq - 1);
       WRITE(F, Bloc);
       Bloc.Tab[1] := Article;
       Bloc.Nombre := 1;
       Dernierseq := 1;
     END;
 END;

{ Le suivant sur un fichier donné }

PROCEDURE Suivant ( I : INTEGER; VAR S: Typearticle);
 BEGIN
   IF (Fichier[I].Numbloc= Fichier[I].Dernierbloc) AND
     (Fichier[I].Depl >= Fichier[I].Dernier)
   THEN S.Cle := Infini
   ELSE
    BEGIN
      Fichier[I].Depl := Fichier[I].Depl + 1;
      IF Fichier[I].Depl <= B
      THEN
        S := Fichier[I].Ptbloc^.Tab(.Fichier[I].Depl.)
      ELSE
       IF Fichier[I].Numbloc = Fichier[I].Dernierbloc
       THEN S.Cle := Infini
       ELSE
        BEGIN
          Fichier[I].Numbloc := Fichier[I].Numbloc + 1;
          CASE I of
            1 : BEGIN
                  SEEK(Fich1, Fichier[I].Numbloc-1);
                  READ(Fich1, Fichier[I].Ptbloc^)
                END;
            2 : BEGIN
                  SEEK(Fich2, Fichier[I].Numbloc-1);
                  READ(Fich2, Fichier[I].Ptbloc^)
                END;
            3 : BEGIN
                  SEEK(Fich3, Fichier[I].Numbloc-1);
```

```
                READ(Fich3, Fichier[I].Ptbloc^)
              END;
          END;
          Fichier[I].Depl := 1;
          S := Fichier[I].Ptbloc^.Tab(.Fichier[I].Depl.)
        END;
    END;
  END;

BEGIN
  ASSIGN(Fs, 'R_f4.Pas');
  REWRITE(Fs);

  { Le fichier à créer }
  ASSIGN(F, 'Fichier');
  WRITELN(Fs, 'Création du fichier...');
  WRITELN(Fs);
  Creation;
  WRITELN(Fs, 'Ouverture du fichier...');
  WRITELN(Fs);
  Ouvrir;
  Bloc.Diff := Table;
  Bloc.Nombre := 0;

  Num := 3;

  { Récupération des caractéristiques et chargemment des premiers
    blocs de chaque fichier }

  FOR I:=1 TO Num DO
    BEGIN
      NEW(Fichier[I].Ptbloc);
      Fichier[I].Ptbloc^.Diff := Inf;
      ASSIGN(Fich, 'Fichier'+ CHR(48+I) );
      RESET(Fich);

      SEEK(Fich, 0);
      READ(Fich, Fichier[I].Ptbloc^);
      Fichier[I].Nombre_elements := Fichier[I].Ptbloc^.Nombre_elements;
      Fichier[I].Dernierbloc := Fichier[I].Ptbloc^.Dernierblocseq;
      Fichier[I].Dernier := Fichier[I].Ptbloc^.Dernierseq;

      Bloc.Diff := Table;
      Fichier[I].Depl := 0;
      Fichier[I].Numbloc := 2;
      SEEK(Fich, 1);
      READ(Fich, Fichier[I].Ptbloc^)
    END;

  ASSIGN(Fich1, 'Fichier1');
  ASSIGN(Fich2, 'Fichier2');
  ASSIGN(Fich3, 'Fichier3');
```

```
  RESET(Fich1);
  RESET(Fich2);
  RESET(Fich3);

  { Fusion des fichiers }

  FOR I:= 1 TO Num DO
    BEGIN
      Suivant(I, S);
      Tabmin[I] :=  S.Cle
    END;
  WRITELN(Fs, 'Trace de la fusion :');
  WRITELN(Fs);
  Min(Pluspetit, Indice, Exist);
  WHILE Exist DO
    BEGIN
      WRITELN(Fs, 'Fichier ', Indice, ' ----> ', Pluspetit);
      S.Cle := Pluspetit;
      Add( S);
      Suivant(Indice, S);
      Tabmin[Indice] := S.Cle;
      Min(Pluspetit, Indice, Exist)
    END;

  { Ecriture du dernier bloc }
  IF Dernierseq > 0
  THEN
    BEGIN
      Inc(Dernierblocseq);
      SEEK(F, Dernierblocseq - 1);
      WRITE(F, Bloc);
    END;
  Bloc.Diff := Inf;
  Bloc.Nombre_elements := Nombre_elements;
  Bloc.Efface := Efface;
  Bloc.Dernierblocseq := Dernierblocseq;
  Bloc.Dernierseq := Dernierseq;

  SEEK(F, 0);
  WRITE(F, Bloc);
  WRITELN(Fs);
  WRITELN(Fs, 'Parcours séquentiel du fichier et listage de ses éléments');
  WRITELN(Fs);
  Impseq;
  CLOSE(Fs);
END.

