Cours NF01

P. Trigano
D. Lenne

Juin 2008

Table des matières

Table des matières 3

I - Chapitre 1 - Introduction 11

A. Ordinateur et programmation....11

B. Architecture d'un ordinateur....11

1. Matériel....11
2. Logiciel....12
3. Définitions et unités de mesure....12

C. Langages....12

D. Représentation de l'information....13

E. Rappels sur la numération....13

1. Généralités....13
2. Système décimal....14
3. Le système binaire....14
4. Autres systèmes....14

F. Conversions....15

1. Correspondances entre bases....15
2. Conversion base b – système décimal....15
3. Conversion système décimal – base b....15
4. Conversion Octal <--> Binaire....16
5. Conversion Hexadécimal <--> Binaire....17
6. Conversion Octal <--> Hexadécimal....17

G. Tables ASCII....17

II - Chapitre 2 - Algorithmes et langages 21

A. Algorithmes....21

1. Définition....21
2. Exemples....21
3. Règles de base....22

B. Notion de variable....22

1. Notions....22
2. Affectation....23
3. Structure de sélection simple....23
4. Structures répétitives....24

C. Représentations d'un algorithme....24

1. Representation....24
2. Exemples....25

D. Grammaires et langages....27

1. Introduction....27

2. Diagrammes de Conway....28
3. Formalisme BNF (Backus-Naur Form)....28
4. Programmation....29

III - Chapitre 3 - Premiers éléments de Pascal 31

A. Structure globale d'un programme Pascal....31
1. En-tête....32
2. Déclarations....32
3. Instructions....32
4. Structure de bloc....33

B. Exemple de programme structuré....33
1. Code source....33
2. Remarques....34

C. Grammaire d'un programme Pascal....34
1. L'Alphabet....34
2. Les mots du langage : définition....35
3. Les mots du langage : les identificateurs....35
4. Les mots du langage : les identificateurs standards....35

D. Déclarations....36
1. Les constantes....36
2. Les types : définitions....37
3. Type standard : Integer....37
4. Type standard : Real....38
5. Type standard : Boolean....39
6. Type standard : Char....39
7. Type standard : String....39
8. Type scalaire : Le type énuméré....39
9. Type scalaire : Le type intervalle....40
10. Les variables....41
11. Exemples de déclarations de constantes, types et variables....42

E. Entrées / Sorties....42
1. Lecture....42
2. Ecriture....43
3. Exemple complet de lectures/ecritures....44

F. Instruction d'affectation....45
1. Syntaxe et exemples....45

G. Opérateurs et fonctions arithmétiques....45
1. Opérateurs disponibles....45
2. Expressions....46
3. Fonctions arithmétiques....46
4. Fonctions logiques....47

H. Programmation structurée....47
1. Les exigences....47
2. Bénéfices attendus....48

IV - Chapitre 4 - Instructions alternatives 49

A. Choix simple....49
1. Définition....49
2. Equation du premier degré....49
3. Maximum de deux nombres....50
4. Exemple avec expressions relationnelles et booléennes....50

B. Choix multiple....50
1. Introduction....50
2. Définition....51
3. Simuler une calculatrice....53
4. Exemple : le loto....53

C. Instructions composées....54
1. Définition....54

V - Chapitre 5 - Instructions itératives 55

A. Boucles à bornes définies....55
1. Définition....55
2. Exemple d'itérations à bornes définies....55
3. La boucle à bornes définies en Pascal : for....56

B. Boucles à bornes non définies....56
1. Boucles à bornes non définies : Boucle TANT QUE....56
2. Boucles à bornes non définies en Pascal : while ... do....57
3. Boucles à bornes non définies : Boucle REPETER ... JUSQU'A....57
4. Boucles à bornes non définies en Pascal : repeat ... until....58
5. Boucles à bornes non définies : Comparaison de deux boucles....59

VI - Chapitre 6 - Tableaux 61

A. Tableaux à une dimension....61
1. Définition....61
2. déclaration d'un type tableau....61
3. Déclaration d'une variable de type tableau....62
4. Ecriture dans un tableau....62
5. Premier exemple d'écriture dans un tableau....63
6. Second exemple d'écriture dans un tableau....63

B. Tableaux à plusieurs dimensions....64
1. Tableau à 2 dimensions....64
2. Exemples....64
3. Exemple complet....65

VII - Chapitre 7 - Chaînes de caractères 67

A. Définition et méthodes....67
1. Définition....67
2. Opérateurs....68
3. Fonctions....68
4. Fonctions de codage / décodage des caractères....69

B. Exemples....70
1. Exemple 1....70
2. Exemple 2....70

VIII - Chapitre 8 - Fonctions et Procédures 71

A. Procédures....71
1. Définition et déclaration....71
2. Exemple....71
3. Appel d'une procédure....72

B. Fonctions....72
1. Définition et déclaration....72
2. Appel....73
3. Exemple....74
4. Différence entre procédure et fonction....74

C. Variables globales et variables locales....75
1. Définitions....75
2. Exemple....75
3. Portée des variables....76
4. Exemple....76
5. Remarques....76
6. Local ou global ?....77

D. Paramètres.....78
1. Exemple : 2 solutions.....78
2. Choix de la 2ème solution.....78
3. Passage de paramètre par valeur.....79
4. Exemples de passage par valeur.....79
5. Passage de paramètre par adresse.....80
6. Exemple de passage par adresse.....81
7. Cas des fonctions.....82
8. Paramètres fonctions.....82

IX - Chapitre 9 - Ensembles **85**

A. Définition et exemples.....85
1. Définition et syntaxe.....85
2. Exemple 1.....85
3. Exemple 2.....86

B. Opérations sur les ensembles.....86
1. Union ---> opérateur +.....86
2. Intersection ---> opérateur *.....87
3. Différence ---> opérateur -.....87
4. Egalité ---> opérateur =.....87
5. Inégalité ---> opérateur <>.....87
6. Inclusion ---> opérateurs <= ou >=.....88
7. Appartenance ---> opérateur in.....88

X - Chapitre 10 - Enregistrements **91**

A. Généralités.....91
1. Définition.....91
2. Déclaration.....91
3. Accès aux champs.....92

B. Ecriture dans un enregistrement.....92
1. Méthodes d'écriture.....92
2. Exemples.....93

C. Instruction with.....94
1. Avantage de l'instruction.....94
2. Exemple.....94

D. Tableaux d'enregistrements.....94
1. Utilité des tableaux.....94
2. Exemples.....95

XI - Chapitre 11 - Fichiers **97**

A. Introduction.....97
1. Définition.....97
2. Manipulation des fichiers.....97
3. Les supports physiques.....98

B. Organisation et accès.....98
1. Définition.....98
2. Organisation séquentielle.....98
3. Organisation relative (accès direct).....99
4. Organisation indexée.....99

C. Les fichiers séquentiels en Pascal.....100
1. Définitions.....100
2. Déclaration d'un fichier.....101
3. Création d'un fichier.....101
4. Ecriture dans un fichier.....102
5. Ouverture d'un fichier existant.....102
6. Lecture dans un fichier.....103

7. Association entre un nom interne et un nom externe............................**104**
8. Fermeture d'un fichier..**105**

D. Les fichiers structurés en Pascal..***105***

1. Définition...**105**
2. Lecture d'un fichier...**105**
3. Ecriture dans un fichier..**106**

E. Les fichiers de texte...***107***

1. Définition...**107**
2. Déclaration..**107**
3. Ecriture dans un fichier texte..**107**
4. Lecture d'un fichier texte...**108**

XII - Chapitre 12 - Récursivité ***109***

A. Principe...***109***

1. Définition...**109**
2. Exemple de programmation itérative...**109**
3. Exemple de programmation récursive..**110**

B. Exemple complet : Somme des n premiers entiers...............................***110***

1. Objectif..**110**
2. Réalisation grâce à la récursivité..**110**
3. Remarque et exemple d'exécution...**111**

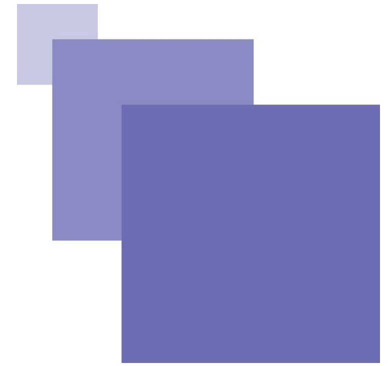
C. Exemples simples..***112***

1. Puissance d'un entier...**112**
2. PGCD Récursif..**112**
3. Exercice d'application...**113**

D. Exemple complet : Tours de HANOI..***113***

1. Problème...**113**
2. Analyse...**114**
3. Programme..**114**
4. Exemple d'exécution...**115**

Introduction



Aujourd'hui, il est indispensable pour tout ingénieur d'avoir eu un premier contact avec l'informatique. L'intérêt d'une telle Unité de Valeur est de permettre aux futurs ingénieurs de mieux comprendre la façon de réaliser des programmes, afin de pouvoir plus facilement utiliser l'ordinateur pour résoudre des problèmes scientifiques, techniques ou organisationnels.

L'objectif de cet enseignement est d'une part de familiariser l'étudiant à l'utilisation de l'ordinateur et d'autre part, de commencer l'apprentissage de l'algorithmique et de la programmation.

L'étudiant devra ainsi apprendre à trouver la solution d'un problème à l'aide d'une forme algorithmique et savoir programmer cette solution algorithmique en un langage structuré.

Il s'agit d'un premier contact avec la programmation. L'étudiant apprend à réaliser des algorithmes puis à les transformer en petits programmes réels.

Aucune connaissance préalable n'est requise. Cette UV s'adresse à des débutants en informatique. Il s'agit d'étudiants de début de tronc commun, destinés à aller dans n'importe quelle branche de l'UTC (mécanique, procédés, systèmes urbains, systèmes mécaniques, biologique, informatique).

Chapitre 1 - Introduction

I

Ordinateur et programmation	11
Architecture d'un ordinateur	11
Langages	12
Représentation de l'information	13
Rappels sur la numération	13
Conversions	15
Tables ASCII	17

A. Ordinateur et programmation

L'informatique intervient aujourd'hui dans de nombreux secteurs d'activité. Parmi les applications courantes on peut citer la bureautique, la gestion, le calcul scientifique, la communication, l'accès à des ressources d'information (au travers d'internet en particulier), le multimédia, les jeux etc.

Ces applications ne sont possibles que grâce à un ordinateur. Cependant, l'ordinateur seul ne suffit pas. Pour chaque application, il est nécessaire de lui fournir un logiciel (ou programme) adapté.

La programmation est donc une activité fondamentale en informatique. La programmation peut être vue comme l'art de déterminer un algorithme (une démarche) pour résoudre un problème et d'exprimer cet algorithme au moyen d'un langage de programmation.

B. Architecture d'un ordinateur

1. Matériel

En première approche, un ordinateur est constitué des éléments suivants :

- une unité centrale (contenant le processeur),
- une mémoire centrale,
- des organes périphériques permettant :
  - la communication avec l'utilisateur : écran, clavier, souris, imprimante ...
  - le stockage : disque dur, lecteurs de cd, de dvd, de bandes, ...
- des composants matériels divers : cartes son, vidéo, cartes d'acquisition ...

2. Logiciel

Un ordinateur ne peut pas fonctionner seul. Il doit être doté d'un **système d'exploitation**. (Ex : windows, unix, mac os, linux, ...)

Le système d'exploitation est le programme de base d'un ordinateur.

Ce programme permet notamment :

- la gestion de la mémoire,
- la gestion des périphériques,
- l'exécution des programmes,
- la gestion des fichiers.

Les programmes (ou logiciels) d'application s'exécutent généralement en s'appuyant sur le système d'exploitation.

Ces programmes peuvent être très divers : logiciels de bureautique (traitements de textes, tableurs, présentation graphique...), logiciels de calcul, systèmes de gestion de bases de données, environnements de programmation, logiciels de jeux, ...

3. Définitions et unités de mesure

Un **bit** (binary digit) est un élément binaire. Sa valeur est donc 0 ou 1.

Un **octet** (ou byte) est un ensemble de 8 bits.

Les longueurs couramment utilisées sont des ensembles de 16, 32 ou 64 bits.

Un kilo-octet (abréviation : **Ko**) correspond à 1024 bits, soit 2^{10} bits.

Un méga-octet (**Mo**) correspond à 1024 Ko, soit 2^{10} Ko.

Un giga-octet (**Go**) est un ensemble de 1024 Mo, soit 2^{10} Mo.

Ces unités de mesures sont fréquemment utilisées pour indiquer des tailles (ou capacités) de mémoires.

C. Langages

Les données et les instructions doivent être codées en binaire. Ce codage n'est pas réalisé par l'utilisateur, ni même en général par le programmeur. Il est réalisé automatiquement par des programmes utilitaires.

Le **langage machine** est le seul langage directement compréhensible par la machine (l'ordinateur). Un programme écrit en langage machine est une succession de 0 et de 1 définissant des opérations précises à effectuer. Ce langage n'est pas utilisé directement pour la programmation.

Le premier langage utilisable pour programmer un ordinateur est l'assembleur. Le **langage assembleur** dépend du processeur de la machine. Ses instructions sont proches de celles du langage machine, mais leur forme est plus utilisable par un programmeur.

Ex : STO (pour store : stocker, ranger en mémoire), JMP (pour jump : branchement en un point du programme)

L'assembleur ne permet de réaliser que des programmes relativement simples, qui dépendent de l'ordinateur utilisé. Pour réaliser des programmes plus complexes et moins dépendants de la machine, il est nécessaire d'utiliser un **langage de programmation**.

Il existe de nombreux langages de programmation : C, C++, C#, Java, Basic,

Pascal, Lisp, Prolog, Fortran, Cobol, ... Le langage Pascal est utilisé dans ce cours en raison de son caractère pédagogique.

D. Représentation de l'information

Toute information doit être codée en binaire pour être exploitable par un ordinateur. C'est le cas pour les nombres et les caractères (voir ci-dessous), mais aussi pour les sons, les images et les vidéo qui doivent être numérisés.

Nombres

Le codage dépend du type : entier naturel, entier relatif, réel, ...

Par exemple, si le nombre possède un signe, il est nécessaire de représenter ce signe. Un moyen simple pour cela est d'utiliser le premier bit (par exemple 0 pour + et 1 pour -), et de représenter le nombre en binaire ensuite.

Ainsi sur un octet on peut représenter les nombres de -128 (1111 1111) à + 127 (0111 1111). Ce n'est cependant pas ce code qui est utilisé généralement. On lui préfère un code plus complexe mais plus efficace (qui sort du cadre de ce cours).

Le nombre maximum représentable dépend du nombre de bits utilisables

Codage des caractères

Les caractères doivent eux-aussi être représentés par des codes binaires.

Les caractères sont non seulement les lettres (majuscules et minuscules), mais aussi les chiffres, les caractères de ponctuation, l'espace, les caractères spéciaux ...

Un des codes possibles est le code ASCII (American Standard Code for Information Interchange).

Exemple : Code ASCII (voir Tables ASCII, à la fin de ce chapitre)

- la lettre A est codée 41 en hexadécimal, soit 65 en décimal.
- la lettre a est codée 61 en hexadécimal et 91 en décimal

E. Rappels sur la numération

1. Généralités

Le système de numération utilisé habituellement est le système décimal. Un ordinateur étant basé sur le système binaire, il est utile de connaître les systèmes binaire (base 2), hexadécimal (base 16) et octal (base 8), ainsi que les techniques de conversion entre ces différents systèmes.

1.5.1 Système à base quelconque

Tout nombre décimal N peut se décomposer de la façon suivante :

$N = a_n b^n + a_{n-1} b^{n-1} + \ldots.. + a_2 b^2 + a_1 b^1 + a_0 b^0 + a_{-1} b^{-1} + a_{-2} b^{-2} + \ldots.. + a_{-p} b^{-p}$

avec $0 \leq a_i \leq b-1$

Cette décomposition est unique. b est la base du système.

On note généralement : $N = a_n\, a_{n-1} \ldots\ldots a_2\, a_1\, a_0\, ,\, a_{-1}\, a_{-2} \ldots.. a_{-p}$

2. Système décimal

Dans le cas du système décimal :

- a base est 10
- les symboles utilisables sont : 0 1 2 3 4 5 6 7 8 9

Ecriture d'un nombre décimal N quelconque :

$N = a_n 10^n + a_{n-1} 10^{n-1} + ... + a_2 10^2 + a_1 10^1 + a_0 10^0 + a_{-1} 10^{-1} + a_{-2} 10^{-2} + ... + a_{-p} 10^{-p}$

avec $0 \leq a_i \leq 9$

Ou encore : $N = a_n\, a_{n-1} ......a_2\, a_1\, a_0\, ,\, a_{-1}\, a_{-2} .....\, a_{-p}$

Exemple

$123{,}45 = 1x10^2 + 2x10^1 + 3x10^0 + 4x10^{-1} + 5*10^{-2}$

3. Le système binaire

Dans le cas du système binaire :

- la base est 2
- les symboles utilisables sont : 0 1

Représentation d'un entier naturel N :

$N = a_n 2^n + a_{n-1} 2^{n-1} + ... + a_2 2^2 + a_1 2^1 + a_0 2^0 + a_{-1} 2^{-1} + a_{-2} 2^{-2} + ... + a_{-p} 2^{-p}$

avec $0 \leq a_i \leq 9$

Exemple

$1010{,}101 = 1x2^3 + 0x2^2 + 1x2^1 + 0x2^0 + 1*2^{-1} + 0*2^{-2} + 1*2^{-3}$

1010,101 = 8 + 2 + 0,5 + 0,125

1010,101 = 10,625 en base 10

$1010{,}101 = 10{,}625_{10}$

Quel est le nombre de bits nécessaires à la représentation d'un nombre N donné ?

Soit k ce nombre. On a : $2^{k-1} \leq N \leq 2^k$

Il faut donc : $k = E(\log_2 N) + 1$ bits

4. Autres systèmes

Le système octal

- la base est 8
- Les symboles utilisables sont : 0 1 2 3 4 5 6 7

Système hexadécimal

- la base est 16
- Les symboles utilisables sont : 0 1 2 3 4 5 6 7 8 9 A B C D E (A correspond à 10 en décimal, B à 11, ..., F à 15)

F. Conversions

1. Correspondances entre bases

Il est recommandé de bien connaître la correspondance des 16 premiers nombres dans les quatre bases

Décimal	Binaire	Octal	Hexa décimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

Tableau 1 : Table de conversions des 16 premiers symboles

2. Conversion base b – système décimal

On développe le nombre selon les puissances de la base b.

Exemple

$1010,101 = 1x2^3 + 0x2^2 + 1x2^1 + 0x2^0 + 1*2^{-1} + 0*2^{-2} + 1*2^{-3}$

$1010,101 = 8 + 2 + 0,5 + 0,125$

$1010,101 = 10,625_{10}$

3. Conversion système décimal – base b

On applique le principe de la division euclidienne :

$n = b * q + r$ avec : $0 <= r < b$

On fait des divisions euclidiennes des quotients successifs par b jusqu'à ce que l'un des quotients devienne inférieur à b-1.

La liste **inversée** des restes ainsi obtenus constitue la décomposition recherchée.

Exemple

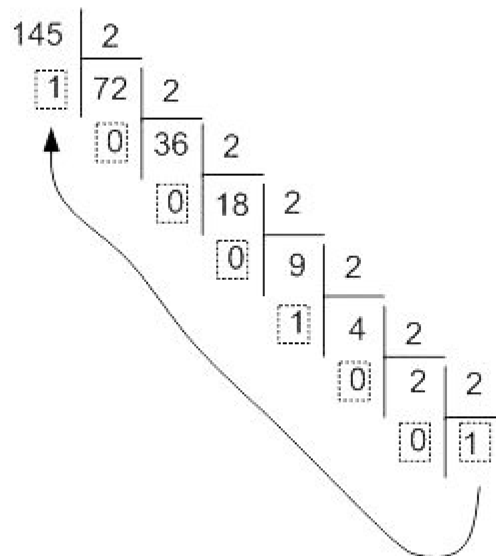


Image 1 : Décimal --> Binaire

Ainsi, on a : $145_{10} = 10010001_2$

Exemple

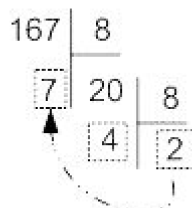


Image 2 : Décimal --> Octal

Ainsi, on a : $167_{10} = 247_8$

Exemple

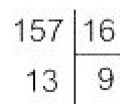


Image 3 : Décimal --> hexadécimal

Donc : $157_{10} = 9D_{16}$

4. Conversion Octal <--> Binaire

Chaque chiffre du nombre en base octale peut être converti en un nombre binaire de trois chiffres (S'il en comporte moins de 3, on complète par des zéros à gauche).
Il suffit donc de regrouper les bits par 3, car nous sommes en base 8
(8 = 2^3, d'où la nécessité de 3 bits).

Exemple

011 000 111 --> 307_8
72_8 --> 111 010
(le zéro en 4ème position a été ajouté car 2 en binaire ne comporte que 2 chiffres).

5. Conversion Hexadécimal <--> Binaire

Cette fois-ci, chaque chiffre du nombre en base hexadécimale peut être représenté par un nombre de 4 chiffres en binaire. On complète à gauche par des zéros si nécessaire.

On regroupe les bits par 4, car nous sommes en base 16

(et 16 = 2^4 , d'où la nécessité de 4 bits).

Exemple

$B5E_{16}$ --> 1011 0101 1110 1100 0111 --> $C7_{16}$

6. Conversion Octal <--> Hexadécimal

Dans ce cas, il est plus simple est de passer par la base binaire, puis de reconvertir dans la base désirée, plutôt que d'utiliser la division euclidienne.

Exemple

307_8 --> 011 000 111 = 1100 0111 --> $C7_{16}$

Ainsi, on convertit chaque chiffre octal en un nombre binaire de 3 bits (conversion octal <--> binaire), puis on regroupe les bits (chiffres binaires) par 4, pour passer en hexa (conversion binaire <--> hexa).

G. Tables ASCII

Ctrl	Dec	Hex	Char	Code	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
^@	0	00		NUL	32	20	sp	64	40	@	96	60	`
^A	1	01	☺	SOH	33	21	!	65	41	A	97	61	a
^B	2	02	☻	STX	34	22	"	66	42	B	98	62	b
^C	3	03	♥	ETX	35	23	#	67	43	C	99	63	c
^D	4	04	♦	EOT	36	24	$	68	44	D	100	64	d
^E	5	05	♣	ENQ	37	25	%	69	45	E	101	65	e
^F	6	06	♠	ACK	38	26	&	70	46	F	102	66	f
^G	7	07	•	BEL	39	27	'	71	47	G	103	67	g
^H	8	08	◘	BS	40	28	(	72	48	H	104	68	h
^I	9	09	○	HT	41	29	)	73	49	I	105	69	i
^J	10	0A	◙	LF	42	2A	*	74	4A	J	106	6A	j
^K	11	0B	♂	VT	43	2B	+	75	4B	K	107	6B	k
^L	12	0C	♀	FF	44	2C	,	76	4C	L	108	6C	l
^M	13	0D	♪	CR	45	2D	-	77	4D	M	109	6D	m
^N	14	0E	♫	SO	46	2E	.	78	4E	N	110	6E	n
^O	15	0F	☼	SI	47	2F	/	79	4F	O	111	6F	o
^P	16	10	►	DLE	48	30	0	80	50	P	112	70	p
^Q	17	11	◄	DC1	49	31	1	81	51	Q	113	71	q
^R	18	12	↕	DC2	50	32	2	82	52	R	114	72	r
^S	19	13	‼	DC3	51	33	3	83	53	S	115	73	s
^T	20	14	¶	DC4	52	34	4	84	54	T	116	74	t
^U	21	15	§	NAK	53	35	5	85	55	U	117	75	u
^V	22	16	▬	SYN	54	36	6	86	56	V	118	76	v
^W	23	17	↨	ETB	55	37	7	87	57	W	119	77	w
^X	24	18	↑	CAN	56	38	8	88	58	X	120	78	x
^Y	25	19	↓	EM	57	39	9	89	59	Y	121	79	y
^Z	26	1A	→	SUB	58	3A	:	90	5A	Z	122	7A	z
^[	27	1B	←	ESC	59	3B	;	91	5B	[	123	7B	{
^\	28	1C	∟	FS	60	3C	<	92	5C	\	124	7C	¦
^]	29	1D	↔	GS	61	3D	=	93	5D	]	125	7D	}
^^	30	1E	▲	RS	62	3E	>	94	5E	^	126	7E	~
^_	31	1F	▼	US	63	3F	?	95	5F	_	127	7F	⌂†

Image 4 : Table ASCII 1

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	φ
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	░	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	

Image 5 : Table ASCII 2

Chapitre 2 - Algorithmes et langages

II

Algorithmes	21
Notion de variable	22
Représentations d'un algorithme	24
Grammaires et langages	27

A. Algorithmes

1. Définition

Définition

Etant donné un traitement à réaliser, un algorithme pour ce traitement est l'énoncé d'une séquence d'actions primitives permettant de le réaliser.

2. Exemples

Exemple : Pour sortir une voiture du garage :

1. Ouvrir la porte du garage
2. Prendre la clef
3. Ouvrir la porte avant gauche
4. Entrer dans la voiture
5. Mettre au point mort
6. Mettre le contact

Exemple : Résolution de l'équation du premier degré : ax +b = 0 dans R :

```
1. lire les coefficients a et b
2. si a = 0 alors
      si b = 0 alors
            afficher ("l'ensemble des solutions est R")
      sinon
            afficher ( "pas de solution")
      fin de si
    sinon
      solution  - b / a
      afficher ("La solution est : ", solution)
    fin de si
```

Remarque

- Les instructions utilisées sont : lire, afficher, si ... alors ... sinon ..., <- (affectation)
- Les symboles a et b représentent les données de l'algorithme
- Le symbole solution représente une variable de l'algorithme

3. Règles de base

Fondamental : Propriétés nécessaires

Un algorithme :

- ne doit pas être ambigu
- doit être une combinaison d'opérations élémentaires
- doit fournir un résultat en un nombre fini d'opérations, quelles que soient les données d'entrée.

Méthode : Première approche de méthode

- Définir clairement le problème
- Etablir l'algorithme au moyen d'une analyse descendante
  - Déterminer une séquence d'instructions de niveau supérieur (sans entrer dans les détails)
  - Ecrire chaque instruction de niveau supérieur à l'aide d'instructions élémentaires
- Ecrire le programme et la documentation
- Tester
- Revenir sur les étapes précédentes si nécessaire

B. Notion de variable

1. Notions

Définition : Donnée

Une donnée est une valeur introduite par l'utilisateur pendant l'exécution du programme :

- directement (clavier, souris)
- ou indirectement (fichier, base de données).

Dans le second exemple du chapitre precedent (partie exemples) les données sont a et b.

Définition : Constante

Une **constante** est une valeur fixe utilisée par le programme. Exemple : Pi, la constante de gravitation, etc.

Définition : Variable

Une variable représente une valeur qui peut changer en cours d'exécution.

Exemples :

- L'inconnue dans une équation
- La vitesse d'un objet en chute libre

Représentation

A une donnée, une variable ou une constante sont associés :

- un nom (ou identificateur),
- un type qui détermine l'ensemble des valeurs possibles,
- une zone mémoire dont la taille dépend du type.

Image 6 : representation

2. Affectation

Définition

L'affectation est l'opération qui consiste à attribuer à une variable la valeur d'une expression.

Notation :

variable ← expression

Complément

L'affectation a donc un double rôle :

- elle détermine la valeur de l'expression à droite de ←
- elle range le résultat dans la variable située à gauche.

Exemple

- z ← 1 (z prend la valeur 1)
- resultat ← 2*3+5 (resultat prend la valeur du résultat de l'opération 2*3+5, i.e. 11)
- solution ← -b / a (-b/a est évalué à l'aide des valeurs des variables a et b. Le résultat de cette évaluation est affecté à solution)
- nb ← nb+1 (nb augmente de 1)

3. Structure de sélection simple

Syntaxe

si <condition> **alors**

< séquence d'instructions >

fin de si

ou

si <condition> **alors**
  < séquence d'instructions >
sinon
  < séquence d'instructions >
fin de si

Exemple : Maximum de deux nombres

si a ≥ b alors
  max ← a
sinon
  max ← b
fin de si

4. Structures répétitives

Syntaxe

tant que <condition> **faire**
  <séquence d'instructions >
Ftq
ou
répéter
  <séquence d'instructions>
jusqu'à condition

Exemple

répéter
  ecrire(« entrez un nombre inférieur à 10 »)
  lire(n)
jusqu'à n < 10

C. Représentations d'un algorithme

1. Representation

Ecriture algorithmique

- langage semi-naturel (pas de syntaxe précise)
- écriture plus ou moins détaillée
- dépendant de la nature du traitement
- censée être comprise par le lecteur

Complément

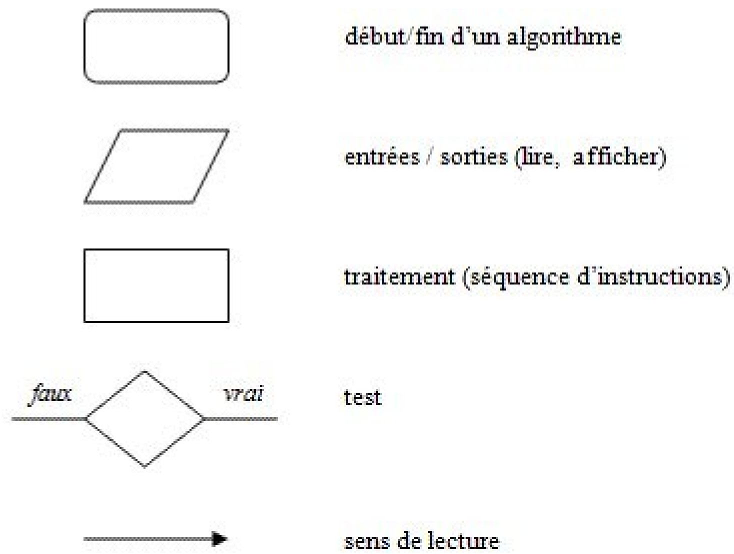


Image 7 : organigramme

2. Exemples

Exemple : Calcul du PGCD

Le pgcd de deux nombres entiers positifs n1 et n2 est leur plus grand diviseur commun. On suppose : n1 ≥ n2 :

Exemple : PGCD de 30 et 8 = 2

L'algorithme est basé sur la division euclidienne :

a=b * q + r avec r < b

PGCD(a,b) = PGCD(b,r)

n1	n2	r
30	8	6
8	6	2
6	2	

Image 8 : tableau

1. Lire les deux entiers naturels a et b
2. r ← a div b
3. Si R est différent de 0 alors
    a ← b
    b ← r
    revenir au point 2
   sinon
    pgcd ← b
4. Afficher le résultat : pgcd

Exemple : Calcul du salaire net d'un employé

On désire calculer un salaire net d'un employé à partir du salaire horaire brut, du nombre d'heures effectuées et du pourcentage de charges à retenir sur le salaire brut.

Données :

- SH : le salaire horaire
- NH : le nombre d'heures
- PR : le % de retenues

Calculer : SB ← SH * NH { le salaire de base }
R ← SB * PR : { les retenues }
SN ← SB - R : { le salaire net }
Ecrire le résultat :
"Salaire net" = SN

Exemple : Calcul du salaire net avec retenues plafonnées :

Cette fois-ci, on introduit un plafond pour les charges retenues sur le brut. On écrit alors un algorithme avec condition. En cas de dépassement du plafond, on ne retient que le plafond.

Données :

- SH : le salaire horaire
- NH : le nombre d'heures
- PR : le % de retenues non plafonnées
- PL : le plafond

Calculer :
SB ← SH*NH { le salaire de base }
R ← SB*PR { les retenues }
Si R > PL alors R ← PL
SN ← SB-R { le salaire net }

Ecrire le résultat :
"Salaire net" = SN

On peut également représenter cet algorithme sous la forme d'un organigramme, comme indiqué ci-après :

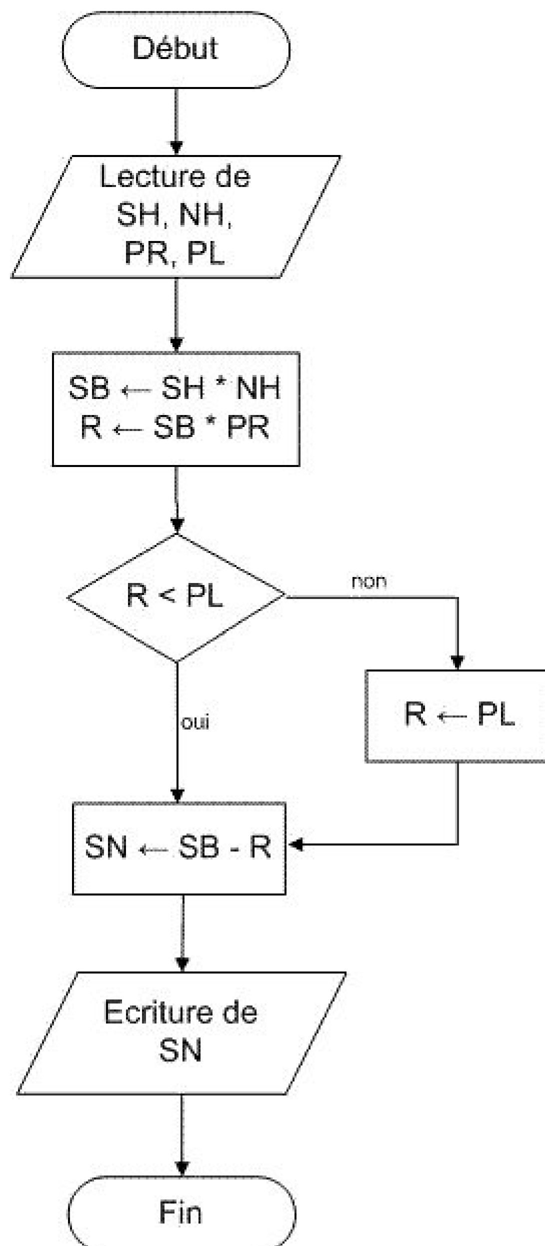


Image 9 : organigramme

D. Grammaires et langages

1. Introduction

Comme les langages naturels, les langages informatiques utilisent une grammaire (ou syntaxe). La syntaxe d'un langage de programmation est cependant plus rigide et dispose d'un vocabulaire plus limité.

Différents formalismes de représentation d'une grammaire ont été définis. Nous considérons ici :

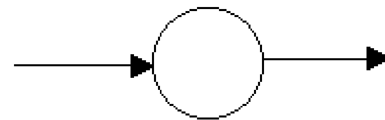
- le formalisme BNF (Backus Naur Form)
- les diagrammes de Conway

Pour obtenir une phrase correcte du langage, il faut partir d'un concept initial (symbole ou atome), puis dériver en appliquant des règles, jusqu'à obtenir un texte uniquement composé de symboles terminaux.

2. Diagrammes de Conway

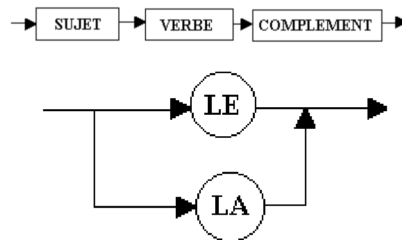
Un rectangle représente une catégorie syntaxique.

Un cercle représente un symbole terminal.



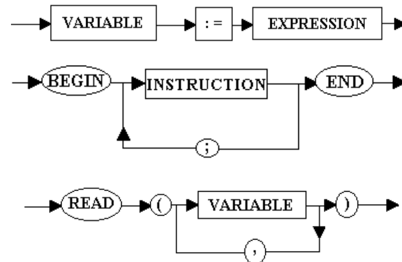
Exemple

- PHRASE
- ARTICLE



Exemple : Exemples pour la syntaxe du Pascal

- Affectation
- Instruction composée (liste d'instructions)
- Lecture



3. Formalisme BNF (Backus-Naur Form)

Définition

Ce formalisme permet de définir des règles de dérivation.

Exemple

<alternative> ::= SI <condition> ALORS <instruction>

<alternative> ::= SI <condition> ALORS <instruction> SINON <instruction>

- SI, ALORS et SINON sont des symboles terminaux,
- <condition> est une catégorie syntaxique,
- ::= est un méta-symbole signifiant « peut être défini par ».

Définition

Un programme est une phrase correcte du langage dérivée à partir d'un symbole initial et ne contenant que des symboles terminaux.

En Pascal : <programme> ::= program <identificateur> ; <bloc>.

4. Programmation

Ecriture du programme

Après avoir déterminé l'algorithme, il faut écrire le programme source en respectant une syntaxe très précise, définie par des règles de grammaire dépendant du langage utilisé. Le programme source peut être écrit à l'aide d'un éditeur de texte tel que le bloc-notes de windows. On préférera cependant utiliser un EDI ou « environnement de développement intégré ». Un EDI facilite l'écriture, la mise au point et l'exécution des programmes.

Compilation

Un programme appelé « compilateur » verifie que le programme source respecte la grammaire du langage et le traduit en langage objet, plus proche de la machine. Un second programme appelé « éditeur de liens » rend ensuite le programme exécutable par la machine.