  CLOSE(Fs);
 END.
```

**Contenu du fichier R_F3.PAS :**

Création du fichier...

Ouverture du fichier...

Trace de la fusion :

Fichier 1 ----> 0
Fichier 1 ----> 17
Fichier 2 ----> 53
Fichier 1 ----> 56
Fichier 2 ----> 96
Fichier 1 ----> 106
Fichier 1 ----> 111
Fichier 2 ----> 111
Fichier 1 ----> 115
Fichier 1 ----> 116
Fichier 2 ----> 164
Fichier 2 ----> 191
Fichier 2 ----> 216
Fichier 1 ----> 221
Fichier 3 ----> 244
Fichier 1 ----> 276
Fichier 2 ----> 278
Fichier 2 ----> 286
Fichier 2 ----> 300
Fichier 2 ----> 323
Fichier 2 ----> 330
Fichier 1 ----> 374
Fichier 1 ----> 395
Fichier 3 ----> 404
Fichier 3 ----> 406
Fichier 3 ----> 420
Fichier 1 ----> 429
Fichier 1 ----> 490
Fichier 3 ----> 494
Fichier 3 ----> 502
Fichier 3 ----> 536
Fichier 2 ----> 541
Fichier 2 ----> 573
Fichier 2 ----> 593
Fichier 1 ----> 603
Fichier 3 ----> 616
Fichier 1 ----> 624
Fichier 3 ----> 632
Fichier 3 ----> 635
Fichier 3 ----> 641
Fichier 2 ----> 643
Fichier 2 ----> 657
Fichier 2 ----> 722
Fichier 3 ----> 756
Fichier 1 ----> 768
Fichier 2 ----> 776

Fichier 3 ----> 820
Fichier 2 ----> 828
Fichier 3 ----> 853
Fichier 3 ----> 858
Fichier 1 ----> 885
Fichier 1 ----> 886
Fichier 1 ----> 896
Fichier 1 ----> 914
Fichier 2 ----> 993

Parcours séquentiel du fichier et listage de ses éléments

+ Caractéristiques du fichier :

>> Nombre d'articles : 55
>> Nombre d'articles effacés :0
>> Dernier bloc :7
>> Dernier indice dans le dernier bloc :5

+ Contenu du fichier :

Bloc 2 :
0 17 53 56 96 106 111 111 115 116

Bloc 3 :
164 191 216 221 244 276 278 286 300 323

Bloc 4 :
330 374 395 404 406 420 429 490 494 502

Bloc 5 :
536 541 573 593 603 616 624 632 635 641

Bloc 6 :
643 657 722 756 768 776 820 828 853 858

Bloc 7 :
885 886 896 914 993

# Annexes

# Langage algorithmique

*Variables et constantes*

Un algorithme opère sur des objets. A tout objet est associé un **nom** qui permet de l'identifier de façon unique. C'est généralement une suite de caractères alphanumériques dont le premier est alphabétique.

On distingue deux types d'objets :
- des objets qui peuvent varier durant le déroulement d'un algorithme **Variables** (ardoises).
- des objets qui ne peuvent pas variés par le déroulement d'un algorithme **Constantes**.

On peut répartir l'ensemble des objets en sous ensembles appelés **classes** ou types. Il existe 4 types standards :
- ENTIER : l'ensemble des entiers relatifs
- REEL : l'ensemble des réels
- BOOLEEN : les valeurs VRAI et FAUX
- CAR : l'ensemble des chaînes de caractères

En langage algorithmique, on définit les objets comme suit :
*CONST Type Nom = valeur*
*VAR Nom1, Nom2, .... : Type*

## Objets composés

### Définition de type

Au niveau du langage algorithmique, un type est construit comme suit :

*TYPE Nomdutype = STRUCTURE*
*Idf1 : Type1*
*Idf2 : Type2*
*....*
*FIN*

Typei peut être un type standard ou construit.

### Déclaration des variables d'un type donné

On définit les objets d'un type construit comme suit:

*Var Nom : Type*

Nom est le nom de l'objet composé et Type un type quelconque préalablement construit.

On peut aussi si nécessaire définir une relation d'ordre entre les éléments appartenant à une même classe d'objets.

### Accès à un champ d'un objet composé

Si X est une variable d'un type donné, X.Nom permet d'accéder au champ Nom de la variable composée X.

## Expressions sur les objets

Une expression est une suite d'opérandes reliés par des opérateurs.

Une expression peut être :
- arithmétique. On utilisera les opérateurs suivants +,-,/, *.
Exemple A + B/C, A/B/C

- logique. Les opérateurs les plus utilisés sont : ET, OU et NON.
Exemple X et Y, NON X OU Z

- relationnelle. Les opérateurs utilisés sont : <, <=, >, >=, =, <>.
Exemple : A < B, A+5 = B/C

## Vecteurs

### Description algorithmique d'un vecteur

On décrit un vecteur en langage algorithmique comme suit :

*VAR Nom : TABLEAU[1..N] DE Type*

Nom est le nom du tableau, et Type est un type quelconque.

### Description algorithmique d'un tableau à d dimensions

On décrit un tableau à plusieurs dimensions en langage algorithmique comme suit :

*VAR Nom : TABLEAU[1..N1, 1..N2, ....., 1..Nd] DE Type*

Nom est le nom du tableau, Type est un type quelconque et d est la dimension du tableau.

### Accès à un élément d'un tableau

Il se fait par l'opération d'indexation. Si T est le nom d'un tableau à une dimension, T[I] permet l'accès à son I-ième élément. Si T est le nom d'un tableau à deux dimensions, T[I,J] permet l'accès à l'élément de ligne I et de colonne J.

## Listes linéaires chaînées

### Définition d'un maillon d'une liste linéaire chaînée

Dans le langage algorithmique, on définira le type d'un maillon comme suit :