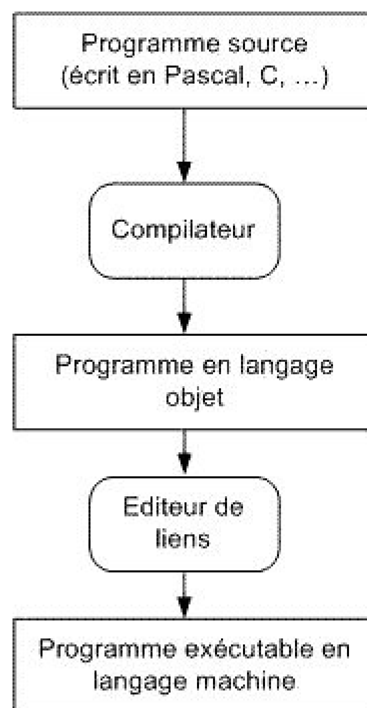
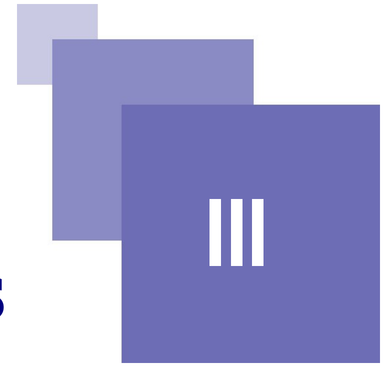


Image 10 : compilation

Remarque

Un programme peut être syntaxiquement correct sans pour autant fournir les résultats attendus. On distingue en effet deux types d'erreurs : les erreurs de syntaxe et les erreurs de logique.

Chapitre 3 - Premiers éléments de Pascal



Structure globale d'un programme Pascal	31
Exemple de programme structuré	33
Grammaire d'un programme Pascal	34
Déclarations	36
Entrées / Sorties	42
Instruction d'affectation	45
Opérateurs et fonctions arithmétiques	45
Programmation structurée	47

C'est un langage typé

- Toutes les variables doivent être déclarées
- Leur type doit être explicitement défini

C'est un langage structuré

- Le langage permet de définir des procédures et fonctions qui sont des sortes de sous-programmes (Cf. chapitre 8).
- Un problème peut ainsi être décomposé en sous-problèmes.

C'est un langage récursif

- Les procédures et fonctions peuvent « s'appeler » elles-mêmes (Cf. chapitre 12).

A. Structure globale d'un programme Pascal

Structure globale :

```
En-tête
Déclarations
    Constantes
    Types
    Variables
    Fonctions / Procédures
Bloc d'instructions exécutables
```

1. En-tête

Définition

C'est la première ligne d'un programme PASCAL.

L'en-tête commence toujours par le mot-réservé **program**. Elle est suivie d'un identificateur choisi par le programmeur.

Syntaxe

```
program identificateur;
```

Exemple

```
program Second_Degre ;
```

2. Déclarations

En Pascal, on peut déclarer :

- des constantes
- des types
- des variables

Attention

L'ordre indiqué doit être impérativement respecté.

3. Instructions

Définition

Une **instruction** est une phrase du langage représentant un ordre ou un ensemble d'ordres qui doivent être exécutés par l'ordinateur.

On distingue les **instructions simples** et les **instructions structurées**.

Définition : Les instructions simples

- Ordre unique, inconditionnel (Ex : affectation)

Définition : Les instructions structurées

- instructions composées
- instructions itératives
- instructions conditionnelles

Remarque : Quelques caractéristiques des instructions :

- Pas de format fixe
- Possibilité de spécifier une instruction sur plusieurs lignes ou plusieurs instructions sur une seule ligne
- Début d'une instruction : un mot clé ou un identificateur
- Fin d'une instruction : par un point virgule ;

4. Structure de bloc

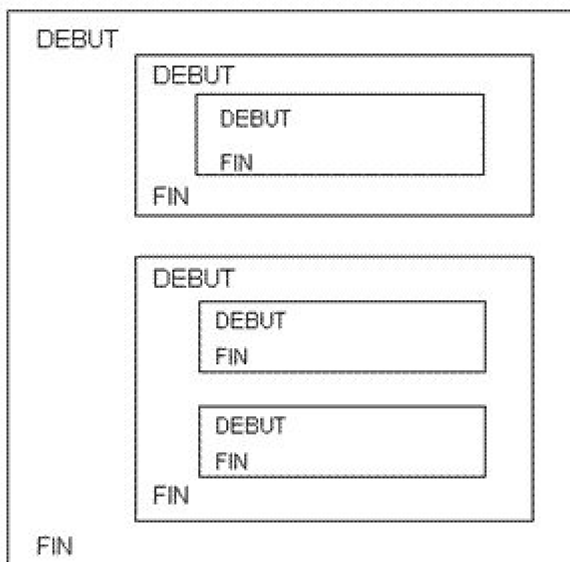


Image 11 : Structure de bloc

Syntaxe

En Pascal, le bloc d'instructions principal commence par « **begin** » et se termine par « **end**. »

B. Exemple de programme structuré

Nous présentons ici un programme qui donne la moyenne de n nombres entrés au clavier par l'utilisateur.

L'utilisateur doit préciser le nombre de données qu'il va entrer.

A ce stade du cours, il n'est pas nécessaire de comprendre le contenu de ce programme. Il suffit simplement de **reconnaître l'architecture globale** décrite précédemment (déclarations de variables, blocs, indentations, begin...end)

1. Code source

```
program moyenne ;                                    {En-tête}
var                                                 {Déclarations}
    donnee, somme, moyenne : real;
    i, n : integer ;
begin                                               {début du bloc
d'instructions}
    writeln('entrer le nombre de données');
    readln(n);
    if n > 0 then
```

```
    begin
       somme := 0;
       for i := 1 to n do
       begin
          read(donnee);
          somme := somme + donnee;
       end;
       moyenne := somme / n;
       writeln('moyenne =',moyenne);
    end
    else
       writeln('pas de donnees');
end.                                        { fin du bloc
d'instructions}
```

2. Remarques

Conseil

Il est conseillé d'utiliser :

- des indentations pour refléter la structure du programme
- des commentaires pour souligner des points importants.

Syntaxe

Un commentaire est un texte encadré par des accolades ou par les caractères (* et *).

Exemple

```
{ ceci est un commentaire } (* en voici un autre*)
```

Remarque

Un commentaire peut être ajouté en n'importe quel point du programme.

Les commentaires sont destinés à faciliter la lecture et la compréhension du programme par les programmeurs.

Ils n'ont aucune incidence sur le fonctionnement du programme.

C. Grammaire d'un programme Pascal

1. L'Alphabet

L'alphabet Pascal est constitué des éléments suivants :

- Les majuscules : A, B,..., Z (26 caractères)
- Les minuscules : a, b,..., z (26 caractères)
- Le caractère "blanc"
- Les chiffres : 0, 1,..., 9
- Les symboles spéciaux
- Les opérateurs :

  - arithmétiques : + - * /
  - relationnels :< ; > ; = ; <= ; >= ; <>
- Les séparateurs : ( ) ; { } ; [ ] ;(* *)
- Le signe "pointeur" : ^
- Les signes de ponctuation : . , ; : ' ` ! ?

2. Les mots du langage : définition

Définition : Mot

Un **mot** est une suite de caractères encadrés par des espaces ou des caractères spéciaux.

Définition : Mot réservé

Certains mots sont **réservés**. Ils ne peuvent être redéfinis par l'utilisateur, parce qu'ils participent à la construction syntaxique du langage.

Exemple : Exemples de mots réservés :

const var type array record begin end procedure
function if then else case while repeat for in
until with do and or ...

3. Les mots du langage : les identificateurs

Définition

Un **identificateur** est un nom donné à un élément du programme (constante, variable, fonction, procédure, programme, ...) par le programmeur.

En pascal :

- Un identificateur est une suite alphanumérique commençant nécessairement par une lettre de l'alphabet et ne comportant pas d'espaces.
- Il est possible de lier plusieurs mots à l'aide de " _ ".

Exemple : Exemples d'identificateurs légaux

x2 Z31 xBarre SOMME salaire_net

Exemple : Exemples d'identificateurs non légaux

3Mots U.T.C. mot-bis A!8 $PROG AUTRE MOT

4. Les mots du langage : les identificateurs standards

Définition

Les **identificateurs standards** sont des identificateurs prédéfinis ayant une signification standard. A la différence des mots réservés, ils peuvent être redéfinis par le programmeur (mais c'est fortement déconseillé).

Exemple : Exemples d'identificateurs standards

Fonctions :

cos sin exp sqr sqrt succ pred

Constantes :

maxint true false

Types :

integer real boolean char

Procédures :

read write reset rewrite

D. Déclarations

En Pascal, tout symbole (constante, type, variable, procédure, fonction) utilisé dans un programme doit être explicitement **déclaré**.

1. Les constantes

Conseil

L'utilisation de constantes en programmation est vivement conseillée.

Définition

Les constantes permettent :

- de clarifier le programme (Exemple : PI à la place de 3,141592653)
- de faciliter la modification : il suffit de modifier la valeur spécifiée dans la déclaration au lieu d'en rechercher les occurrences et de les modifier dans tout le programme.

Syntaxe

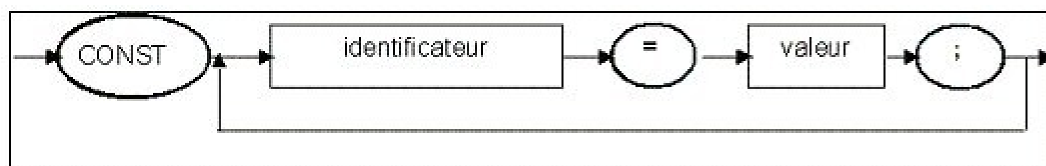


Image 12 : Syntaxe des constantes

Exemple

```
const
  DEUX = 2;
  PI = 3.14;
  VRAI = true;
  FAUX = false;
  CAR_A = 'A';
  PHRASE = 'il fait beau';
```

Attention

Le point virgule est obligatoire à la fin de chaque déclaration

2. Les types : définitions

Définition : Type

Un **type** définit l'ensemble des valeurs que peut prendre une donnée.

Il existe des types standard, mais on peut également déclarer de nouveaux types.

Définition : Type standard

Un **type standard** est un type qui est normalement connu de tout langage Pascal et qui n'a donc pas été déclaré par l'utilisateur.

Les types standards sont: integer, real, boolean, char et string.

Définition : Type scalaire ou non standard

Un type est dit **scalaire** s'il est :

- soit un type scalaire standard (integer ou real),
- soit un type énuméré
- soit un type intervalle.

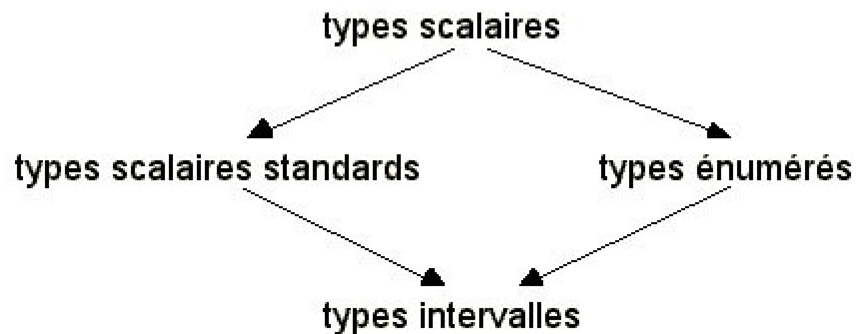


Image 13 : Organisation des types scalaires

3. Type standard : Integer

Définition

Les valeurs correspondant à ce type sont des nombres entiers.

Ce sont des suites de chiffres, éventuellement précédées d'un signe + ou −, qui ne contiennent ni point décimal, ni exposant.

Exemple : Exemples d'entiers corrects

589 0 7 +7 -28 -9999

Exemple : Exemples d'entiers incorrects

79. 644 895

Fondamental

On ne peut pas représenter en mémoire tous les entiers.

Le nombre de bits utilisable pour coder un entier est fixe, mais varie en fonction des compilateurs :

- Sur n bits, il sera possible d'utiliser des entiers compris entre : - 2^{n-1} et (2^{n-1} - 1)

- Par exemple, sur 16 bits , les entiers seront compris entre : -32768 et 32767 (2^{16}=32768)

Complément

La plupart des compilateurs définissent également le type **longint** qui permet de coder des « entiers longs », en doublant le nombre de bits utilisables.

4. Type standard : Real

Définition

Les valeurs sont des réels.

Syntaxe : Représentation décimale

Signe + partie entière + point + partie décimale

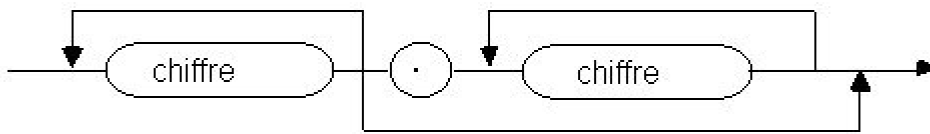


Image 14 : Diagramme syntaxique d'un réel non signé

Exemples de réels correctement écrits :

3.5 -7.80 0 -0.656 +95000.0

Exemples de réels incorrectement écrits :

8. 75,632 100.

Syntaxe : Représentation en virgule flottante

On ajoute un exposant.

Notation : lettre E suivie d'un entier signé ou non signé

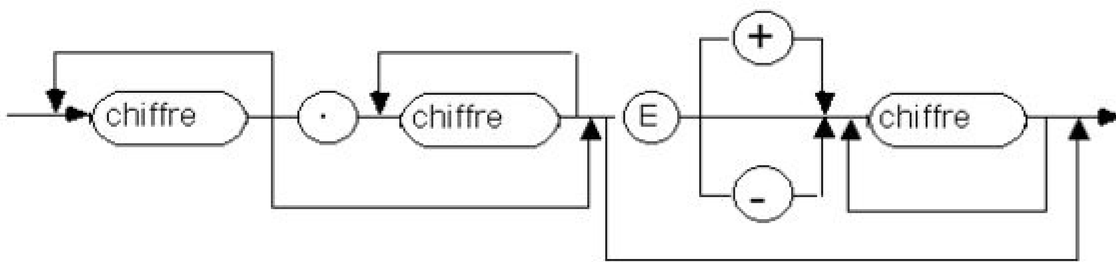


Image 15 : Diagramme syntaxique général (valable aussi bien pour les entiers que les réels)

Exemples d'écriture correcte :

7E-5 -1.2345e2 +60e+4

Exemples d'écriture incorrecte :

45 E3 6.3E2.4 IX 2/3

Il est possible de représenter en mémoire de très grandes valeurs, mais là encore il est impossible de représenter tous les nombres.

5. Type standard : Boolean

Définition

Les valeurs sont dites **logiques**.

Ce sont les deux constantes : *«true»* (vrai) et *«false»* (faux)

Complément

Les opérateurs booléens (ou logiques) tels que **and** ou **or** sont associés à ce type.

6. Type standard : Char

Définition

Les valeurs sont les caractères, en particulier :

- alphanumériques : 'a' . . 'z' 'A' . . 'Z' '0' . . '9'
- le caractère blanc : ' '
- les caractères accentués
- les caractères de ponctuation

Syntaxe

Dans un programme, les valeurs de type char doivent être entre apostrophes (quotes).

Exemple

Exemple : 'd'

7. Type standard : String

Définition

Les valeurs sont des chaînes de caractères.

Syntaxe

Elles sont également représentées entres apostrophes.

Lorsqu'il y a une apostrophe dans la chaîne de caractères, elle doit être doublée.

Exemple

Exemple : 'il fait beau aujourd**''**hui'

Attention

Il ne faut pas confondre avec les **commentaires**, situés entre deux accolades, qui n'interviennent pas directement dans le programme.

Ces commentaires servent au programmeur, lorsqu'il doit reprendre le programme plus tard, pour le modifier.

8. Type scalaire : Le type énuméré

Définition

Un **type énuméré** est une séquence ordonnée d'identificateurs.

Syntaxe

```
type
  identificateur = (id1, id2,..., idn) ;
```

Exemple

```
type
  couleur = (jaune, vert, rouge, bleu, marron );
  semaine =(lundi,mardi,mercredi,jeudi,vendredi,
samedi,dimanche);
  reponse = (oui, non, inconnu);
  sexe = (masculin, féminin);
  voyelle = (A, E, I, O, U);
```

Attention

Le mot réservé type ne doit être écrit qu'une seule fois.

Remarque

- Deux types énumérés différents ne peuvent contenir le même identificateur. Les ensembles énumérés sont donc disjoints.
- La séquence de valeurs étant ordonnée, le système connaît les successeurs et prédécesseurs d'un élément :
  - mardi : prédécesseur de mercredi.
  - mercredi : successeur jeudi.

9. Type scalaire : Le type intervalle

Définition

Un **type intervalle** est nécessairement un sous-type d'un type scalaire (standard ou énuméré) déjà défini.

Toutes les valeurs de l'intervalle sont autorisées.

Syntaxe

```
type
  identificateur = [borne inf] .. [borne sup] ;
```

Exemple : Intervalle d'entiers

```
type
  Decimal = 0 .. 9 ;
  Octal = 0 .. 7 ;
  Age = 0 .. 150 ;
```

Exemple : Intervalle de caractères

```
type
  ABC = 'A' .. 'C' ;
  Maj = 'A' .. 'Z' ;
```

Exemple : Intervalle avec un type non-standard

```
type
  Ouvrable = lundi .. vendredi ;
  WeekEnd = samedi .. dimanche ;
  Lettres = 'A' .. 'Z' ;
```

Remarque

- On ne peut pas définir un type intervalle à l'aide du type «*real*» (type non scalaire).
- L'ordre ascendant est requis : «*borne-inf*» doit être placé avant «*borne-sup*» dans le type énuméré source.

Exemple : Exemples de déclarations incorrectes

type
  Octal= 7 .. 0 ;
  Ouvrable = vendredi .. lundi ;

10. Les variables

Définition

Déclarer une variable, c'est définir l'ensemble des valeurs qu'elle peut prendre.
Toutes les variables utilisées dans un programme doivent être déclarées.
On peut déclarer une variable :

- à l'aide d'un type standard ou d'un type déclaré au préalable.
- par une déclaration explicite (et spécifique à cette variable) de l'ensemble des valeurs qu'elle peut prendre.

Syntaxe

```
var
  identificateur : type ;
```

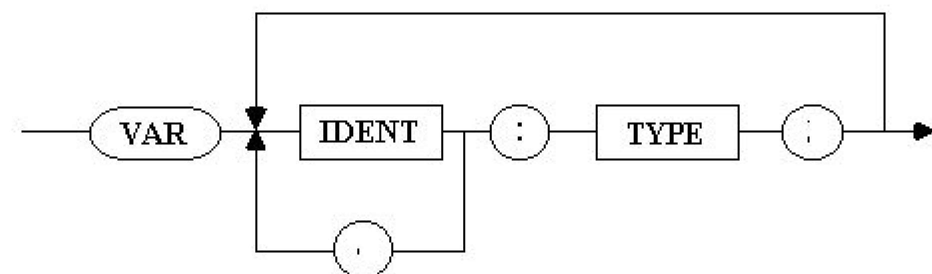


Image 16 : Diagramme syntaxique

Remarque

- var ne peut apparaître qu'une seule fois
- il est possible de grouper plusieurs variables pour le même type (en les séparant par des virgules).

Exemple

```
var
  jour: semaine ;
  a, b, c : real;
  i, j, k : integer ;
  conge : week-end ;
  vivant : boolean ;
```

Exemple : avec déclaration locale explicite

```
var
  lettre : 'A' . . 'Z' ;
  feux : (vert, orange, rouge) ;
```

11. Exemples de déclarations de constantes, types et variables

Exemple

```
const
    JOUR_MAX = 31 ;
    AN_MIN = 1901 ;
    AN_MAX = 2000 ;
type
    Siecle = AN_MIN . . AN_MAX ;
    Semaine = (lundi, mardi, mercredi, jeudi, vendredi,
samedi, dimanche) ;
    Annee = (janvier, février, mars, avril, mai, juin,
juillet, aout, septembre, octobre, novembre, decembre);
var
    mois: annee ;
    jour: semaine ;
    nbJours : 1 .. JOUR_MAX ;
    an: siecle ;
    ouvrable : lundi .. vendredi ;
    i, j : integer ;
    numEtudiant : 1 .. maxint ;
```

E. Entrées / Sorties

Ce sont des échanges d'informations entre la mémoire (variables et constantes) et les périphériques (clavier, écran ...).

Les types autorisés sont : entier, réel, booléen, caractères et chaînes de caractères.

1. Lecture

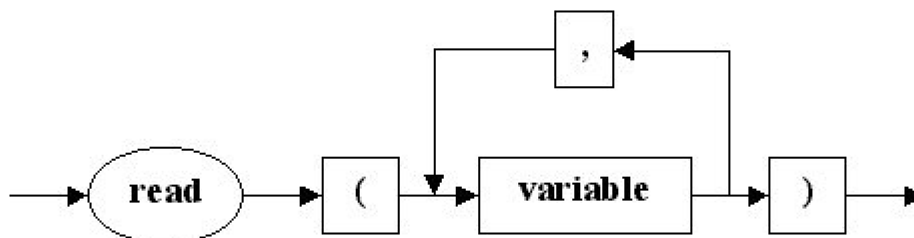


Image 17 : Diagramme syntaxique

Syntaxe : Read

```
read (v1, v2, v3, ..., vn);
```

Cette instruction permet de lire les valeurs entrées par l'utilisateur au clavier et de les stocker dans les variables v1,..., vn

```
read (v1, v2, v3, ..., vn); <=> read(v1); read(v2);...
read(vn);
```

Syntaxe : Readln

```
readln (v1, v2, v3, ..., vn);
```

permet, de manière analogue, de lire n valeurs et passe ensuite à la ligne suivante en ignorant ce qui reste éventuellement sur la ligne.

readln; peut être employé sans paramètre

Exemple

```
program Test;
var
  i, j : integer ; {déclaration des variables}
begin
  read(i,j); {lecture de deux entiers}
end.
```

2. Ecriture

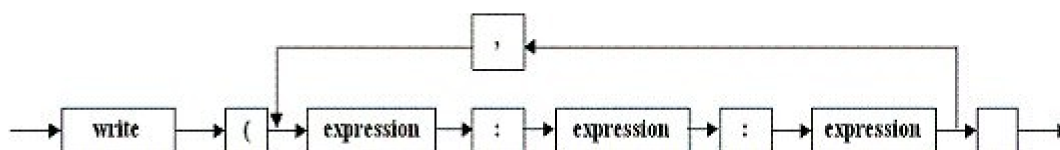


Image 18 : Diagramme syntaxique

Syntaxe

```
write (v1, v2, ..., vn);
writeln(v1, v2, ..., vn); {écrit les valeurs des variables
passées en paramètre puis va à la ligne}
writeln;                    {peut être employé sans paramètre}
```

Exemple

```
program Test;
var
  i : integer;
  r : real;
begin
  write('Entrez un entier et un réel :');
  read(i,r);                              {lecture des
valeurs}
  writeln('L''entier vaut : ', i : 5);     {affichage de i}
  writeln('et le réel est : ', r : 6:2);   {affichage de r}
end.
```

Le programme lit les valeurs de i et r tapées par l'utilisateur et les affiche ensuite à l'écran, sur 5 caractères pour i, et sur 6 caractères pour r, dont 2 chiffres après la virgule.

3. Exemple complet de lectures/ecritures

Exemple

```
program Exemple;
var
  a, b : integer;
  c1, c2 : char;
  resultat : boolean;
  x, y : real;
begin
  write('entrez 2 entiers : ');
  readln(a,b); {lecture des 2 valeurs}
  write('maintenant, entrez une lettre');
  readln(c1); {lecture d'un caractère}
  write('entrez une autre lettre : ');
  readln(c2);
  write('entrez maintenant un réel : ');
  readln(x); {lecture d'un réel}
  writeln;
  resultat := (a<=b); {resultat prend la valeur de
l'expression a<=b}
  writeln('le 2ème entier est-il inf ou égal au 3ème ? =>',
resultat);
  y:=sqr(x); {calcul de x au carré}
  writeln('le carré du réel est : ', y:4:2);
  writeln('ou encore : ',y:8:4);
         {affichage sur 8 caractères, dont 4 chiffres après
la virgule}
  writeln('le code ASCII de ',c1, ' est : ', ord(c1):4);
  resultat := (c1>c2);
         {resultat prend la valeur de l'expression c1>c2}
         {il est vrai si c1>c2, et faux sinon}
  writeln('Le caractère 1 est-il avant le caractère 2 ? =>
', resultat);
  write('Le code de ',c1,' est : ', ord(c1):4);
  writeln(' et celui de ',c2, ' est : ', ord(c2):4);
end.
```

F. Instruction d'affectation

1. Syntaxe et exemples

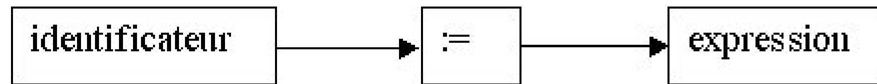


Image 19 : Diagramme syntaxique

Définition

L'instruction d'affectation à un double rôle :

- Evaluation de l'expression (calcul)
- Puis affectation (rangement) dans la variable (identificateur)

Remarque

- Les types doivent être compatibles (les mélanges de types sont interdits)
- Ne pas confondre ":=", l'opérateur d'affectation et "=", l'opérateur de test.

Exemple

Soit X une variable de type integer

X := 10 signifie que l'on affecte la valeur 10 à la variable X, donc X vaut 10 après l'exécution de cette instruction.

On peut tester si X est égal à une certaine valeur avant d'effectuer un calcul :

if X = 3 then X := X / 2 ;

Après exécution de cette instruction, la valeur de X est toujours 10, car le test X = 3 n'est pas vérifié (puisque la valeur 10 a été placée dans X)

G. Opérateurs et fonctions arithmétiques

1. Opérateurs disponibles

Syntaxe

+ somme

- soustraction

* multiplication

/ division

DIV division entière (Ex : 5 div 3 = 1)

MOD modulo (Ex : 5 mod 3 = 2)

Exemple

```
var
  A, B, C, D : real;
  I, J, K : integer;
begin
```

```
  A := 7.4 ;
  B := 8.3 ;
  C := A + B ;
  D := A / B + C ;
  I := 42 ;
  J := 9 ;
  K := I mod J ; { K vaut 6 }
end.
```

2. Expressions

Définition : Expression

Une **expression** est une combinaison d'opérandes (variables et constantes), d'opérateur et de fonctions.

Exemple

- "i+1"
- "2.08E3 * x"
- "(x>2) OR (x<8)"

Définition : Evaluation des expressions

L'**évaluation** utilise les règles de décomposition syntaxique et l'ordre des priorités mathématiques.

Exemple

a*b+c se décompose en :

Expression → Expression simple → Terme + Terme → (Facteur * Facteur) + Facteur

donc a*b+c est équivalent à : (a*b)+c

Exemple

- a>3 and a<10 n'est pas correct (pas de solution possible lors des décompositions)
- (a>3) and (a<10) est correct

3. Fonctions arithmétiques

- ABS (X) : valeur absolue de X
- ARCTAN (X) : arctangente de X
- CHR (X) : caractère dont le numéro d'ordre est X
- COS (X) : cosinus de X
- EXP (X) : exponentielle de X
- LN (X) : logarithme népérien de X
- ORD (X) : numéro d'ordre dans l'ensemble de X
- PRED (X) : prédécesseur de X dans son ensemble
- ROUND (X) : arrondi de X
- SIN (X) : sinus de X
- SQR (X) : carré de X

- SQRT (X) : racine carrée de X
- SUCC (X) : successeur de X dans son ensemble
- TRUNC (X) : partie entière de X

4. Fonctions logiques

- EOF (X) : vrai si la fin de fichier X est atteinte
- EOLN (X) : vrai si fin de ligne du fichier
- ODD (X) : vrai si X est impair, faux sinon

H. Programmation structurée

La programmation structurée consiste à :

- rechercher et à identifier les tâches nécessaires à la résolution d'un problème donné
- organiser l'ensemble de ces tâches
- faire apparaître cette structure dans le programme correspondant.

Pour cela, il faut respecter une certaine discipline de programmation en s'efforçant de satisfaire les exigences suivantes : la **clarté**, la **modularité**, l'**efficacité**.

1. Les exigences

Définition : La Clarté

- Faire des déclarations explicites de toutes les entités manipulées
- Utiliser des noms significatifs (prix pour le prix d'un objet et non pr5...)
- Ne pas employer de méthodes hermétiques
- Ne pas faire de "branchements"
- Utiliser des indentations, c'est-à-dire des "marges décalées"

Définition : La Modularité

- Décomposition du problème en plusieurs sous-problèmes
  - réduction de la complexité
  - synthèse de modules
- Réunion structurée des différents modules

Définition : L'Efficacité

- Conformité des résultats (vérification d'un programme)
- Vitesse d'exécution
- Utilisation optimale de la mémoire

2. Bénéfices attendus

Bénéfices attendus

- Programmation plus simple
- Lecture plus commode
- Modifications plus aisées
- Modules faciles d'accès
- Partage du travail
- Fiabilité supérieure

Chapitre 4 - Instructions alternatives

IV

Choix simple	49
Choix multiple	50
Instructions composées	54

A. Choix simple

1. Définition

Syntaxe

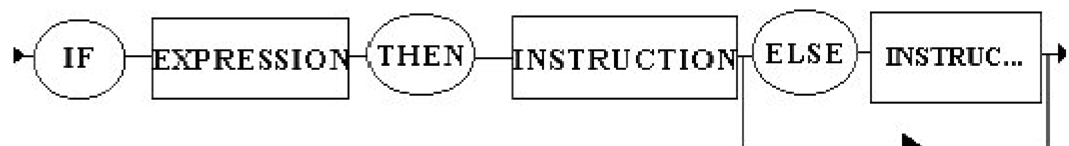


Image 20 : choix simple

Cette instruction évalue l'expression booléenne (condition). Si le résultat est *true*, c'est le premier bloc d'instructions (après *then*) qui est exécuté sinon c'est le second (après *else*).

Remarque : Remarques

- On peut imbriquer plusieurs "if"
- Attention à la présentation : il est souhaitable d'utiliser des indentations (marges décalées) pour augmenter la lisibilité du programme

Attention : Points importants

- Surtout pas de point virgule immédiatement avant ELSE !
- La partie alternative (else + bloc d'instructions) est facultative
- La valeur de la condition doit être booléenne
- Les instructions peuvent être simples ou composées

2. Equation du premier degré

Question

Ecrire un programme qui résout une équation du premier degré : Ax+B = 0. Les valeurs de A et B seront entrées par l'utilisateur.

3. Maximum de deux nombres

Question

Ecrire un programme qui calcule le maximum de deux nombres entrés au clavier.

4. Exemple avec expressions relationnelles et booléennes

Exemple

```
program GRAND_PETIT ;
        var SEX : char ;
        MAJEUR, PETIT, GRAND, FEMME, HOMME : boolean ;
        AGE : 1..120; TAILLE :  50..250;
begin
        read (SEX, AGE, TAILLE) ;
        FEMME :=  SEX='F' ;   {femme est vrai si sex = F}
        HOMME   :=  not FEMME ; {homme vaut le contraire de
femme} MAJEUR  :=  AGE>18 ;
        if  FEMME then
                begin
                        PETIT  :=  TAILLE<150 ;  {petit est
vrai si TAILLE < 150}
                        GRAND  :=  TAILLE>170 ;
                end
        else
                begin
                        PETIT  :=  TAILLE<160 ;
                        GRAND  :=  TAILLE>180 ;
                end ;
        writeln (MAJEUR, FEMME, HOMME) ;
        writeln (AGE, PETIT, GRAND) ;
end.
```

B. Choix multiple

1. Introduction

Cette méthode est utilisée pour tester une solution parmi N.

Par exemple, lorsqu'un menu est proposé à l'utilisateur :

1. lire
2. écrire
3. calculer
4. sortir

il est nécessaire de savoir si l'utilisateur a tapé 1, 2, 3 ou 4.

Au lieu d'utiliser plusieurs *if... then... else...* imbriqués, il est préférable de choisir une sélection multiple (*case* en Pascal).

Exemple

Ainsi au lieu d'écrire :

```
if reponse=1 then
{ instructions de lecture... }
else if reponse=2 then
{ instructions d'écriture... }
else if reponse=3 then
{ instructions de calcul... }
```

Il est préférable d'écrire :

```
case reponse of
1 : {instructions de lecture... }
2 : {instructions d'écriture...}
3 : {instructions de calcul...}
end;
```

2. Définition

Syntaxe (voir la figure page suivante – Diagramme de Conway)

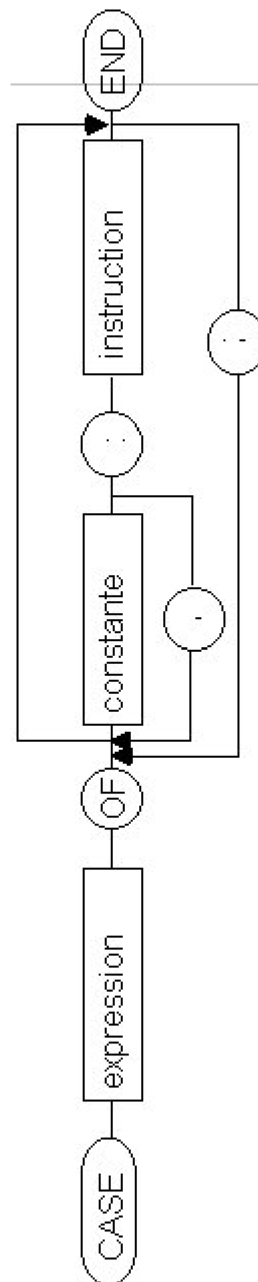


Image 21 : choix multiple

Seules les égalités sont possibles au niveau du test (Pas de comparaisons de type <, >, <=, >= ou <>). On peut néanmoins utiliser des intervalles.

Remarque

L'instruction *case of* est utile :

- pour remplacer des structures *if... then... else...* imbriquées
- pour tester l'égalité à des valeurs données d'une expression

Elle améliore la **lisibilité du programme** !

3. Simuler une calculatrice

Exemple

```
program calculette ;
        var a, b : real ;
        resultat : real;
        touche : char;
begin
        write('entrez une opération ');
        write('(taper un nombre, un opérateur puis un
nombre):');
        readln(A,touche,B);
        case touche of
                '+' : resultat:= a+b;
                '-' : resultat:= a-b;
                '*' : resultat:= a*b;
                '/' : resultat:= a/b;
        end;
        writeln(a,touche,b,' = ',resultat);
end.
```

4. Exemple : le loto

Exemple

```
program bingo ;
        var x : integer ;
begin
        write('entrez un entier : ');
        readln(x);
        case x of
                1..10 : writeln('bingo');
                11..50 : writeln('pas de chance');
        end;
        if x > 50 then
                writeln('valeur non autorisée');
end.
```

C. Instructions composées

1. Définition

Définition

Une **instruction composée** (ou bloc d'instructions) permet de regrouper, dans un même bloc, un ensemble d'instructions qui seront exécutées au même niveau.

Syntaxe

Séquence de deux ou plusieurs instructions comprises entre *begin* et *end* et séparées par des points virgules.

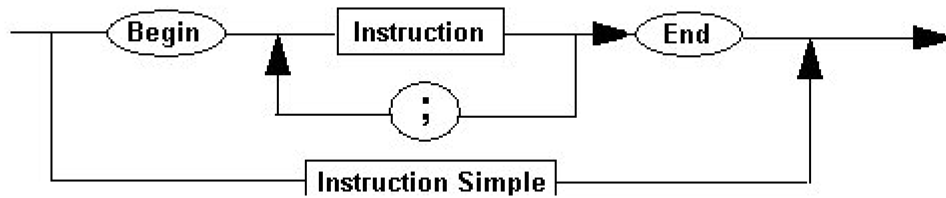


Image 22 : instruction composée

Remarque

Il est possible d'imbriquer des instructions composées. On utilise alors des indentations pour améliorer la lisibilité du programme.

Chapitre 5 - Instructions itératives

Boucles à bornes définies 55
Boucles à bornes non définies 56

A. Boucles à bornes définies

1. Définition

Définition

Une **boucle** permet de parcourir une partie d'un programme un certain nombre de fois. Une **itération** est la répétition d'un même traitement plusieurs fois.

Définition

Une boucle à **bornes définies** est une boucle pour laquelle le nombre d'itérations à effectuer, est connu grâce aux valeurs des bornes minimum et maximum.

Un indice de boucle varie alors de la valeur minimum (initiale) jusqu'à la valeur maximum (finale) .

Syntaxe

POUR variable **VARIANT DE** valeur initiale **A** valeur finale **FAIRE** <séquence d'instructions>

Exemple

POUR mois VARIANT DE Janvier A Décembre FAIRE <budget>

POUR Jour VARIANT DE Lundi A Vendredi FAIRE <Travail>

POUR i VARIANT DE 1 A nombre_etudiants FAIRE <corriger copies du ième étudiant>

2. Exemple d'itérations à bornes définies

Exemple

On désire généraliser l'algorithme de calcul du salaire net, vu précédemment, à la paie de N personnes .