```
        TYPE Typedumaillon = STRUCTURE
 Valeur  : Typeqq    { désigne un type quelconque }
 Adresse  : POINTEUR(Typedumaillon)
FIN
```

POINTEUR est un mot-clé du langage algorithmique servant à définir la classe des adresses.

### Définition d'une liste linéaire chaînée

Une liste linéaire chaînée est définie par l'adresse de son premier élément.

*VAR Liste :  POINTEUR(Typedumaillon)*

### Modèle sur les listes linéaires chaînées

Afin de développer des algorithmes sur les listes linéaires chaînées, on construit une machine abstraite avec les opérations suivantes :

Allouer, Libérer, Aff_Adr, Aff_Val, Suivant, Valeur

définies comme suit :

Allouer(T, P) : allocation d'un espace de taille spécifiée par le type T. L'adresse de
   cet espace est rendue dans la variable POINTEUR P.
Libérer(P)  : libération de l'espace pointé par P.
Valeur(P)  : consultation du champ Valeur du maillon d'adresse P.
Suivant(P)  : consultation du champ Adresse du maillon d'adresse P.
Aff_Adr(P, Q): dans le champ Adresse du maillon d'adresse P, on range l'adresse Q.
Aff_Val(P,Val): dans le champ Valeur du maillon d'adresse P, on range la valeur Val.

## Fichier

### Description algorithmique d'un fichier

On définit un fichier comme suit :

*VAR Nom : FICHIER DE Type*

Nom est le nom du fichier et Type le type d'un article ou bloc du fichier.

Le fichier peut être vu comme
- un  ensemble d'articles
- un ensemble de blocs

Dans le cas où le fichier est vu comme un ensemble de blocs, on peut considérer le cas où les articles sont de longueur fixe et le cas où ils sont de longueur variable.

Dans le premier cas, le bloc est une structure contenant au moins un tableau d'objets. Dans le second cas, le bloc est considéré comme un tableau de caractères. La récupération des différents champs des articles est à la charge du programmeur.

**Opérations**

Ouvrir(Nom) : ouverture du fichier
Fermer(Nom) : ferméture du fichier

Positionner(Nom, I) : positionnement au I-ième article ( bloc ) du fichier Nom.

Lirefichier(Nom, Zone) : lecture dans le buffer Zone de l'article ( ou le bloc) courant.
Ecrirefichier(Nom, Zone) : écriture du contenu du buffer Zone dans le fichier à la position courante.

Les opérations de lecture et d'écriture permettent d'avancer la position courante dans le fichier d'une unité. A l'ouverture du fichier, la position courante est à 1.

**Article ou bloc d'en-tête**

Au sein d'un même fichier, on peut avoir des articles ( ou blocs ) de types différents : un type pour les articles ( bloc ) et un type pour l'article (bloc) d'en-tête contenant les caractéristiques, c'est à dire toutes les informations utiles pour l'exploitation du fichier.

```
    TYPE Nomdutype = STRUCTURE
 CAS Select DE
 1 : ( définitions des champs des articles ou des blocs )
 2 : ( définition des champs de l'article ou du bloc d'en-tête )
FIN
```

Afin de modifier le type courant, on positionne le selecteur à 1 ou 2 selon que l'on fait une lecture (écriture) d'un article du fichier ou d'un article d'en-tête.

Si Zone est une variable de type Nomdutype, la selection se fait l'opération

*Select(Valeur)*

avec Valeur dans l'ensemble { 1, 2 }

La selection courante reste valide jusqu'à la nouvelle opération de sélection.

## Structures de contrôle

Tous les algorithmes peuvent être exprimés par les "tournures" suivantes qu'on appelle structures de contrôle du fait qu'elles permettent de contrôler le déroulement des algorithmes :

**Le séquencement**

Exprime qu'un ensemble d'actions est exécuté dans un ordre bien précis.

**La répétitive " TANTQUE"**

Elle exprime qu'un ensemble d'actions se déroule plusieurs fois tant qu'une condition reste vérifiée.

*TANTQUE condition :*
*Action 1*
*Action 2*
*....*
*Action n*
*FINTANTQUE*

La condition constitue le critère d'arrêt.

**La répétitive " POUR"**

On utilise cette structure quand on connaît à l'avance le nombre d'itérations d'une boucle. La formulation algorithmique est la suivante :

*POUR Variable := Expression1, Expression2, Expression3*
*Action 1*
*Action 2*

*.....*

*Action n*
*FINPOUR*

Variable désigne une variable entière.
Expression1, Expression2 et Expression3 désignent des expressions entières.
Expression1 est la valeur initiale de Variable, Expression2 la valeur finale et expression3 l'incrément.

Toute boucle "POUR" peut se traduire en boucle "TANTQUE". Par contre, l'inverse n'est pas toujours possible.

**La conditionnelle**

Elle exprime qu'un ensemble d'actions est exécuté que si une condition est vérifiée.

*SI Condition :*
*Action 1*
*Action 2*
*....*
*Action n*
*FSI*

**L'alternative**

Elle exprime un choix entre deux ensembles d'actions à exécuter selon la valeur d'une condition.

*SI condition :*
*Action 1*
*Action 2*
*....*
*Action n*
*SINON*
*Action n+1*
*Action n+2*
*....*
*Action n + p*
*FSI*

**Remarque :** Chaque action peut être à son tour une répétitive, une conditionnelle, une alternative.

## Autres actions du langage algorithmique

### Affectation

C'est l'opération de base. Elle permet d'attribuer la valeur d'une expression calculable à une variable. On utilisera le symbole :=.

### Lecture

Elle permet d'introduire des données dans des variables. Nous utiliserons l'opération

*LIRE( X, Y, Z, .....)*

Ce qui est équivalent à

X := première donnée
Y := deuxième donnée
Z := troisième donnée

### Ecriture

Elle permet de restituer des résultats. Nous utiliserons l'opération

*ECRIRE(Expression1, Expression2,.....)*

## Modules

### Actions composées

Quand une séquence d'actions se répète plusieurs fois dans un algorithme, il est avantageux et même nécessaire de créer une **action composée** (ou **module**). En langage algorithmique, on définit un module de la façon suivante :

*ACTION Nom ( V1, V2, )*

*Définition des paramètres d'entrée et de sortie*
*Définition des objets locaux*
*DEBUT*
*Corps*
*FIN*

Au niveau de la définition de l'action , les paramètres sont dits **formels**.
L'appel se fait par Nom (P1, P2,....). Les paramètres sont dits **réels** ou **effectifs**.

**Fonctions**

Quand le résultat de l'action composée est unique et de type arithmétique (ou caractère) , il est préférable d'écrire l'action composée sous la forme d'une *fonction*.

Une fonction est définie comme suit :

*FONCTION Nom ( Paramètres d'entrée ) : Type*
*Définition des paramètres et des objets locaux*
*DEBUT*
*Corps*
*FIN*

Une fonction est utilisée directement dans une expression.

Dans le corps de la fonction il doit toujours exister une affectation du genre
" Nom de la fonction := valeur ".
C'est cette valeur qui est retournée par la fonction.

**Prédicat**

Quand le résultat de l'action composé est unique et de type booléen, il est préférable d'écrire l'action composée sous la forme d'un *prédicat*.

*PREDICAT Nom ( Paramètres )*
*Définition des paramètres et des objets locaux*
*DEBUT*
*Corps*
*FIN*

Dans le corps du prédicat il doit toujours exister une affectation du genre
" Nom du prédicat := valeur ".

**Structure d'un algorithme**

Un algorithme est composé de deux parties :
- définition des données et des modules : en-tête,
- corps de l'algorithme.

L'en-tête précise
- le nom de l'algorithme,
- la définition des objets manipulés et des modules utilisés.

Le corps est un ensemble d'actions ordonnées composé généralement de 3 parties :