1. Données N, PR, PL
2. Pour I variant de 1 à N, exécuter les instructions suivantes :
    a. Lire NH, SH

   b. SB <- SH x NH
   c. R <-SB x PR
   d. Si R > PL alors R <- PL
   e. SN <- SB - R
   f. Ecrire SN
3. Fin de l'itération

3. La boucle à bornes définies en Pascal : for

Remarque

La variable doit être de type scalaire (entier, énuméré, intervalle ou caractère). Elle ne peut pas être de type réel.

Remarque

Si exp1 > exp2 le for est ignoré

Exemple

```
program boucle_for;
var
     i:integer;
begin
     for i:=1 to 5 do
          writeln('le carré de ', i, ' est :', i*i);
     writeln;
     writeln('fin');
end.
```

Il est possible d'imbriquer plusieurs boucles FOR :

```
for x1 := c1 to c2 do
     begin
     ...
     for x2 := d1 to d2 do
          begin
          ...
         end;
         ...
end;
```

B. Boucles à bornes non définies

1. Boucles à bornes non définies : Boucle TANT QUE

Syntaxe

TANT QUE <condition> **FAIRE** <séquence d'instruction>

Exemple

```
TANT QUE le nombre considéré est positif
     FAIRE recherche de la racine carrée
```

Exemple

```
TANT QUE le feu est vert
     FAIRE action de passer
```

Exemple

```
TANT QUE il reste une fiche de paie non traîtée
     FAIRE traitement de la fiche concernée
```

2. Boucles à bornes non définies en Pascal : while ... do

Syntaxe

while expression **do** <bloc d'instructions>;

Remarque

Le bloc d'instructions n'est pas exécuté si la valeur de expression est false. Il n'est donc pas exécuté du tout si la valeur de l'expression est false au départ

Remarque

L'incrémentation doit être gérée par le programmeur lui-même. Il n'y a pas contrairement à la boucle for d'augmentation automatique d'une variable

Exemple

```
program boucle_while;
var
     i:integer;
begin
     i:=1;
     while i <= 5 do
     begin
          writeln('le carré de ', i, ' est :', sqr(i));
          i:=i+1; { incrémentation gérée par le programmeur
}
     end;
     writeln;
     writeln('FIN.');
end.
```

3. Boucles à bornes non définies : Boucle REPETER ... JUSQU'A

Syntaxe

REPETER <séquence d'instruction> **JUSQU'A** <condition>

Exemple

```
REPETER recherche de la racine carrée
     JUSQU'A  le nombre considéré est négatif
```

Exemple

```
REPETER action de passer
     JUSQU'A le feu n'est pas vert
```

Exemple

```
REPETER  traitement d'une fiche de paie
     JUSQU'A il ne reste plus de fiche de paie non traitée
```

4. Boucles à bornes non définies en Pascal : repeat ... until

Syntaxe

repeat <bloc d'instructions> **until** <expression>;

Remarque

La boucle s'effectue tant que la valeur de expression est *false*. On s'arrête quand l'expression devient *true*. C'est le contraire de la boucle *while*.

Remarque

Contrairement au *while*, il y a au moins un passage (1 boucle), même si l'expression est vraie au départ.

Remarque

De même que pour le *while*, c'est le programmeur qui gère l'incrémentation.

Exemple

```
program boucle_repeat;
     var i:integer;
begin
     repeat
          writeln('le carré de ', i, ' est :', sqr(i));
          i:=i+1;  { incrémentation gérée par le
programmeur }
     until i>5;
     writeln;
     writeln('FIN.');
end.
```

Attention

Il faut examiner en particulier :

- les conditions initiales,
- les conditions d'arrêt,
- l'incrémentation.

Avant de lancer le programme, il est conseillé de le faire "tourner" à la main (c'est-à-dire simuler l'exécution du programme pas à pas), en faisant évoluer les variables.

Les instructions contenues dans la boucle doivent permettre l'**évolution** de la valeur retournée par l'expression, sinon le programme peut rester bloqué dans une boucle infinie.

5. Boucles à bornes non définies : Comparaison de deux boucles

Les deux boucles peuvent être choisies indifféremment. Cependant, l'une est le contraire de l'autre, au niveau de la condition d'arrêt :

- **Tant que** condition1 est vraie, **faire** bloc d'instructions
- **Répéter** bloc d'instructions, **jusqu'à** ce que condition2 soit vraie Dans ce cas, condition1 est l'opposé de condition2

Exemple

les deux boucles suivantes sont équivalentes :

- tant que (i <> 10) faire i **<-** i+1 (on fait varier i jusqu'à 10)
- répéter i <- i+1 jusqu'à (i=10)

Il est toujours équivalent d'utiliser une boucle TANT QUE ou une boucle REPETER.

Cependant, il existe une différence entre les deux boucles :

Dans le cas d'une boucle REPETER ... JUSQU'A, le bloc d'instructions est effectué **au moins une fois**, ce qui n'est pas forcément vrai pour une boucle TANT QUE.

En effet, pour ce dernier type de boucle, si la condition est fausse dès le départ, le bloc d'instructions ne sera pas du tout exécuté. En revanche, avec une boucle REPETER ... JUSQU'A, si la condition est fausse dès le départ, le bloc d'instructions sera quand même exécuté une fois.

Remarque

Les boucles REPETER et TANT QUE peuvent être utilisées même si les bornes sont définies. Il est cependant préférable d'utiliser dans ce cas une boucle POUR.

Exemple

Reprenons l'exemple de la paie de N personnes

1. Données N, PR, PL
2. Initialiser I avec la valeur 1
3. Tant que I est inférieur ou égal à N, faire:
    a. Lire NH, SH
    b. SB <- SH x NH
    c. R <- SB x PR
    d. Si R > PL alors R <- PL
    e. SN <- SB - R
    f. Ecrire SN
    g. Donner à I la valeur suivante.
4. Fin de l'itération

Il en est de même avec Répéter...jusqu'à...

1. Données N, PR, PL
2. Initialiser I avec la valeur 1
3. Répéter les instructions suivantes :
    a. Lire NH, SH
    b. SB <- SH x NH
    c. R <- SB x PR
    d. Si R > PL alors R **<-** PL
    e. SN **<-** SB - R
    f. Ecrire SN
    g. Donner à I la valeur suivante.
4. Jusqu'à ce que I=N

Chapitre 6 - Tableaux

VI

Tableaux à une dimension	61
Tableaux à plusieurs dimensions	64

A. Tableaux à une dimension

1. Définition

Définition

Un **tableau** est une collection ordonnée d'éléments ayant tous le même type. On accède à chacun de ces éléments individuellement à l'aide d'un indice.

Un tableau à une dimension est parfois appelé vecteur .

Il peut être représenté sous la forme suivante :

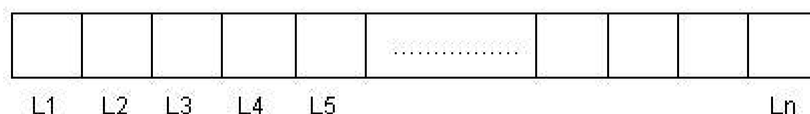


Image 23 : tableau

- Dimension du tableau : 1
- Taille du tableau : n
- Les Li (i = 1 ... n) doivent être de même type

2. déclaration d'un type tableau

Syntaxe

```
type identificateur = array[type-index] of type-éléments;
```

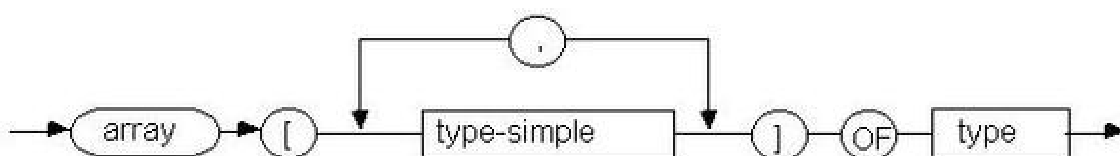


Image 24 : déclaration tableau

Remarque

- L'indice doit être **de type ordinal**, c'est à dire qu'il doit prendre ses valeurs dans un ensemble fini et ordonné (l'indice ne peut donc pas être de type réel).
- Les éléments doivent tous être de même type. Tout type est autorisé.
- Quand on ne connaîtconnait pas exactement le nombre d'éléments à l'avance, il faut majorer ce nombre, quitte à n'utiliser qu'une partie du tableau.
- Il est impossible de mettre une variable dans l'intervalle de définition du tableau. Il est conseillé en revanche d'utiliser des constantes définies au préalable dans la section **const**.

Exemple

```
Liste = array [1..100] of real;
Chaine = array [1..80] of char ;
Symptome = (FIEVRE, DELIRE, NAUSEE) ;
Malade = array [symptome] of boolean ;
Code = 0..99 ; Codage = array [1..N_VILLE] of code ;
```

3. Déclaration d'une variable de type tableau

Pour déclarer une variable de type tableau, il est préférable que le type correspondant ait été déclaré auparavant.

Exemple

```
const NMAX=100 ;
type Vecteur=array[1..NMAX] of real ;
var v : Vecteur;
```

Conseil

La déclaration suivante est autorisée, mais déconseillée :

```
var v : array[1..100] of integer
```

Remarque

L'indiçage peut être un type énuméré :

```
type Mois=(janvier, fevrier, mars, ..., decembre);
TnbJours=array[mois] of integer;
var T: TnbJours
```

On peut alors écrire :

```
T[mars]:=31;
T[fevrier]:=28; ...
```

4. Ecriture dans un tableau

On peut écrire de plusieurs façons dans un tableau :

- par affectation directe case par case : T[I] := ... ;

- par affectation globale : T := To;
- par lecture : read (T[I]);

5. Premier exemple d'écriture dans un tableau

Exemple

Le programme suivant permet de "remplir" un tableau à l'aide d'une boucle repeat ... until.

```
program Mon_tableau;
const
  Taille_max=10;
type
  TAB=array[1..Taille_max] of integer;
var
  Tableau:TAB;
  indice: integer;
begin
  for indice:=1 to Taille_max do
     Tableau[indice]:=0;
     indice:=1;
  repeat
         write('entrez l''élément N° ',indice,':');

         readln(Tableau[indice]);
         indice:=indice+1;
   until indice > Taille_max;
end.
```

6. Second exemple d'écriture dans un tableau

Exemple

Le programme suivant calcule le produit scalaire de deux vecteurs entrés par l'utilisateur.

```
program PRODUIT-SCALAIRE ;
type
  Coordonnee = (X1, X2, X3) ;
  Vecteur = array [Coordonnee] of real ;
var
  u, v : vecteur ;
  resultat : real ;
  c : Coordonnee ;
begin
  resultat := 0 ;
  for C := X1 to X3 do
  begin
    read (u[c]) ;
    readln (v[c]) ;
    resultat := resultat + u[c] * v[c] ;
  end ;
  writeln ('le produit scalaire est : ', resultat) ;
end.
```

B. Tableaux à plusieurs dimensions

1. Tableau à 2 dimensions

Un tableau de dimension 2 est parfois également appelé "MATRICE".
Un tel tableau peut être représenté sous la forme suivante :

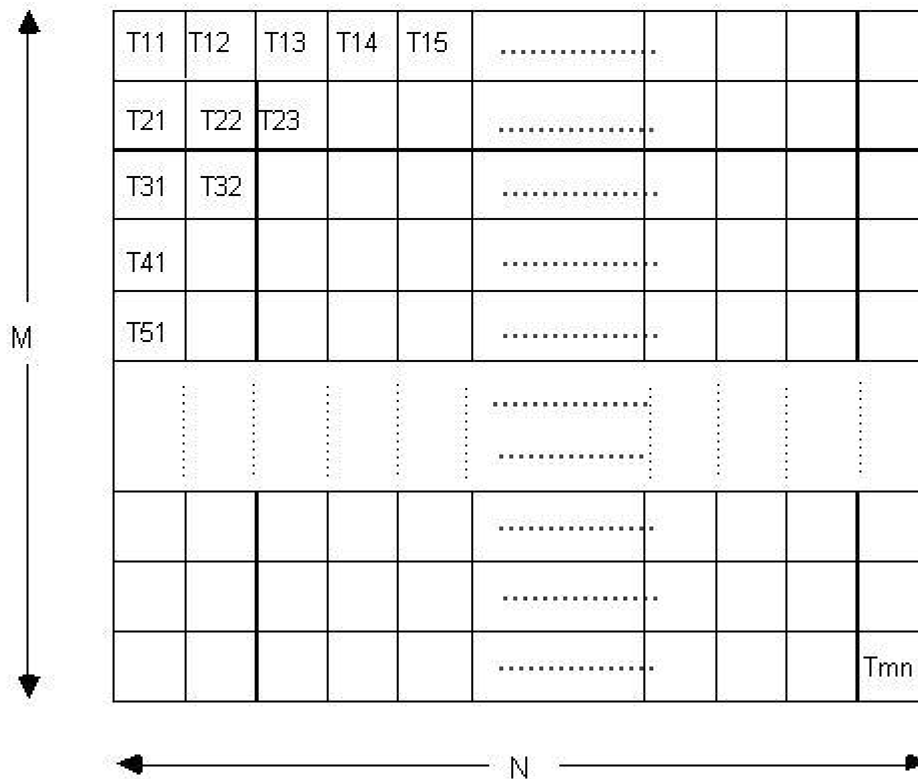


Image 25 : tableau à plusieurs dimensions

Syntaxe : Déclaration du type

```
type identificateur = array[1..M, 1..N] of type-éléments;
```

Méthode : Accès à un élément d'un tableau T

T[i][j] ou T[i,j]

2. Exemples

Syntaxe : Déclarations

```
type
  Tableau = array [1..10,1..20] of integer ;
  Point = array [1..50,1..50,1..50] of real ;
  Matrice = array [1..M,1..N] of real ;
  Carte = array [1..M] of array [1..N] of real ;
var
```

lieu : Carte;

Méthode : accès à un élément

lieu[i][j] ou lieu[i,j]

3. Exemple complet

Exemple

Initialisation d'une matrice unité de dimension 10. Il s'agit donc d'une matrice à 10 colonnes et 10 lignes, ne comportant que des 0, sauf sur sa diagonale où il n'y a que des 1.

```
program SOMME_MATRICE ;
const
  l_max = 10 ;
  c_max = 10 ;
type
    Matrice = array [1..l_max,1..c_max] of integer ;
var
   i, j : integer;
   mat : Matrice;
begin
    for i := 1 to l_max do
       begin
            for j := 1 to c_max do
                begin
                    if i = j then
                        mat [i, j] := 1
                    else
                        mat [i, j] := 0 ;
                   write (mat [i, j]);
                end ;
                writeln ;
       end ;
end.
```

Chapitre 7 - Chaînes de caractères

VII

Définition et méthodes 67
Exemples 70

A. Définition et méthodes

1. Définition

Définition

Une **chaîne de caractères** est une suite de caractères regroupés dans une même variable.

En Pascal, on utilise le type **string**, qui n'est pas un type standard. Il est construit à partir d'un tableau de 255 caractères maximum.

Ce type permet de manipuler des chaînes de **longueur variable**.

Il est possible de définir un sous-type, comportant moins de caractères :

Exemple : Déclarations

```
var
  s : string;
  s2 : string(20);
```

Pour accéder à un caractère particulier de la chaîne s, on écrit simplement, comme pour un tableau : s[i]

On peut aussi manipuler la chaîne de manière globale, ce qui est très utile pour des affectations ou des tests .

Exemple

```
s := 'Bonjour';
s[4] := 's';
s[6] := 'i';
  { à présent, s vaut 'Bonsoir' }
s := 'ok';
  { à présent, s vaut 'ok'}
```

Remarque

La taille de s est variable (dans l'exemple elle est de 7 caractères au départ et de 2 ensuite).

2. Opérateurs

a) Opérateurs de comparaison

Il est possible de comparer des chaînes de caractères avec les opérateurs : =, <, >, <=, >=, <>
L'ordre utilisé est l'ordre lexicographique (utilisation du code ASCII).

Exemple

```
s1:='salut';
s2:='bonjour';
s3:='bonsoir'
```

alors : s1 > s3 > s2

b) Concaténation

Définition

La concaténation est une opération qui permet d'accoler 2 ou plusieurs chaînes de caractères.

Syntaxe : Pascal

s:= s1 + s2 + s3... ;

Exemple

```
s1:='bon';
s2:='jour';
s3 := s1 + s2 ;
{ s3 vaut 'bonjour' }
```

3. Fonctions

a) Longueur d'une chaîne

En Pascal, la fonction length permet de connaître la longueur d'une chaîne.

Exemple

```
s1:='salut';
s2:='bonjour';
```

length(s1) vaut 5 et length(s2) vaut 7.

b) Position d'une sous-chaîne dans une chaîne

Fonction **pos** :

```
pos(souschaîne, chaîne)
```

renvoie la position de la sous chaîne dans la chaîne.

c) Extraction d'une sous-chaîne

Fonction **copy** :

```
copy (source, debut, l)
```

extrait la sous-chaîne de la chaîne source de longueur l et commençant à la position debut.

Exemple

```
s:=copy('bonjour monsieur', 4, 4)
```

s vaut alors 'jour'

4. Fonctions de codage / décodage des caractères

a) Détermination du code d'un caractère

La fonction **ord** renvoie le code ASCII d'un caractère donné.

Exemple

ord('A') vaut 65 et ord('a') vaut 97

b) Détermination du caractère correspondant à un code ASCII

La fonction **chr** renvoie le caractère correspondant à un code ASCII donné

Exemple

chr(65) vaut 'A' et chr(97) vaut 'a'

c) Exemples

Exemple

On désire transformer une lettre minuscule en lettre majuscule. Soit c le caractère à transformer. On écrira alors :

```
c := chr ( ORD(c) - ord('a') + ord('A') );
```

Explications :

Si c correspond à la lettre 'a', alors :

ord(c) - ord('a') = ord('a') - ord('a') = 0

donc

ord(c) - ord('a') + ord('A') = ord('A') = 65

et :

chr (ord('A')) = chr(65) = 'A'

Nous avons bien transformé 'a' en 'A'

B. Exemples

1. Exemple 1

Exemple

```
program ExChaines;
var
     s, s1, s2 : string;
begin
     write('entrez une chaîne caractères : ');
     readln(s1);
     write('entrez en une autre : ');
     readln(s2); s:= s1 +' ' + s2 ;
     write('La longueur de la 1ère chaîne est ');
     writeln(length(s1));
     write('La longueur de la 2ème chaîne est ');
     writeln(length(s2));
     writeln;
     writeln('La chaîne finale est : ', s );
     writeln('Sa longueur est : ', length(s):3 );
end.
```

2. Exemple 2

Exemple

```
program Test ;
var
     s1, s2 : string;
     i : integer;
     c : char;
begin
     write('entrez une chaîne caractères : ');
     readln(s1); s2 := s1;
     for i := 1 to length(s1) do
          if s[i] in ['a'..'z'] then
          begin c:=chr( ord(s1[i] - ord('a') + ord('A') );
               s2[i] := c;
          end;
     writeln;

writeln('------------------------------------------------')
;
     writeln;
     writeln('L''ancienne valeur était : ', s1);
     writeln('La nouvelle valeur est : ',s2);
end.
```

Chapitre 8 - Fonctions et Procédures

VIII

Procédures	71
Fonctions	72
Variables globales et variables locales	75
Paramètres	78

A. Procédures

1. Définition et déclaration

Définition

Une procédure permet de définir un traitement autonome, nommé par un identificateur et appelable par cet identificateur à partir du programme principal.