- initialisations ( lectures, .. ),
- itérations ( parcours d'un ensemble de données),
- opérations finales ( écritures, .. ).

La description algorithmique est la suivante :

*ALGORITHME Nom*
*Définition des objets*
*Définition des modules*
*DEBUT*
*Corps*
*FIN*

# Notions élémentaires

*Configuration physique d'un ordinateur*

Un ordinateur est une machine électronique comprenant essentiellement:
- une **mémoire centrale**, où est rangée temporairement l'information manipulée par l'ordinateur.
- une **unité de traitement** de l'information, permettant de réaliser un certain nombre d'opérations élémentaires.

Encore faut-il que l'homme puisse "communiquer" avec cette machine, c'est à dire, lui fournir des données et des ordres, et en recevoir des résultats. C'est ce que réalisent les **organes périphériques**, ou d'entrées/sorties. Parmi ceux-ci, on utilise le plus couramment les **moniteurs** pour communiquer des informations ou des ordres à l'ordinateur et aussi pour restituer les résultats. On utilise également l'**imprimante** pour la restitution des résultats.

### Forme binaire de l'information

A l'intérieur d'un ordinateur, toute information ( au sens large : donnée et ordre) est représentée à l'aide d'éléments physiques pouvant se trouver dans un état parmi 2 (correspondant à la présence ou l'absence d'une impulsion électrique) que l'on identifie par convention aux valeurs 0 et 1 . Ainsi toute information se représente par une suite de 0 et 1. Un tel chiffre binaire s'appelle une **monade**.

Par exemple le caractère 'X' sera représenté par 01011000 en ASCII, c'est à dire un groupe de 8 chiffres binaires, que l'on appelle **octet**. Un **mot** est en général un multiple de 2 d'octets.

### Programme, données externes, résultats

Un programme est une suite d'ordres intimés à l'ordinateur, codifiés dans une langue qui lui soit accessible.

Supposons que vous désirez faire résoudre une équation du second degré par l'ordinateur : vous allez donc étudier l'algorithme mathématique permettant de résoudre le problème et le transcrire dans un langage de programmation (FORTRAN, PLI, PASCAL, ..), mais vous n'allez pas écrire le programme pour résoudre une équation particulière : il vous faudrait alors réécrire le programme pour chaque équation et l'avantage de l'ordinateur deviendrait nul.

Vous allez donc programmer la résolution de l'équation générale : ax2 + bx + c = 0 et vous demanderez à l'ordinateur de prendre en compte les valeurs des coefficients a, b , c qui vous intéresse pour l'instant, ce qui se fera par un ordre de **lecture**, réalisé par un organe d'entrée : l'écran en général.

Les valeurs concrètes a, b, c sont des **données externes** pour votre programme, qui pourront varier d'une exécution à une autre de votre programme, qui lui demeurera inchangé.

De même, l'équation résolue, vous aimerez bien que l'ordinateur ne garde pas pour lui la solution, mais vous la communique par un ordre d'**écriture**, dont la réalisation concrète s'effectuera par un organe de sortie: l'écran en général.

### Les compilateurs

Un ordinateur, en réalité, ne comprend véritablement qu'un seul langage : le langage machine, où chaque ordre est une suite de monades précisant des opérations très élémentaires.

La manipulation d'un tel langage serait bien lourde, bien fastidieuse, et généralement engendre beaucoup d'erreurs. C'est pourquoi dans la pratique, on utilise des langages symboliques dits évolués, indépendants de l'ordinateur particulier sur lequel on travaille, comme le PASCAL ou FORTRAN par exemple.

Si l'ordinateur peut comprendre de tels langages, c'est qu'il existe, associé à chacun d'entre eux, un programme tout fait, capable de traduire un texte ( PASCAL ou FORTRAN ) en langage de la machine : ces programmes sont appelés**compilateurs**

Le rôle du compilateur est double :

- d'une part il analyse votre programme pour voir s'il respecte les règles du langage (syntaxe) et vous signale toute erreur qu'il décèle, tentant parfois de la corriger.

- d'autre part, il le traduit en langage machine, permettant ainsi à l'ordinateur d'exécuter vos ordres.

Attention, le compilateur ne porte aucun jugement sur la valeur logique de votre programme : c'est un simple interprète entre deux langages.

### Le superviseur (système d'exploitation)

Pour réaliser une application donnée, travail que l'on soumet à un ordinateur, on a besoin d'utiliser un certain nombre de**logiciels**, programmes tout faits mis à la disposition de l'utilisateur tels que les compilateurs, les éditeurs de texte,...