Il est en particulier utile de définir une procédure lorsqu'un même traitement doit être **effectué à plusieurs reprises** dans le programme.

L'utilisation de procédures permet de **structurer** un programme et d'augmenter sa lisibilité.

En Pascal les déclarations de procédures et de fonctions doivent être placées après les déclarations de variables. Les déclarations s'effectuent donc dans l'ordre suivant :

```
En-tête
    Déclarations
    Constantes
    Types Variables
    Fonctions / Procédures
Bloc d'instructions exécutables
```

Syntaxe : Déclaration d'une procédure

procedure <identificateur> <liste de paramètres> ;

2. Exemple

Exemple : Affichage d'un vecteur

```
procedure affichage ( v : Vecteur ; n : integer);
```

```
var
    i: integer; { i est une variable locale, Cf. 8.3 }
begin
    for i:=1 to n do write(v[i]);
    writeln;
end;
```

Cette procédure peut être utilisée pour une variable de type :

Vecteur = array[1..Nmax] of real ; Nmax étant une constante définie au préalable.

3. Appel d'une procédure

Une fois la procédure déclarée, elle peut être utilisée dans le programme principal par un "appel" à cette procédure, à partir du programme principal, ou à partir d'une autre procédure.

Syntaxe : Appel d'une procédure

<identificateur de la procédure> <liste de paramètres> ;

Exemple : Pour appeler la procédure d'affichage de l'exemple précédent

```
affichage(v1,3) ; { v1 étant une variable de type Vecteur ,
de dimension 3 }
```

B. Fonctions

1. Définition et déclaration

Définition

Une **fonction** est analogue à une procédure mais elle doit de plus retourner une valeur.

Le type de cette valeur doit être spécifié explicitement.

Syntaxe : Déclaration

On déclare les fonctions au même niveau que les procédures

(après **const**, **type**, **var**)

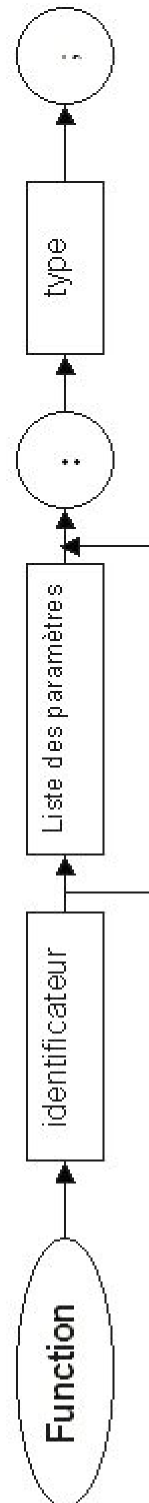


Image 26 : Diagramme de déclaration d'une fonction

2. Appel

On appelle une fonction au niveau d'une expression et non d'une instruction comme c'était le cas pour une procédure.

La fonction est appelée lors de l'évaluation de l'expression et c'est la valeur qu'elle

retourne qui est utilisée dans l'évaluation.

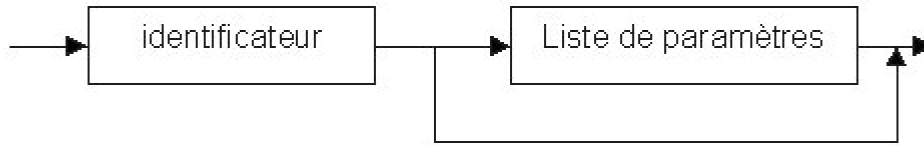


Image 27 : Diagramme d'appel d'une fonction

Ainsi, le simple fait d'écrire l'identificateur de fonction avec ses paramètres a pour conséquence d'appeler la fonction, d'effectuer le traitement contenu dans cette fonction, puis de remplacer l'identificateur par la valeur de la fonction dans le calcul.

3. Exemple

Exemple : Ecrire un programme qui calcule le cube d'un nombre réel

```
program Puissance3 ;
var
    unNombre : real;
function cube (x : real) : real;
begin
    cube:=x*x*x; { on affecte la valeur à
l'identificateur }
end; { Fin du code de la fonction }
begin { Début du programme principal }
    readln (unNombre);
    writeln ('Le cube de', unNombre,' est : ',
cube(unNombre) ;
end. { Fin du programme }
```

Cette procédure peut être utilisée pour une variable de type :

Vecteur = array[1..Nmax] of real ; Nmax étant une constante définie au préalable.

4. Différence entre procédure et fonction

- Une procédure peut être considérée comme une instruction composée que l'utilisateur aurait créée lui-même. On peut la considérer comme **un petit programme**.
- Une fonction quant à elle **renvoie toujours une "valeur"**. Elle nécessite donc un type (entier, caractère, booléen, réel, etc...).

Remarque

Il **est interdit** d'utiliser l'identificateur d'une fonction comme nom de variable en dehors du bloc correspondant à sa déclaration.

Exemple

```
program exemple;
var
    x,y : integer;
function double (z : integer) : integer;
```

```
begin
    double := z*2;
end;
begin
    readln(x);
    y := double(x);
    double := 8; { erreur à cette ligne lors de la
compilation }
end.
```

Ce programme ne pourra pas fonctionner, car on lui demande d'affecter la valeur 8 à la fonction double. Or, il **est interdit** d'utiliser l'identificateur d'une fonction comme nom de variable en dehors du bloc correspondant à sa déclaration.

C. Variables globales et variables locales

1. Définitions

Définition

Jusqu'à présent, nous avons effectué toutes les déclarations de variables en tête de programme. Or il est également possible de déclarer des variables, **au sein d'un bloc fonction ou procédure.** Dans ce cas, les déclarations se font dans le même ordre : constantes, types, variables (et même procédure(s) et/ou fonction(s)).

Lorsque la déclaration est effectuée en en-tête du programme, on parle de **variable globale**.

Dans le cas contraire, c'est-à-dire lorsque la déclaration est faite à l'intérieur même de la procédure ou de la fonction, on parle de **variable locale**.

2. Exemple

Exemple

Dans l'exemple suivant N et T sont des variables globales, Y et I sont locales à la fonction puissance.

```
program EXPOSANT ;
var
    N : integer;
    T : real;
    function puissance (X : real; N : integer) : real;
    var
        Y : real;
        I : integer;
    begin { Code de la fonction }
        Y := 1;
        if N > 0 then
            for I := 1 to N do Y := Y * X;
        else
            for I := -1 downto N do Y := Y / X;
        puissance:= Y;
    end; { Fin du code de la fonction }
```

```
begin
    readln (T, N);
    writeln (puissance (T, N)); {appel de la fonction }
end.
```

3. Portée des variables

Variable locale

la portée est limitée à la procédure ou à la fonction dans laquelle est est déclarée et à toutes les fonctions et procédures de niveau inférieur.

Variable globale

active dans le bloc principal du programme, mais également dans toutes les procédures et fonctions du programme.

Remarque

Les variables déclarées dans un bloc sont actives dans ce bloc et dans tous les blocs de niveau inférieur

4. Exemple

Exemple

```
procedure NIVEAU-SUP;
var
   X, Y, ...

    procedure NIVEAU-INF;
     var
        X,Z,...
     begin
        X:=...
        Y:=...
        Z:=...
     end;
begin
end.
```

Image 28 : Portée des variables entre blocs

La variable X déclarée dans NIVEAU_SUP est **locale** à cette procédure.

Dans la procédure NIVEAU-INF, cette variable est occultée par l'autre variable X, déclarée encore plus localement.

La portée de Y est celle de la procédure NIVEAU-SUP, sans restriction (elle est **locale** à NIVEAU-SUP, et donc **globale** dans NIVEAU-INF).

En revanche, Z a une portée limitée à la procédure NIVEAU-INF (elle est **locale** à NIVEAU-INF, et ne peut être utilisée dans NIVEAU-SUP).

5. Remarques

Remarque

Dans le cas où un même nom est utilisé pour une variable globale et pour une variable locale, cette dernière à la **priorité** dans la procédure ou dans la fonction où elle est déclarée (niveau local).

Dans le cas où un même nom est utilisé pour une variable locale et pour une variable globale, le programme considère ces variables comme **deux variables différentes** (mais ayant le même nom).

Exemple

```
program Portee;
var
    x : real;
procedure proc1;
var
    x : real;
begin
    x:=0;
    writeln (x);
  end;
begin { programme principal }
    x:=5;
    proc1;
    writeln(x)
end.
```

A l'exécution, le programme affiche 0, puis 5. En effet, l'appel de proc1 ne modifie pas la variable globale x , puisque x est redéclarée localement à la procédure proc1.

6. Local ou global ?

Il vaut toujours mieux privilégier les variables locales aux variables globales.

Inconvénients d'une utilisation systématique de variables globales :

- manque de lisibilité
- présence de trop nombreuses variables au même niveau
- récursivité plus difficile à mettre en oeuvre
- risque d'effets de bord si la procédure modifie les variables globales

Avantages d'une utilisation de variables locales :

- meilleure structuration
- diminution des erreurs de programmation
- les variables locales servent de variables intermédiaires (tampon) et sont "oubliées" (effacées de la mémoire) à la sortie de la procédure

Attention

Une procédure doit effectuer la tâche qui lui a été confiée, en ne modifiant que l'état de ses variables locales.

D. Paramètres

1. Exemple : 2 solutions

Énoncé

Supposons qu'on ait à résoudre des équations du second degré en divers points d'un programme :

- la première fois Rx2 + Sx + T= 0,
- la deuxième fois Mx2 + Nx + P = 0,
- la troisième fois Ux2 + Vx + W = 0.

Comment faire en sorte qu'une même procédure puisse les traiter toutes les trois, c'est-à-dire **travailler sur des données différentes** ?

1ère possibilité

utiliser des variables globales A, B, C pour exprimer les instructions dans la procédure, et, avant l'appel de la procédure, faire exécuter des instructions telles que :

A := R ; B := S ; C := T, etc.

Cette solution utilise des variables globales et multiplie les affectations. Il faut donc l'écarter !

2ème possibilité

définir une procédure de résolution d'une équation du second degré avec une liste de paramètres.

2. Choix de la 2ème solution

La déclaration sera :

```
procedure second_degre(A,B,C:integer);
```

Un paramètre est spécifié par un identificateur et par une déclaration de type. On peut grouper plusieurs paramètres de même type en les séparant par des virgules

A l'appel, on écrira

```
second_degre (M, N, P); { appel avec les valeurs de M, N et
P }
second_degre (R, S, T); { appel avec les valeurs de R, S et
T }
```

Lors de l'appel de la procédure, il y a remplacement de chaque paramètre formel par un paramètre effectif, bien spécifié.

Ainsi, au premier appel, A prendra la valeur de M, B celle de N et C celle de P. Au second appel : A prendra la valeur de R, B celle de S et C celle de T.

Remarque

Attention à la compatibilité des types !

Cette transmission d'information est équivalente à une **affectation de valeurs** dans des sortes de variables locales, qui sont en fait représentées par les **paramètres** (ou arguments) de la procédure.

3. Passage de paramètre par valeur

Méthode

Passer à la procédure des valeurs qui seront les données d'entrée et de travail pour la procédure.

Exemple : Déclaration

```
procedure ID_PROC (X, Y : real; I : integer; TEST:
boolean);
```

Points importants

- Les paramètres formels sont des **variables locales** à la procédure, qui reçoivent comme valeurs initiales celles passées lors de l'appel

  Exemple : ID_PROC (A, B, 5, true);

  X a alors pour valeur initiale celle de A, Y celle de B, I a pour valeur initiale 5, et TEST true
- Le traitement effectué dans la procédure, quel qu'il soit, ne pourra modifier la valeur des paramètres effectifs.

  Par exemple : après exécution de ID_PROC, A et B auront **toujours la même valeur qu'auparavant**, même s'il y a eu des changements pour ces variables, dans la procédure. **La transmission est unilatérale**.
- Le paramètre spécifié lors de l'appel peut être une expression.

  Ainsi, ID_PROC (3 / 5, 7 div E, trunc (P), true); est un appel correct.

Avantages

- les paramètres effectifs peuvent être des expressions.
- les erreurs et les effets de bord sont évités (Cf exemple 2).
- les paramètres sont utilisés pour passer des valeurs à la procédure, mais ne sont jamais modifiés.

A noter

Le résultat de l'action accomplie par la procédure n'est pas transmis au programme appelant (Cf. exemple 1).

Si on désire récupérer, le paramètre modifié, il faut alors utiliser un autre procédé, décrit dans la section suivante (passage **par adresse**).

4. Exemples de passage par valeur

Exemple : Nous cherchons à écrire un programme qui échange les valeurs de deux variables saisies par l'utilisateur.

```
program TEST;
var
    A, B : real;
procedure ECHANGE (X, Y : REAL);
var
    T : real;
begin
    T := X;
    X := Y;
```

```
    Y := T;
    Writeln(X,Y);
end;
begin
    readln (A, B);
    ECHANGE (A, B);
    writeln (A, B);
end.
```

Cette solution est mauvaise. En effet, une simulation de son exécution donnera :

A = 5 B = 7 { saisie }

X = 7 Y = 5

A = 5 B = 7 {A et B restent inchangés !}

Exemple : Ce programme a pour seul intérêt d'illustrer le passage de paramètres ainsi que la notion d'effet de bord.

```
program Effet-de_bord;
var
    i,j : integer;
function double (i : integer) : integer;
begin
    double := 2 * i;
end;
function plus_un (j : integer) : integer;
var
    i: integer;
begin
    i := 10;
    plus_un := j + 1;
end;
function moins_un (j : integer) : integer;
begin
    i := 10;
    moins_un := j - 1;
end;
begin { programme principal }
    i := 0; j := 0; {i=0 ; j=0}
    j := plus_un(i); {i=0 ; j=1}
    j := double(j);{i=0 ; j=2}
    j := moins_un(j);{i=10 ; j=1}
end.
```

La variable i a été modifiée dans la procédure, alors que l'on s'attendait à des modifications sur j. On appelle cela un **effet de bord**.

Un tel phénomène peut être très **gênant** (difficulté à retrouver les erreurs).

5. Passage de paramètre par adresse

Méthode

On fournit en paramètre une variable (ou plutôt son adresse) et on travaille directement sur celle-ci, et non sur la valeur contenue dans cette variable.

Pour réaliser un passage de paramètre par adresse, il faut lors de la déclaration de la procédure (ou de la fonction) ajouter le **mot clé var** devant la déclaration du paramètre concerné. Il est ainsi possible de **récupérer les modifications effectuées** sur cette variable, à la fin de l'exécution de la procédure.

Exemple : Déclaration

```
procedure ID_PROC (var X, Y : real; Z : integer);
```

Points importants

- Lors de l'appel, des paramètres réels sont substitués aux paramètres formels.
- Tout changement sur le paramètre formel variable change aussi le paramètre effectif spécifié lors de l'appel.
- Seule une variable peut être substituée aux paramètres réels, il est impossible de faire l'appel avec une constante ou une expression évaluable.

  Exemple :

  ID_PROC (U, V, 7); { correct }

  ID_PROC (4, A - B, 8); {tout à fait incorrect }

A noter

Lorsqu'il y a nécessité de renvoyer une modification au programme appelant, on emploie un passage par adresse.

6. Exemple de passage par adresse

Exemple

```
program essai;
var
    i : integer;
procedure double (x : integer ; var res : integer);
begin
    res := 2 * x;
end;
begin
    i := 1; { i=1 }
    double (5,i); {i=10}
end;
```

Dans cet exemple, x est un paramètre transmis par valeur, alors que res est un paramètre passé par adresse.

Exemple : reprenons et corrigeons le programme qui échange les valeurs de deux variables saisies par l'utilisateur.

```
program TEST_BIS;
var
    A, B : real;
procedure ECHANGE (var X, Y : real);
var
    T : real;
begin
    T := X;
    X := Y;
    Y := T;
    writeln(X,Y);
end;
```

```
begin
    readln (A, B);
    ECHANGE (A, B);
    writeln (A, B);
end.
```

Une simulation du déroulement du programme donnera :

A = 5 B = 7 (saisie)

X = 7 Y = 5

A = 7 B = 5 (A et B ont été **modifiés** !)

Le résultat de l'action de la procédure a été transmis au programme appelant.

7. Cas des fonctions

Pour les fonctions, on peut agir de même. Le principe est strictement analogue à celui des procédures. On distingue donc, là encore, les passages de paramètres par valeur des passages de paramètres par adresse (ou variables).

Exemple

```
function LETTRE (c : char) : boolean;
begin
    if (c in ['A'..'Z']) or (c in ['a'..'z']) then
        lettre := true
    else
        lettre := false;
    end;
function MAJ (var c : char) : boolean;
begin
    if not LETTRE(c) then
        maj : false
    else
    begin
        if c in ['a'..'z'] then
        begin
            c := chr (ord(c) - ord('a') + ord('A'));
            maj := true;
        end
        else
            maj := false;
    end;
end;
```

Ce programme résume ce que nous avons pu voir avec les procédures.

La première fonction utilise un passage de paramètre par valeur car nous n'avons pas besoin de modifier ce paramètre ; cette fonction est utilisée pour tester si un caractère est une lettre (minuscule ou majuscule).

La fonction MAJ (qui utilise la fonction précédente) modifie un caractère qui est une lettre minuscule en sa majuscule correspondante.

8. Paramètres fonctions

Il est possible d'utiliser une fonction comme paramètre d'une procédure ou d'une fonction. Cependant, ceci reste relativement rare. Ce procédé est assez peu utilisé, et ne fait pas partie du programme de ce cours.

Exemple : exemple à titre indicatif uniquement

```
program DEMONSTRATION;
var
    TAN, COT, LOG_A : real;
function QUOTIENT (function NUM, DEN, X : real) : real;
begin
    QUOTIENT := NUM (x) / DEN (x); {utilisation des
fonctions NUM et DEN}
end;
begin
    readln (x);
    TAN := QUOTIENT (sin, cos, x);
    writeln (TAN);
    COT := QUOTIENT (cos, sin, x);
    writeln (COT);
end.
```

On peut faire de même avec les paramètres procédures.

Chapitre 9 - Ensembles

IX

Définition et exemples	85
Opérations sur les ensembles	86

En Pascal, il est possible de définir des ensembles finis et d'effectuer des operations sur ces ensembles.

A. Définition et exemples

1. Définition et syntaxe

Définition

Soit un ensemble de base E, de type **énuméré** ou **intervalle**.

Le type ensemble associé à E est l'ensemble des **sous-ensembles** (ou parties) de E.

Syntaxe

```
type
  identificateur = set of type_simple;
```

2. Exemple 1

Exemple

```
type
  Palette = (bleu, blanc, rouge);
  Couleur = set of Palette ;
var
  teinte : Couleur ;
...
```

Le type ensemble associé à Palette contient tous les sous-ensembles de Palette :

{ } {bleu} {blanc} {rouge} {bleu, blanc} {bleu, rouge} {blanc, rouge} {bleu, blanc, rouge}

soit 2^3=8 valeurs.

Remarque

De manière plus générale, si card E = n, le nombre de valeurs possibles est : 2^n

Exemple

Avec les déclarations de l'exemple 1, on peut écrire des instructions telles que :

```
  teinte := [ ] ;              { ensemble vide }
  teinte := [blanc,rouge] ;    { 2 éléments : blanc et
rouge }
  teinte := [bleu..rouge];     { toutes les valeurs de bleu à
rouge }
```

3. Exemple 2

Exemple

```
type
  Decimal = 0..9 ;
  EnsChiffres = set of Decimal ;
var
  tirage : EnsChiffres ;
```

Dans le corps du programme, on pourra écrire :

```
tirage := [ ];
tirage := [3,5,8] ;
tirage := [4..8] ;
tirage := [1..4,7..9] ;
```

B. Opérations sur les ensembles

1. Union ---> opérateur +

Exemple

```
var
  A,B,C : set of 1..10;
begin
  A := [2, 5, 9] ;
  B := [3, 5, 7] ;
  C := A + B ;
end.
```

Exécution : C = [2, 3, 5, 7, 9]

2. Intersection ---> opérateur *

Exemple

```
var
  A,B,C : set of 1..10;
begin
  A := [2, 5, 9] ;
  B := [3, 5, 7] ;
  C := A * B ;
end.
```

Exécution : C = [5]

3. Différence ---> opérateur -

Exemple

```
var
  A,B,C : set of 1..10;
begin
  A := [2, 5, 9] ;
  B := [3, 5, 7] ;
  C := A - B ;
end.
```

Exécution : C = [2, 9]

4. Egalité ---> opérateur =

Exemple

```
type
  Note = (do, ré, mi, fa, sol, la, si) ;
var
  accord : set of Note;
```

Syntaxe

On pourra écrire :

```
if accord = [do, mi, sol] then ...
```

5. Inégalité ---> opérateur <>

Exemple

```
type
  Note = (do, ré, mi, fa, sol, la, si) ;
var
  accord : set of Note;
```

Syntaxe

On pourra écrire :

```
if accord <> [do, mi, sol] then ...
```

6. Inclusion ---> opérateurs <= ou >=

Exemple

```
type
  Note = (do, ré, mi, fa, sol, la, si) ;
var
  accord : set of Note;
```

Syntaxe

L'expression

```
[mi, sol] <= [ré . . sol]
```

sera évaluée à *«true»*

Syntaxe

De même :

```
[4,7,9] >= [7, 9]
```

7. Appartenance ---> opérateur in

Exemple

```
if note in accord
if note in [do, mi , sol]
```

On utilise souvent cette opération sur des ensembles de caractères, pour tester si un caractère est une lettre minuscule ou majuscule.

Exemple

```
function lettre (c : char) : boolean;
begin
  if (c in ['A'..'Z']) or (c in ['a'..'z']) then
    lettre := true
  else
    lettre := false;
end;
function maj (var c : char) : boolean;
begin
  if not LETTRE(c) then
    maj : =false
  else
    begin
      if c in ['a'..'z'] then
        c := chr (ord(c) - ord('a') + ord('A'));
      maj := true;
```

```
    end;
end;
```

Chapitre 10 - Enregistrements

Généralités	91
Ecriture dans un enregistrement	92
Instruction with	94
Tableaux d'enregistrements	94

A. Généralités

1. Définition

Définition

Une variable de type **enregistrement** est une variable structurée avec plusieurs champs.

Les **champs** sont les attributs ou caractéristiques de l'enregistrement.

Ils sont parfois appelés rubriques ou propriétés.

Alors que les éléments d'un tableau sont nécessairement de même type, les champs d'un enregistrement peuvent être de **types différents**.

2. Déclaration

Syntaxe

```
type
    dentificateur = record <liste de champs> ;
end;
```

Syntaxe : Spécification d'un champ dans la déclaration

```
identificateur : type_champ ;
identificateur1, identificateur2, ............. : type_champ ;
```

Exemple

```
type
      personne = record
                       nom:string[40];
                       prenom:string[50];
                       age:integer;
                    end;
      voiture = record
                    marque : string ;
                    cylindree : real ;
                    couleur : string;
                    nom : string ;
                    prix : integer ;
                  end ;
      une_couleur = (trefle, carreau, coeur, pique);
      une_valeur = (sept, huit, ..., dame, roi, as);
      carte = record
                 couleur : une_couleur;
                 valeur : une_valeur;
               end;
```

3. Accès aux champs

Complément

Il faut ensuite déclarer les variables associées.
Soit v une variable de type enregistrement.
Pour **accéder** à un champ de cet enregistrement, il suffit simplement d'écrire :
v.identificateur_du_champ

Exemple

```
soient les déclarations suivantes :
type
      voiture = record
                    marque : string ;
                    cylindree : real ;
                    couleur : string;
                    nom : string ;
                    prix : integer ;
                  end ;
var auto : voiture;
```

Méthode

```
Pour afficher la marque et le prix de la variable auto, on
pourra écrire :
writeln(auto.marque, auto.prix...);
```

B. Ecriture dans un enregistrement

1. Méthodes d'écriture

Syntaxe : Affectation globale (comme pour les tableaux)

```
Si les deux enregistrements sont exactement de même type,
on peut faire une affectation globale, comme pour un
tableau : enreg1:= enreg2;
```

Syntaxe : Affectation directe sur un champ

```
enreg.champ:= valeur;
```

Syntaxe : Par une instruction de lecture

read(enreg.champ);

2. Exemples

Exemple : Création de la "307 HDI"

```
Il suffit d'écrire les instructions suivantes :
    auto.marque := 'Peugeot';
    auto.cylindree := 2.0;
    auto.couleur := 'gris';
    auto.nom := '307 HDI';
    auto.prix := 18000;
```

Exemple : Création d'un enregistrement UTC

```
type
    Adr = record
             numero : integer;
             rue : string;
             CP, ville : string;
             pays : string;
          end;
    Bat = record
             nom : string;
             adresse : Adr;
             departement : array[1..6] of string;
          end;
    Universite = record
              nom : string;
              tel : string;
               batiment : Bat;
          end;
 var
       fac : Universite ;
```

```
begin
   ...
      fac.tel := '0344234423';
      fac.nom := 'Universite de Technologie de Compiègne' ;
      fac.batiment.nom := 'Franklin' ;
      fac.batiment.adresse.rue := 'Personne de Roberval' ;
      fac.batiment.departement[1] := 'GI' ;
      fac.batiment.departement[2] := 'GM' ;
   ...
```

C. Instruction with

1. Avantage de l'instruction

L'écriture devient lourde et fastidieuse à cause de la répétition de l'identificateur de l'enregistrement.

L'instuction **with** permet d'alléger l'écriture en « factorisant » l'identificateur

Syntaxe

```
with <enregistrement> do
begin
        <Bloc d'instructions>
end;
```

2. Exemple

Exemple

```
with auto do
begin
        marque := 'Peugeot' ;
        cylindree := 2.0 ;
        couleur := 'gris' ;
        nom := '307 HDI' ;
        prix := 18000 ;
end ;
```

Remarque

Cela évite d'avoir à écrire plusieurs fois le préfixe auto : (auto.marque, auto.couleur, auto.nom, auto.prix...).

D. Tableaux d'enregistrements

1. Utilité des tableaux

Rappel

Nous avons vu précédemment (exemple 2, sur l'université) qu'il est possible d'utiliser des tableaux dans un enregistrement.

Complément

Inversement, il est souvent utile d'intégrer des enregistrements dans des tableaux. On parle alors de **tableaux d'enregistrements**.

Ce type de structure est particulièrement bien adapté pour représenter des groupes de personnes, par exemple. Nous illustrons cette notion avec l'exemple d'un groupe d'étudiants

Exemple

```
const
Max = 160;
type
      Etudiant = record
                          nom, prenom : string;
                          sexe : (M,F);
                          numInsee : string;
                          age : integer;
                        end;
      UV : array[1..Max] of ETUDIANT;
var
      NF01 : UV;
```

La variabl e NF01 est de type UV. Sa valeur est un tableau d'éléments de type ETUDIANT.

Méthode

```
On peut accéder à un étudiant particulier, par son indice
dans le tableau NF01.
Ainsi,
  NF01[1] correspondra au premier élément du tableau,
  NF01[12] au 12ème étudiant contenu dans ce tableau...
On pourra alors écrire :
  NF01[1].nom:='Machin';
  NF01[1].age:=19;
  NF01[2].nom:='Martin';
```

2. Exemples

Exemple : Saisie de tous les étudiants

```
for i:=1 to Max do
with NF01[i] do
  begin
    writeln;
    writeln('Saisie de l''étudiant n° ', i:3);
    readln(nom);
    readln(prenom);
    readln(age);
    readln(N°INSEE);
  end;
```

Exemple : Affichage de tous les étudiants

```
for i:=1 to Max do
with NF01[i] do
  begin
    writeln;
    writeln('Etudiant n° ', i:3);
    writeln('============');
    writeln(nom);
    writeln(prenom);
    writeln(age);
    writeln(numInsee);
  end;
```

Chapitre 11 - Fichiers

XI

Introduction	97
Organisation et accès	98
Les fichiers séquentiels en Pascal	100
Les fichiers structurés en Pascal	105
Les fichiers de texte	107

A. Introduction

1. Définition

Définition

Un **fichier** est une collection d'informations stockée sur un support physique (disque, bande, CD-Rom...).

Remarque

Un fichier permet de conserver durablement l'information (données programmes, résultats). L'information persiste à l'arrêt du programme.

2. Manipulation des fichiers

Le système d'exploitation fournit des **primitives de programmation** pour :

- La création de fichier
- L'ouverture d'un fichier
- La lecture dans un fichier
- L'écriture dans un fichier
- La fermeture d'un fichier
- La destruction d'un fichier

Il fournit une **bibliothèque de programmes utilitaires** pour :

- Copier des fichiers
- Renommer des fichiers
- Lister des fichiers
- Détruire des fichiers

3. Les supports physiques

Supports séquentiels (bandes et cassettes)

- Toutes les informations sont stockées de façon séquentielle, les unes à la suite des autres.
- Pour accéder à une information particulière, il faut nécessairement faire défiler le support à partir du début, et ce jusqu'au moment où cette information est retrouvée.
- La capacité de ces supports peut être importante, mais l'accès est lent.

Supports aléatoires (disques, CD-Rom...)

L'accès à une information particulière est possible sans qu'il soit indispensable de faire défiler le support à partir du début. On accède directement à une donnée à partir de son adresse (on parle alors de **support adressable**).

C'est le temps d'accès qui est aléatoire, et non le mode d'accès !

B. Organisation et accès

1. Définition

Définition

L'**organisation d'un fichier** est la manière dont sont rangés les enregistrements du fichier sur le support physique.

Remarque

L'organisation est choisie à la création du fichier. Le choix d'une organisation correspond à un compromis entre rapidité d'accès et espace de stockage disponible.

2. Organisation séquentielle

Définition

Elle ne permet qu'un seul accès : le séquentiel.

Toutes les informations sont enregistrées de façon **séquentielle** (linéaire) les unes à la suite des autres.

Fondamental : Lecture

Pour accéder à une information particulière, il faut nécessairement parcourir le fichier à partir du début, et ce jusqu'au moment où cette information est retrouvée.

Pour lire le ième enregistrement du fichier, il faudra donc d'abord lire les i-1 enregistrements qui le précèdent.

Fondamental : Ecriture

Pour ajouter une information, il faut se placer en fin de fichier

A la fin du fichier, on trouve un enregistrement spécial FIN_DE_FICHIER

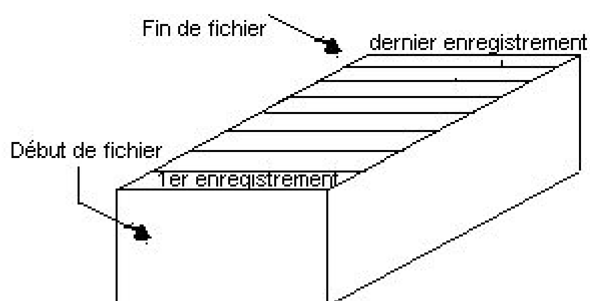


Image 29 : Organisation séquentielle

3. Organisation relative (accès direct)

Définition

Chaque enregistrement possède un **numéro**. On accède à la fiche recherchée en spécifiant ce numéro d'enregistrement. L'indication d'un numéro permet donc un **accès direct** à l'information ainsi référencée.

Fondamental

Cette organisation exige un support aléatoire (adressable), mais l'accès à l'information est beaucoup plus rapide.

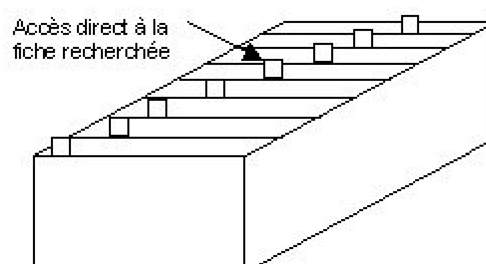


Image 30 : Organisation relative

4. Organisation indexée

Définition

On crée des fichiers supplémentaires d'**index** (voir figure page suivante).

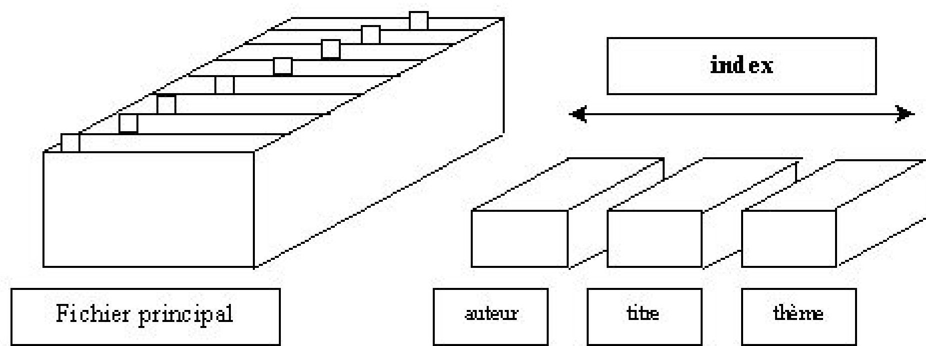


Image 31 : Organisation indexée

Fondamental

On parcourt un index pour rechercher une clef. On obtient ainsi l'adresse exacte de l'information recherchée. On peut utiliser des opérateurs de comparaisons, sur les index (=, <>, <, <=, >, >=).

Exemple

Il est alors possible, par exemple, de retrouver rapidement toutes les personnes de nom 'Dupont' ou de prénom 'André', ou toutes celles de plus de 18 ans....

Exemple

Dans l'exemple schématisé ci-dessus, on pourrait, grâce aux index, retrouver rapidement des livres à partir de leur auteur, de leur titre ou de leur thème.

C. Les fichiers séquentiels en Pascal

1. Définitions

Définition : Fichier

Un **fichier** Pascal est une séquence de données de même type et de longueur indéfinie.

Ces données sont stockées de manière permanente sur un support physique.

Définition : Fin de fichier

La **fin du fichier** est repérée par un marqueur de fin de fichier.

Pour tester la fin de fichier, on emploie la fonction booléenne eof (end_of_file) :

- eof (fichier) = true si la fin de fichier est atteinte
- eof (fichier) = false sinon.

Définition : Longueur

La **longueur du fichier** est le nombre d'enregistrements du fichier. Elle n'est pas définie lors de la déclaration du fichier.

Définition : Fichier vide

Un **fichier vide** est un fichier qui n'a aucun enregistrement.

2. Déclaration d'un fichier

Syntaxe : Déclaration

```
type
  IDENTIFICATEUR = file of ID_TYPE_ELEMENTS;
```

Tous les types sont autorisés pour les éléments (sauf le type fichier !).

Exemple

```
type
  Codes = file of integer ;
  Texte = file of char ;
  Adresse = record
    ......
    end;
  Repertoire = file of Adresse ;
  Coordonnees = record
    abscisse, ordonnee : real ;
    end ;
  TRACE = file of coordonnees ;
var
  fichier_client : Repertoire ;
  courbe : Trace ;
```

3. Création d'un fichier

Syntaxe : Instruction rewrite

```
rewrite (ID_NOM_DE_FICHIER);
```

Cette instruction permet d'ouvrir un fichier 'en écriture',c'est-à-dire de **créer** le fichier, et d'autoriser des opérations d'**écriture** dans ce dernier.

Remarque

Lors de l'execution de l'instruction : rewrite(f) :

- Si le fichier correspondant à f n'existe pas il est créé
- S'il existe déjà toutes ses données sont effacées
- eof(f) devient vrai
- Le pointeur de fichier (ou fenêtre) est positionné au début du fichier vide correspondant à f

Définition : Pointeur de fichier

Le **pointeur de fichier** est un repère (un marqueur) servant à indiquer au système d'exploitation l'adresse à laquelle il faudra lire ou écrire le prochain enregistrement.

4. Ecriture dans un fichier

Fondamental

L'écriture dans un fichier est presque analogue à l'affichage d'une donnée à l'écran. Il suffit simplement d'ajouter en premier paramètre le nom logique du ficher.

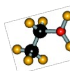
Syntaxe : Instruction write

```
write (f,x) ;
```

Ecriture de l'enregistrement x dans le fichier logique f.

Remarque

Le type de x doit être compatible avec le type des enregistrements du fichier.