Un dialogue doit donc être établi entre la machine et l'utilisateur. Toute cette série d'informations est communiquée à l'ordinateur à l'aide d'un langage spécial: le langage de commande qu'il est obligatoire d'utiliser pour tout travail à soumettre à l'ordinateur.

Ces commandes sont analysées par un programme spécial, le**superviseur**.

# Les disques

*Description physique des disques*

L'information sur disque est rangée sur une surface composée d'une ou plusieurs **plateaux**. L'information est généralement rangé dans des **pistes** successives sur la surface du disque. Chaque piste est souvent divisée en un nombre de **secteurs.** Un secteur est la plus petite portion adressable du disque.

Quand une lecture est lancée pour un octet particulier se trouvant sur un fichier sur disque, le système de gestion de fichiers trouve la surface concernée, la piste puis le secteur. Ensuite il lit tout le secteur dans une zone spéciale dans la RAM appelée **buffer**, puis détermine exactement l'octet désiré.

Si l'unité de disque utilise plusieurs plateaux, on l'appelle **disk pack.** Les pistes qui sont directement dessous et dessus forme un **cylindre** (colonne). Toutes les informations sur un même cylindre peuvent être accédées sans bouger le bras qui tient les têtes de lecture/écriture. Le mouvement de ce bras est appelé recherche ("SEEKING" ) et constitue la partie la plus lente dans une opération d'entrée/sortie.

### Estimation de la capacité des disques

Il existe des disques de 3 à 14 pouces. La capacité de stockage dépend de la densité. Une densité faible, pas chère, consiste à ranger 4 K-octets par piste et 35 pistes par surface. Une densité forte, donc chère peut avoir 50K-octets par surface et 1000 pistes par surface.

Capacité d'une piste = nombre de secteurs/piste X nombres d'octets/secteur.
Capacité du cylindre = nombre de piste par cylindre X capacité d'une piste.
Capacité du disque = nombre de cylindre X capacité du cylindre.

Si on connaît le nombre d'octets du fichier, on peut utiliser ces relations pour calculer l'espace disque nécessaire.

Exemple: soit un fichier de 20 000 articles de longueur fixe à ranger sur un disque avec les caractéristiques suivantes :
Nombre d'octets/secteur : 512
Nombre de secteurs/piste : 40
Nombre de piste par cylindre : 11

Combien nous faut-il de cylindre si chaque article possède 256 octets. ==> réponse (20000/2) / (40 X 11) = 24,7 cylindres. Les cylindres ne seront pas forcément contigus.

### Organisation des données

Il existe 2 façons d'organiser les données sur disque : par secteur et par bloc défini par l'utilisateur.

**Par secteur**

**Placement physique des secteurs**

Si les secteurs d'un fichier sont adjacents sur une piste, la lecture séquentielle est inefficace du fait que l'on perd le début du secteur suivant pendant le transfert de l'information.

Une bonne solution consiste à laisser un intervalle entre deux secteurs logiques.

Comme exemple, pour lire les 32 secteurs d'une piste il faut 32 révolutions dans le cas où les secteurs sont adjacents. Par contre si on prend un intervalle égal à 5 (principe des "secteurs interleaving"), alors 5 révolutions suffissent pour lire tous les secteurs.

**clusters**

C'est un nombre fixe de secteurs contigus. Tous les secteurs d'un cluster sont lus sans bouger la tête de lecture/écriture. Le système de gestion de fichier utilise une table pour gérer ces clusters. Chaque entrée dans cette table est le couple ( n° du cluster, adresse du cluster).

Pour un système donné, le nombre de secteurs par cluster est généralement fixé. L'idéal, bien sûr, c'est d'avoir des grands clusters. Ceci permettrait alors de lire des fichiers sans trop bouger le bras, ce qui donne de très bonne performance quand le fichier est traité séquentiellement.

**Etendue**

Quand il y a assez d'espace sur le disque, il est préférable de créer des fichiers entièrement avec des clusters contigus : c'est ce nous appelons une étendue. S'il n'y a pas assez d'espace, le fichier est alors distribué en plusieurs étendues. Il est certain que quand le nombre d'étendues augmente, les performances se dégradent.

**fragmentation**

Si la taille d'un secteur n'est pas un multiple de la taille d'un article, deux éventualités peuvent se produire :
- l'article n'est rangé dans le secteur que si la place est disponible.
- l'article peut être à cheval sur deux secteurs.

Dans le premier cas, il y a création de la fragmentation interne ( apparition de "trous"). Dans le second cas, deux accès sont exigés pour récupérer certains articles.

La fragmentation interne peut aussi résulter de l'utilisation des clusters : si le nombre d'octets dans un fichier n'est pas un multiple de la taille du cluster, il peut exister une fragmentation interne dans la dernière étendue du fichier.