Simulateur

Avant l'exécution de la commande :

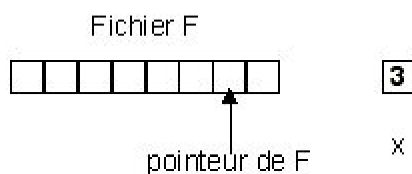


Image 32 : Avant exécution

Après l'exécution de la commande :

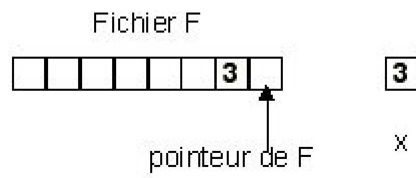


Image 33 : Après exécution

La commande écrit la valeur de x (ici la valeur 3) à l'endroit où pointe le fichier et déplace ce pointeur d'une position vers la droite.

5. Ouverture d'un fichier existant

Fondamental

Dans un fichier séquentiel, pour pouvoir lire une donnée, il est impératif que le fichier ait d'abord été ouvert **'en lecture'**.

On utilise pour cela l'instruction **reset**.

Syntaxe : Instruction reset

```
reset (id_fichier);
```

Remarque

Effet de l'instruction reset :

- Le pointeur de fichier est positionné sur le premier enregistrement
- Si le fichier contient au moins un article, eof(id_fichier) devient faux

Remarque

Il n'est pas possible de lire des informations dans un fichier ouvert avec rewrite (ouverture **'en écriture'**). En effet, nous avons vu précédemment que lorsqu'on ouvre un fichier existant avec rewrite les anciennes données sont perdues (le fichier est « écrasé »).

6. Lecture dans un fichier

Fondamental

Comme pour l'écriture, la lecture dans un fichier est quasiment analogue à la lecture d'une donnée au clavier. Il suffit là encore d'ajouter en premier paramètre le nom interne du ficher.

Syntaxe : Instruction read

```
read(f,x);
```

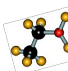
Remarque

Cette instruction permet de lire l'enregistrement du fichier repéré par le pointeur de fichier, puis de placer cet enregistrement dans la variable x passée en paramètre.

De même que pour l'instruction write, le type de x doit être le même que celui spécifié lors de la déclaration du fichier.

Attention

Cette instruction ne peut être exécutée que si la commande reset(f) a été utilisée auparavant.



Simulateur

Avant l'exécution de la commande :

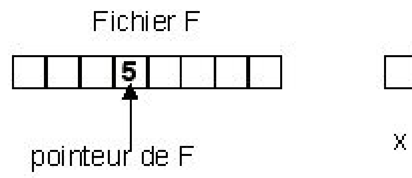


Image 34 : Avant exécution

Après l'exécution de la commande :

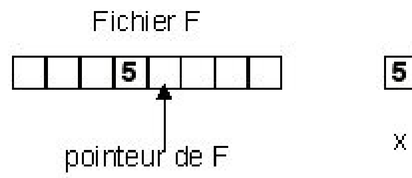


Image 35 : Après exécution

La commande lit la valeur pointée par le fichier et l'assigne à la variable précisée dans son deuxième argument. x prend alors la valeur 5.

Le pointeur est ensuite déplacé d'une position vers la droite.

Attention

Si la fin du fichier est atteinte, eof(f) devient vrai.

Dans ce cas, si on essaie de lire le fichier, le programme génère une erreur.

Remarque

A chaque exécution de l'instruction **reset**, le pointeur de fichier est repositionné au début et eof devient faux.

7. Association entre un nom interne et un nom externe

Définition : Nom interne

Le **nom interne** (ou **nom logique**) correspond au nom utilisé dans le programme.

C'est donc l'identificateur déclaré comme variable (ex : var f : file of ...).

Il s'agit donc du nom de fichier vu par le programmeur et par le programme Pascal.

Définition : Nom externe

Le **nom externe** (ou **nom physique**) représente quant à lui le nom utilisé sur le disque, visible dans le répertoire (dossier ou directory).

Il s'agit donc du nom de fichier vu par l'utilisateur et par le système d'exploitation.

Remarque : Association des deux noms

Certains compilateurs permettent l'association des deux noms au niveau des instructions rewrite ou reset :

```
rewrite(ID_FICHIER_INTERNE, ID_FICHIER_EXTERNE);
reset(ID_FICHIER_INTERNE, ID_FICHIER_EXTERNE);
```

Exemple

```
rewrite (fOut, 'FichierSortie.txt') ;
reset (fIn, 'FichierEntree.txt') ;
```

Remarque : Instruction assign

D'autres compilateurs utilisent l'instruction assign :

```
assign(ID_FICHIER_INTERNE, ID_FICHIER_EXTERNE);
```

Exemple

```
assign(f, 'mon_fichier') ;
rewrite (f); {ou reset(f) pour une ouverture en lecture }
```

Remarque : Nom du fichier externe

Il est préférable d'utiliser une constante ou une variable pour le nom du fichier externe.

Exemple

```
write('nom du fichier externe ? ');
readln(nom_externe);
assign(f, nom_externe) ;
rewrite(f) ;
...
```

8. Fermeture d'un fichier

Fondamental

Chaque fichier ouvert en lecture (reset) ou en écriture (rewrite) doit être **fermé** à l'aide de l'instruction close.

Syntaxe : Instruction close

```
close(fichier) ;
```

Remarque

L'exécution de cette instruction garantit l'écriture de tous les enregistrements (pour un fichier ouvert en écriture) et met fin à l'association nom interne-nom externe.

D. Les fichiers structurés en Pascal

1. Définition

Définition : Fichier structuré

Un **fichier structuré** est un fichier dont les enregistrements contiennent chacun plusieurs champs (ou rubriques).

En Pascal, un enregistrement d'un fichier structuré est de type **record**.

2. Lecture d'un fichier

Le programme suivant ouvre un fichier et affiche les données qu'il contient.

Exemple : Programme LectureAdresses

```
program LectureAdresses;
type
  Adresse = record
      nom, rue, ville : string;
      numero : integer;
    end;
  Fichier = file of Adresse ;
var
  carnet : Fichier;
  client : Adresse;
```

```
begin
  assign(carnet, 'carnet_adresses') ;
  reset(carnet) ;
  with client do
    while not eof (carnet) do
    begin
      read(carnet, client);
      write ('NOM :', nom) ;
      write ('NUMERO, numero) ;
      write ('RUE :', rue) ;
      write ('VILLE :', ville) ;
    end ;
  close (carnet) ;
end.
```

3. Ecriture dans un fichier

Exemple : Programme Adresses

```
program Adresses;
  Adresse = record
      nom, rue, ville : string;
      numero : integer;
    end;
  Fichier = file of Adresse ;
var
  carnet : Fichier;
  client : Adresse;
  c : char;
begin
  assign (carnet, 'carnet_adresses') ;
  rewrite (carnet) ;
  c := 'O' ;
  with client do
    while c <> 'N' do
    begin
      write ('NOM :') ; readln (nom) ;
      write ('NUMERO :') ; readln (numero) ;
      write ('RUE :') ; readln (rue) ;
      write ('VILLE :') ; readln (ville) ;
      write(carnet,client) ;
      writeln ('Autre adresse ?') ;
      readln (c) ;
    end ;
  close (carnet) ;
end.
```

E. Les fichiers de texte

1. Définition

Définition : Fichier de texte

Les **fichiers de texte** ont un statut particulier. Ils sont constitués de caractères, mais peuvent être vus également comme une suite de lignes de texte de longueur variable.

Conseil

En Pascal, il est possible pour ces fichiers d'utiliser les instructions **readln** et **writeln** au lieu de read et write.

Remarque

La fin d'une ligne est marquée dans le fichier par un ou deux caractères spéciaux non imprimables (codes 0D 0A sous Windows et 0A sous Linux).

2. Déclaration

Le type **Text** est un type prédéfini en Pascal pour les fichiers texte.

Il suffit donc de déclarer une variable de type Text :

Syntaxe : Déclaration

```
var
  fich : Text ;
```

Remarque

En fonction des compilateurs, on doit parfois utiliser à la place de Text l'un des types suivants :

- File of char;
- File of string;
- File of Text;
- TextFile;

3. Ecriture dans un fichier texte

Dans l'exemple suivant, on souhaite que l'utilisateur puisse entrer un texte au clavier et que ce texte soit sauvegardé dans un fichier. L'utilisateur indiquera que le texte est terminé en tapant '$' sur une seule ligne.

Exemple : Programme MON_TEXTE

```
program MON_TEXTE;
var
  f: Text;
  s: string;
begin
  assign(F, 'montexte.txt' );
  rewrite(f);
  writeln(' Tapez un texte et terminez par $' );
  readln(s) ;
  while (s <> '$') do
  begin
    writeln(f,s);
    readln(s);
  end;
  close(f);
end.
```

4. Lecture d'un fichier texte

Cette fois le texte est stocké dans le fichier « montexte.txt » et le programme doit l'afficher à l'écran.

Exemple : Programme Lecture

```
program Lecture;
var
  f: Text;
  s: string;
begin
  assign(f, 'montexte.txt' );
  reset(f);
  while not eof(f) do
  begin
    readln(f,s);
    writeln(s);
  end;
  close(f);
end.
```

Chapitre 12 - Récursivité

XII

Principe	109
Exemple complet : Somme des n premiers entiers	110
Exemples simples	112
Exemple complet : Tours de HANOI	113

A. Principe

1. Définition

Définition

Une fonction ou une procédure est dite **récursive** s'il est fait appel à cette fonction ou à cette procédure dans le corps d'instructions qui la définit.

En d'autres termes, la fonction (ou la procédure) s'appelle elle-même.

2. Exemple de programmation itérative

Exemple : La fonction factorielle

Soit n un nombre entier :

```
    n! = 1 * 2 * ... * (n - 1) * n
```

Ceci est une définition **itérative**, car il faut utiliser une boucle pour réaliser l'algorithme associé.

La fonction Pascal correspondante est :

```
function factorielle(n:integer):longint;
var
   i : integer;
   fact : longint;
begin
   fact:=1;
   for i := 1 to n do
      fact := fact * i;
   factorielle := fact;
end;
```

3. Exemple de programmation récursive

Exemple : La fonction factorielle

Une définition récursive de la fonction factorielle est :

```
    n! = n * (n - 1) !
```

avec une condition d'arrêt qui est : **1! = 1**.

Voici la fonction Pascal correspondante :

```
function factorielle(n:integer):longint;
begin
   if (n > 1) then
      factorielle:= n * factorielle (n-1)
   else
      factorielle:= 1;
end;
```

B. Exemple complet : Somme des n premiers entiers

1. Objectif

Exemple

On désire calculer la somme des n premiers entiers naturels :

```
    S(n)=1+2+3+...+n
```

Syntaxe

On peut alors facilement écrire la fonction itérative correspondante :

```
function somme(N:integer):longint;
var
   i : integer;
   sum : longint;
begin
   sum:=0;
   for i := 1 to N do sum := sum + i;
   somme := sum;
end;
```

Mais on peut aussi définir cette fonction de façon **récursive**.

2. Réalisation grâce à la récursivité

Méthode

Pour définir cette fonction de façon **récursive**, on utilise le fait que :

```
    S(n) = S(n - 1) + n
    S(0) = 0 pour la condition d'arrêt
```

Syntaxe

Voici le programme complet :

```
program SommePremiersEntiers ;
var
   n:integer;
function somme ( n : integer ) : integer ;
begin
if n = 0 then
   somme:= 0
else
   somme:= somme (n - 1) + n;
end;
begin
   readln(n);
   writeln(somme (n));
end ;
```

3. Remarque et exemple d'exécution

Remarque

La fonction *«somme»* est une fonction **récursive** car la définition de *«somme»* utilise *«somme»* elle-même.

Lors de l'exécution du programme, à chaque appel de la fonction somme, le système va effectuer un **empilement** (sauvegarde) des valeurs paramètres et des résultats de calculs.

A la sortie de la fonction, ces valeurs seront restituées (on dit qu'elles sont **dépilées**) et réinjectées dans le programme.

Exemple : Exemple d'exécution du programme avec N=4

```
début
   appel de SOMME avec N = 3
      début
         appel de SOMME avec N = 2
            début
               appel de SOMME avec N = 1
                  début
                     appel de SOMME avec N = 0
                        début
                           SOMME <- 0
                        fin
                        SOMME <- SOMME(0) + 1
                  fin
                  SOMME <- SOMME(1) + 2
            fin
            SOMME <- SOMME(2) + 3
      fin
      SOMME <- SOMME(3) + 4
   fin
   Résultat : SOMME(4) = 10
Fin
```

C. Exemples simples

1. Puissance d'un entier

Rappel

On peut écrire une fonction récursive pour calculer la puissance d'un entier en utilisant :

$x^n = x * x^{n-1}$ si n > 0

$x^n = x^{n+1} / x$ si n < 0

avec la condition d'arrêt: $x^0 = 1$

Syntaxe

```
program Puiss;
var
   x : real ;
   p: integer ;
function Puissance (x: real ; n : integer ) : real ;
begin
   if n = 0 then Puissance := 1
else
   if n > 0
      then Puissance:= Puissance (x, n - 1) * x
      else Puissance:= Puissance (x, n + 1) / x
end;
begin
   write('Nombre :');
   readln(x);
   write('Exposant entier :');
   readln(p);
   writeln('Le résultat est :', Puissance (x, p) ) ;
end ;
```

2. PGCD Récursif

Rappel

On utilise le résultat suivant :

PGCD (a , b) = PGCD (b , r) si r <> 0, avec r = a mod b

(r est le reste de la division euclidienne de a par b)

Méthode

Condition d'arret :

PGCD (a , b) = b si r = 0

Exemple

PGCD (20 , 6) = PGCD (6 , 2) = 2

Syntaxe

Le programme sera donc le suivant :

```
program PgcdRec;
var
   a, b : integer
function PGCD (i,j : integer ) : integer ;
begin
   if j = 0 then
      PGCD := i
   else
      PGCD := PGCD (j , i mod j) ;
end;
begin
   readln (a , b) ;
   writeln ('Le PGCD est :', PGCD (a , b) ) ;
end ;
```

3. Exercice d'application

Vous devez deviner à quoi correspond le programme suivant, et quel sera le résultat à la fin de son déroulement.

Exemple

```
program A_TROUVER ;
const
   POINT = '.' ;
procedure FAIRE ;
var
   car : char ;
begin
   read (car) ;
   if car <> POINT then FAIRE ;
   write (car) ;
end;
begin
   writeln ('Entrez un texte') ;
   FAIRE ;
   writeln;
end ;
```

D. Exemple complet : Tours de HANOI

1. Problème

Exemple

Il s'agit de déplacer la pile de disques de la tour 1 à la tour 3; en ne déplaçant qu'un disque à la fois, et en s'assurant qu'à aucun moment un disque donné ne repose sur un disque de plus petit diamètre.

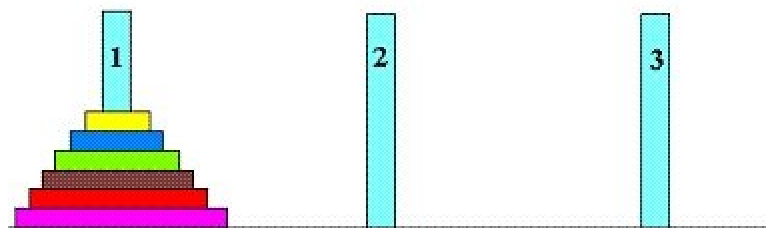


Image 36 : Situation initiale

2. Analyse

Conseil

On va imaginer, afin de ramener le problème à un nombre de disques plus petit, que l'on sait faire passer tous les disques sauf le dernier, d'un pilier à un autre. On a donc diminué de un le nombre de disques à déplacer.

Rappel

C'est le principe de la **récursivité** : on va définir le problème à partir d'un problème similaire d'ordre inférieur (comme pour la fonction factorielle n!=n*(n-1)! ou la fonction puissance $x^n = x * x^{n-1}$).

Méthode

Pour déplacer n disques de la tour 1 à la tour 3 :

- déplacer n - 1 disques de la tour 1 à la tour 2 *«(on suppose qu'on sait le faire)»*
- déplacer 1 disque de la tour 1 à la tour 3 *«(on sait le faire : il ne reste plus qu'un seul disque)»*
- déplacer n - 1 disques de la tour 2 à la tour 3 *«(même supposition qu'au début : on sait le faire)»*

3. Programme

Syntaxe

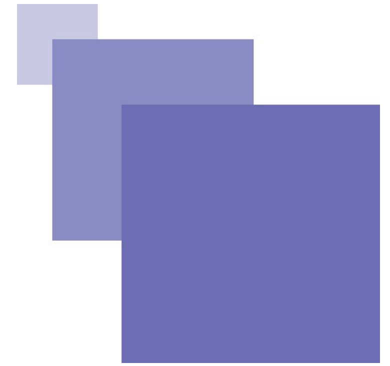
```
program HANOI ;
var
   nbDisques : integer ;
procedure deplacer(nDisk, trOrig, trDest, trInterm:
integer);
begin
   if nDisk > 0 then
   begin
      deplacer (nDisk - 1 , trOrig, trInterm, trDest) ;
      deplacer ('Déplacer le disque de ', trOrig,'
à',trDest);
      deplacer (nDisk - 1 , trInterm, trDest, trOrig) ;
   end;
end;
```

```
begin
   writeln('Entrez le nombre de disques :');
   readln(nbDisques);
   deplacer (nbDisques, 1 , 3 , 2 );
end ;
```

4. Exemple d'exécution

Exécution pour N = 4

```
Déplacer le disque de1 à 2
Déplacer le disque de1 à 3
Déplacer le disque de2 à 3
Déplacer le disque de1 à 2
Déplacer le disque de3 à 1
Déplacer le disque de3 à 2
Déplacer le disque de1 à 2
Déplacer le disque de1 à 3
Déplacer le disque de2 à 3
Déplacer le disque de2 à 1
Déplacer le disque de3 à 1
Déplacer le disque de2 à 3
Déplacer le disque de1 à 2
Déplacer le disque de1 à 3
Déplacer le disque de2 à 3
```



Conclusion

Nous espérons que ce premier contact avec l'algorithmique et la programmation vous aura donné envie d'approfondir ces notions.

D'autres UV d'informatique vous seront proposées, aussi bien en tronc commun que dans toutes les branches de l'UTC. Elles s'appuient toutes sur l'enseignement fondamental dispensé en NF01.

Vous pouvez approfondir vos connaissances et vous préparer aux examens à travers des compléments de cours, des exercices, des annales d'examens, des QCM, jeux ou vidéos pédagogiques, disponibles sur le site web de l'UV : *http://www4.utc.fr/~nf01*[1]

N'oubliez pas qu'en informatique, il faut beaucoup de pratique pour commencer à maîtriser les concepts de base : « c'est en forgeant qu'on devient forgeron ! »

1 - http://www4.utc.fr/~nf01