**Par bloc**

Utilisé dans ce contexte, le terme bloc référence un groupe d'articles qui sont rangés ensemble sur le disque et traités comme une unité d'entrée/sortie. Quand les blocs sont utilisés, l'utilisateur est mieux placé pour faire l'organisation physique des données correspondant à son organisation logique, et donc quelquefois améliore les performances. Les unités organisées en blocs peuvent aussi permettre au contrôleur de rechercher parmi les blocs sur une piste un article avec une clé donnée sans transmettre le bloc non trouvé en RAM.

### Facteurs affectant le temps d'un accès disque

Le coût d'un accès disque peut être mesuré en terme de temps. C'est le temps occupé par le mouvement du bras, le*délai rotationnel* et le *transfert*. Si on utilise le principe des "secteurs interleaving", il est possible d’accéder à des secteurs logiquement adjacents en les séparant physiquement par un ou plusieurs secteurs. Bien qu’ils prennent plus de temps pour l'accès à un simple article directement que séquentiellement, le temps du positionnement du bras exigé pour l'accès direct le rend beaucoup plus lent que l'accès séquentiel quand une série d'articles est à accéder.

# Système d'exploitation MS-DOS
## Aide-mémoire

*Mise en route*

La mise en route d'un micro ordinateur s'effectue comme suit :
- Mise sous tension
- Introduire une disquette formatée (voir plus loin l'ordre Format) et contenant les fichiers système suivants :
  . Les fichiers cachés *IO.SYS* et *MSDOS.SYS*
  . *COMMAND.COM*
  . *KEYBFR.COM*
- L'apparition du prompt A>: le système attend une commande MS-DOS

*Les fichiers-disquettes*

Physiquement, un fichier correspond à des inscriptions magnétiques sur des secteurs de la disquette.
Une disquette peut contenir des utilitaires système, des programmes ou des données.
Les fichiers peuvent être sous forme texte (code ASCII) ou binaire.
Le nom d'un fichier est formé comme suit :
[lecteur] :Nom. [Extension]
Les crochets désignent des parties facultatives.
Le lecteur peut être a: ou b:
8 caractères au maximum pour former le nom
3 caractères au maximum pour former l'extension.

**Les fichiers .Bat**

Ils comprennent une liste de commandes par lot à exécuter en séquence.

La création ou modification de ses fichiers se fait par la commande
*Copy Con: <nom du fichier>*
Chaque commande se termine par la touche 'Entrée'.
Les commandes sont terminées par **^z** ( Ctrl Z).
Le fichier *AUTOEXEC.BAT* s'il existe est exécuté à la mise en route.

**Les caractères ASCII**

Les 256 caractères peuvent être obtenus par *ALT* + code ASCII

**Les commandes principales**

Commandes internes :

*DIR <nom de fichier> [/P] [/w]  Affichage du répertoire*

*DEL <nom de fichier>  Destruction d'un ou plusieurs fichiers*

*REN <ancien nom> <nouveau nom>  Change le nom d'un fichier*

*COPY <nom fichier source> <nom copie>  Crée une copie de fichier*

*TYPE <nom de fichier>  Affiche le contenu d'un fichier ASCII*

*CLS  Efface l'écran*

Dans les commandes peuvent figurer des nom ambigus :
? : remplace un caractère quelconque,
* remplace un groupe de caractères.

Commandes Externes :

Elles correspondent à des fichiers *.COM ou *.EXE de la disquette.

*FORMAT [a:/ b:] [/s] [/v]* . Formate les disquettes verges et recopie le système si l'option /s est présente.

*DISKCOPY <unité source> <unité cible>* Copie physique de toute la disquette

*MORE*  Affichage d'un fichier page par page

*SORT*  Tri alphabétique d'un fichier

### Enchaînement de plusieurs commandes

Il se fait par la barre verticale | (Alt 124 )

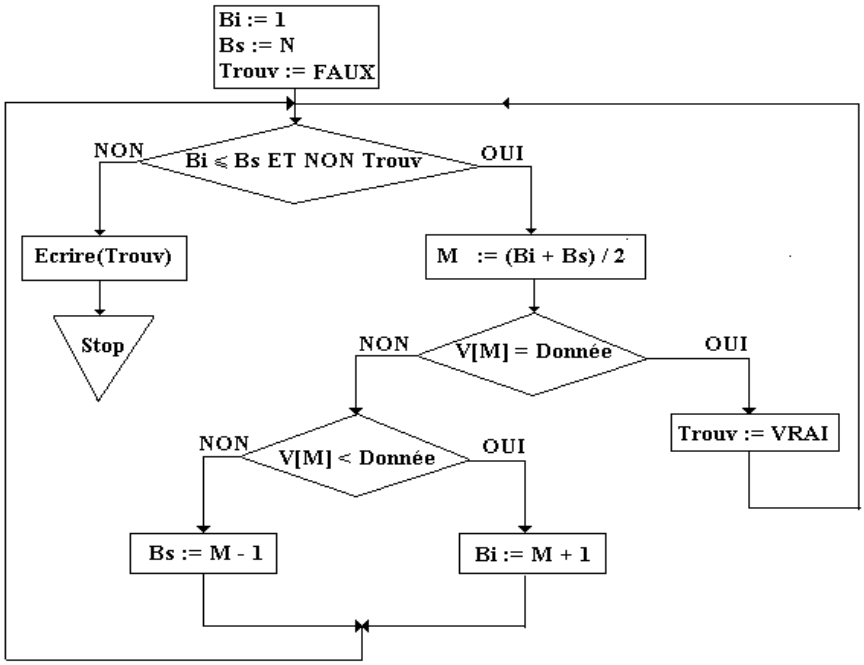
### Redirection des entrées/sorties

Elle se fait par < et > et >> sur des fichiers ou l'imprimante (*PRN*)

# Organigramme

C'est la représentation schématique d'un algorithme. Généralement, on utilise deux formes géométriques : losange pour représenter un test et rectangle pour représenter des actions.

Comme exemple, l'algorithme de la recherche dichotomique présenté en cours peut être schématisé sous la forme d'un organigramme comme suit :



# Traduction des langages de haut niveau vers des langages de bas niveau

Il est utile de connaître comment passer des algorithmes structurés (langage algorithmique utilisé dans cet ouvrage) aux algorithmes exprimés uniquement avec des branchements conditionnelles et inconditionnelles. C'est le cas par exemple du passage vers les langages d'assemblage.

**Répétitive**

Toute répétitive

*TANTQUE Condition :*
*Action 1*
*Action 2*
*...*
*Action n*
*FINTANTQUE*

se traduit par la séquence :

*Etiq : SI NON Condition ALLERA Suite*
*Action 1*
*Action 2*
*....*
*Action n*
*ALLERA Etiq*
*Suite: .....*

**Conditionnelle**

Toute conditionnelle

*SI Condition*
*Action 1*
*Action 2*
*.....*
*Action n*
*FSI*

se traduit par :

*SI NOT Condition ALLERA Suite*
*Action 1*
*Action 2*

*...*
*Action n*
*Suite : .....*

**Alternative**

Toute alternative

*SI Condition*
*Action a1*
*Action a2*
*.....*
*Action an*
*SINON*
*Action b1*
*Action b2*
*.....*
*Action bm*
*FSI*

se traduit par :

*SI NOT Condition ALLERA Suite1*
*Action a1*
*Action a2*
*...*
*Action aN*
*ALLERA Suite2*
*Suite1 : Action b1*
*Action b2*
*.....*
*Action bm*
*Suite2 : ...*

# Rapport de programmation

Afin de présenter un rapport de programmation, il est souhaitable de suivre le plan suivant tout en ayant à l'esprit qu'il doit être indépendant du langage de programmation d'une part, et d'autre part il doit être bien présenté et rédigé sans erreur de français.

*Problématique*

Il s'agit de reprendre l'énoncé en présentant clairement la problématique. De plus, à ce niveau, il faut définir toutes les données et les objectifs désirés ( résultats).

**Analyse**

Elle consiste à faire une analyse du problème en présentant tous les modules avec leurs entrées, sorties et rôles. Donner l'arborescence qui est généralement le résultat d'une analyse descendante. Pour les algorithmes les plus importants, donner des pseudo-algorithmes.

Rappeler si nécessaire les outils théoriques utilisés ( par exemple les systèmes de numération, principe de conversion, principe de détermination de la racine cubique, etc.)

**Algorithmes**

- Si l'algorithme est modulaire, faire bien ressortir les actions composées. Pour chacune d'elles bien préciser les entrées, les sorties et le rôle.

- On peut ne faire que des pseudo-algorithmes

- On résout les problèmes techniques; toutes les écritures de présentation ou de dialogue sont inutiles et n'ont pas de sens au niveau algorithmique.

- Si on donne un lexique, ne faire figurer que les variables importantes

**Programmation**

- Donner une grande importance au listing.
- Sauter des lignes (ou des pages) pour séparer les procédures.
- Faire des titres, des commentaires.
- On peut être amené à rajouter des procédures pour la présentation des sorties (résultats). On ne doit pas faire ressortir ces procédures dans le rapport. Par contre, dans le rapport, on peut donner des modèles de sortie.
- Intenter le programme de sorte à bien faire ressortir les structures de contrôle.
- Donner toutes les informations jugées utiles pour une bonne compréhension de la programmation

# Proverbes de programmation

***- Définissez les problèmes complètement.***

***- Réfléchissez d'abord, vous programmerez plus tard.***

***- Choisissez bien vos identificateurs.***

***- Evitez les astuces.***

***- Employez les commentaires.***

***- Soignez la présentation.***

***- Fournissez une bonne documentation.***

***- Testez le programme à la main avant de l'exécuter.***

***- Ne vous occuper pas d'une belle présentation des résultats avant que le programme soit correct.***

***- Quand le programme tourne soignez la présentation des résultats.***

***- N'ayez pas peur de tout recommencer.***

***- Divisez pour régner.***

***- Ne supposez jamais que l'ordinateur suppose quelque chose.***

Comprendre progressivement l'art de la programmation, maîtriser les algorithmes de base. Tels sont les objectifs recherchés à travers cet ouvrage.

---

**_ Ce livre - en deux tomes -décrit d'une manière très succincte et originale les concepts de base de l'algorithmique et d'une manière générale de la programmation.**
**_ De nombreux algorithmes sont développés sur la machine de Turing permettant de s'expérimenter sur le formalisme algorithmique.**
**_ Une méthode de conception d'algorithmes qu'est l'analyse descendante est exposée en mettant en évidence ses caractéristiques.**
**_ On y trouvera également des notions de quelques structures de données élémentaires telles que les objets composés, les tableaux et les listes linéaires chaînées.**
**_ Une introduction aux fichiers et aux structures de fichiers est également exposée et étoffée de nombreux programmes.**
**_ Un éventail de sujets d'examens avec des corrigés-type portant sur tous les cours est proposé. Ainsi, plus d'une centaine d'algorithmes sont proposés et solutionnés dans un langage algorithmique clair et concis.**
**_ Enfin, une série d'exercices programmés en PASCAL est aussi fournie.**

---

Ce livre s'adresse à des étudiants désirant apprendre l'art de la programmation. Il est aussi destiné aux enseignants, principalement comme un guide.